Real-Time Data Semantics and Similarity-Based Concurrency Control

Tei-Wei Kuo, *Member*, *IEEE*, and Aloysius K. Mok, *Member*, *IEEE*

Abstract—This paper formalizes the concept of *similarity* which has been used on an ad hoc basis by application engineers to provide more flexibility in concurrency control. We show how the usual correctness criteria of concurrency control, namely, final-state, view, and conflict serializability, can be weakened to incorporate similarity. We extend the weakened correctness criteria in [16] for real-time applications which may run continually, have concurrent transaction executions, or skip unimportant computations. A semantic approach based on the similarity concept is then taken to propose a sufficient condition for scheduling real-time transactions without locking of data.

Index Terms—Concurrency control, schedule correctness, real-time database, serializability, similarity.

1 Introduction

While past works in database performance stress mostly on throughput, there is increasing interest in the performance of transaction systems that have significant response time requirements. These requirements are usually posed as hard or soft deadlines on individual transactions so that a concurrency control algorithm must attempt to meet deadlines, as well as preserve database consistency. A number of analytic and simulation studies on the performance of scheduling algorithms to meet deadlines have been reported in the literature, e.g., [1], [2], [13], [27], [30], [34], [37], [38], [36], [41]. In these studies, database consistency is preserved by enforcing serializability. However, serializability is often too strict a correctness criterion for real-time applications, where the precision of an answer to a query may still be acceptable even if serializability is not strictly observed in transaction scheduling. Obviously, violation of serializability must be justified in the context of the semantics of the application domain. The subject of this paper is to explore a weaker correctness criterion for concurrency control in real-time transactions by investigating the notion of *similarity*.

Similarity is closely related to the important idea of imprecise computation in real-time systems [25] and also to the idea of partial computation for databases [8]. 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 change in the sensor reading over a short time interval. For certain computations, engineers often consider the change in sensor reading over a few consecutive cycles to be insignificant in the execution of the avionic software. It is sometimes acceptable to use a sensor value that is not the most recent update in a transaction. This suggests that serializability can be weakened in concurrency control of real-time transactions. *However, it is imperative for us to justify and make explicit the implicit assumptions that are behind the currently ad hoc engineering practice*. They can be the cause of costly errors.

Real-time databases should model the real world with sufficient precision. Several consistency requirements besides *internal consistency* [4] have been introduced in [15], [16], [20], [30], [26], [36], [41]. *External consistency* requirements keep real-time databases up-to-date; *temporal consistency* requirements ensure that multiple data objects read by a transaction are compatible in the currency of their data. External and temporal consistency constraints are design specifications that are introduced to cope with the dynamic nature of the operating environment, inasmuch as a real-time database can capture the values of real-world objects only up to a certain precision. Hence, the consistency constraints on real-time databases are inherently concerned with imprecise values. Intuitively, data values that are sufficiently close by some metric may be interchanged as input to a transaction without undue adverse effects. This motivates the concept of similarity among data values. The satisfaction of external and temporal consistency constraints ensures that such interchanges are admissible, but the adequacy of the consistency constraints must be justified by the semantics of data similarity.

In this paper, we formalize the concept of *similarity*, which has been used on an ad hoc basis by application engineers to provide more flexibility in concurrency control. We show how the usual correctness criteria of concurrency control, namely, final-state, view, and conflict serializability, can be weakened to incorporate similarity. We then generalize the weakened correctness criteria in [16] for infinite schedules and extend the criteria when true parallelism, instead of interleaving, is considered. We propose the idea of physical schedules in which a real-time database scheduler may skip unimportant computation or updates to meet time constraints and/or satisfy some safety requirements on the system. The correctness of physical

- *T.-W. Kuo is with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106, ROC. E-mail: ktw@csie.ntu.edu.tw.*
- *A.K. Mok is with the Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712. E-mail: mok@cs.utexas.edu.*

Manuscript received 5 Aug. 1997; revised 11 Aug. 1998; accepted 20 Sept. 2000.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 105465.

schedules is justified by the notion of similarity. We then take a semantic approach based on the similarity concept to propose a sufficient condition for scheduling real-time transactions without locking of data.

The rest of this paper is organized as follows: In Section 2, we define the system model and formally introduce the concept of *similarity*. In Section 3, we show how the traditional correctness criteria of concurrency control can be weakened to incorporate similarity. In Section 4, we extend the proposed weaker consistency requirements [16] on infinite schedules, parallel executions, and physical schedules. Section 5 describes a sufficient condition with which transactions can be scheduled independently. We then further extend the results. Section 6 is the conclusion.

2 Semantics of Real-Time Transactions

2.1 Data Objects, Events, Transactions, and Schedules

A real-time database is a collection of data objects. Each data object takes its value from its *domain*. We define a database state as an element of the Cartesian product of the domains [29] 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.

Events are primitive database operations (read or write). 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. To distinguish between a transaction and an instance of it, we shall use the notation $\tau_{i,j}$ to denote the jth instance of transaction τ_i. An *interpretation* of a set of transactions is a collection of transaction definitions and data domain definitions [29].

A *schedule* over 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. In this paper, schedules are represented by sequences of events that are consistent with the partial order of the schedule. A *serial schedule* is a sequence of transaction instances (i.e., a schedule in which the transaction instances are totally ordered).

2.2 Timed Events and Timed Schedules

A real-time computation may be represented as a collection of events with time-stamps. The time-stamp of each event in the computation indicates its occurrence time. Events with such time-stamps are called *timed events*. In other words, a real-time computation is a collection of timed events.

Let a *timed schedule* over a set of transactions be a collection of timed events issued by instances of the transaction set. Each event in a timed schedule is issued by one transaction instance. The time ordering of events in a timed schedule must be consistent with the partial order of the corresponding events as specified by each transaction. 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.

A real-time computation, such as an engine warming-up procedure, is a timed schedule. It should be not only logically correct, but also in timely fashion. The timeliness of events can be properly checked regarding their occurrence times, whereas the logical correctness of the computation must be justified through a careful examination of its corresponding (untimed) schedule, which preserves the time-stamp order of events in the computation. In this paper, we shall devote our efforts in justifying the logical correctness of real-time computations.

2.3 Similarity

A real-time database models an external environment that changes continuously. The value of a data object that models an entity in the real world cannot in general be updated continually to perfectly track the dynamics of the real-world entity. The time needed to perform an update alone necessarily introduces a time delay which means that the value of a data object cannot be instantaneously the same as the corresponding real-world entity. Fortunately, it is often unnecessary for data values to be perfectly up-to-date or precise to be useful. In particular, data values of a data object that are slightly different in age or in precision are often interchangeable as read data for transactions. This observation underlies the concept of similarity among data values. The concept of *similarity* can be described in terms of regions in the state space of a database.

As an example, let us consider the similarity of data read by two transactions in a railroad-crossing monitor system. Suppose there are two data objects, $distance$ and $velocity$, which provide information about the nearest approaching train. In Fig. 1, s is a database state (a point) in the database state space of the system. Let τ_1 be a transaction that displays the $distance$ and $velocity$ of the approaching train on the monitor. The other transaction, τ_2, controls the crossing gate, which depends only on the $distance$ of the train. With different precision requirements, τ_1 and τ_2 regard data values falling inside, respectively, τ_1's box and τ_2's box to be similar to their counterparts in the state s. Notice that, because τ_2 does not read $velocity$, all values in the domain of $velocity$ are similar to one another and, therefore, similar to that at s. In our model, two values of a data object are similar if and only if all transactions that may read them consider them as similar.

2.3.1 Definition of Similarity

Similarity is a binary relation on the domain of a data object. Every similarity relation is reflexive and symmetric, but not necessarily transitive. Different transactions can have 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.

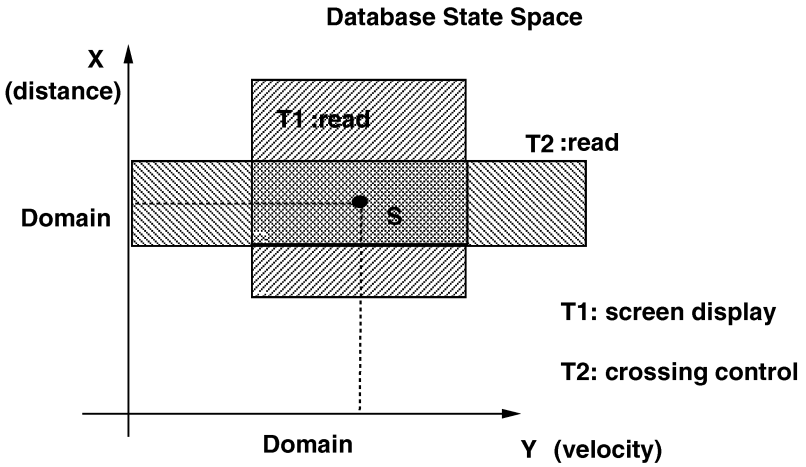

Fig. 1. Similarity among data/database states.

As one might expect, there is no general criterion for determining whether two values are similar. More often than not, proximity in data value alone may not be a good criterion. For example, a temperature of $99°C$ is equidistant from $98°C$ and $100°C$. Whereas one might consider $99°C$ to be similar to $98°C$ as far as hotness of water is concerned, there is a qualitative difference between $99°C$ and $100°C$ because water turns into steam at $100°C$. In general, similarity need not be transitive if the proximity of values is defined in terms of "magnitude difference." However, some mappings such as "rounding" or "=" that may be used in establishing similarity do yield transitive similarity relations.

Similarity is an inherently application-dependent concept and we expect the application engineer to define it for specific applications. Similarity can be defined by explicit declaration or other syntatic conventions. For example, a transaction might have associated with it a set of parameters to specify read-data similarity. A related approach is found in [31].

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 τ 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 Correctness Criteria

3.1 Related Work

Previous work on real-time databases can be roughly classified into three types: semantics and requirements of real-time databases, design of real-time transaction scheduling algorithms, analytic and experimental studies of concurrency control protocols with deadlines. A compendium of recent approaches can be found in [3], [35].

In conventional databases, the notion of correctness of a schedule has mainly been based on the concept of *serializability* [29]. Three increasingly restrictive criteria for correctness are commonly accepted and have been studied in depth. They are: final-state serializability, view serializability, and conflict serializability [29]. Other different correctness criteria have been proposed for different purposes and application areas [7], [10], [11], [14], [21], [30], [31], [33]. We list some of them below.

Several new consistency requirements besides *internal consistency* [29] have been discussed [20], [22], [26], [30], [36] in relation to real-time systems. Song and Liu [36] also evaluate the effectiveness of multiversion lock-based concurrency control algorithms in maintaining the temporal homogeneity of shared data. New real-time transaction scheduling algorithms are proposed in [2], [20], [27], [30], [34], [39], [41]. In particular, Xiong et al. [41] exploited temporal data similarity and evaluated the performance improvement of transactions when combinations of similarity and forced wait policies were considered. A force wait policy may force a transaction to delay further execution until a new version of sensor data becomes available.

Garcia-Molina and Wiederhold in [11] discarded consistency considerations for read-only transactions, with the stipulation that, after read-only transactions have been removed, the resulting schedule should be serializable. Garcia-Molina and Salem [10] also proposed "SAGAS" so as to solve consistency problems brought on by long-lived transactions. SAGAS are long-lived transactions that can be broken up into a collection of subtransactions that can be interleaved in any way with other transactions. Thus, SAGA is not atomic, but should be executed as a unit. It means that correct schedules can be nonserializable.

Peng and Lin proposed the idea of compatibility matrix to allow transactions to acquire different degrees of consistency requirements [30]. Their work was motivated by avionic systems and automated factories that have a limited number of high-speed sensors with frequent user-initiated command processing. The rationale behind their work was that the consistency between the device readings and the current values used by transactions could be more important than the serializability of transactions.

Korth and Speegle [21] proposed a formal model which allows transactions to specify preconditions and

postconditions. These conditions can be specified in conjunctive normal form. They enforced serializability with respect to every conjunct in the conjunctive normal form by a criterion called *predicatewise serializability*. Their model also includes consideration of nested transactions and multiple versions.

Epsilon-serializability (ESR) [31], [33] formalizes the query behavior by deriving the formulae that express the inconsistency in the data values read by a query. Transactions are associated with limits of importing inconsistency and exporting inconsistency. Query transactions are allowed to view inconsistent data in a controlled fashion. Kamath and Ramamritham [14] then introduced the idea of hierarchical inconsistency bounds that allows inconsistency to be specified at different granularities such as transactions and objects. They provided mechanisms to control the inconsistency and reported the evaluation of the performance improvement due to ESR.

Our proposed criteria can be viewed as extensions of the standard correctness criteria to exploit the concept of similarity. As a result of our extension, it is possible to permit much more concurrency for updates that are close in time.

3.2 View Similarity and Δ-Serializability

Our proposed criteria can be viewed as extensions of the standard serializability-based correctness criteria [29] to exploit the concept of similarity. Three correctness criteria defined in [16] are final-state, view, and conflict Δ-serializability. The definition of final-state Δ-serializability can be found in [16].[1]

The transaction view of a transaction instance is a vector of data object values such that the ith component is the value read by the ith read event of the transaction instance [29].

Definition: 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,*[2] *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. The schedule

$$\pi_1 = W(\tau_{3,1}, X), R(\tau_{1,1}, X), W(\tau_{1,1}, X), R(\tau_{2,1}, X), \rightarrow R(\tau_{2,1}, Y), W(\tau_{2,1}, Y), W(\tau_{1,1}, Y)$$

is view similar to the schedule

$$\pi_2 = \tau_{3,1}, \tau_{2,1}, \tau_{1,1} = W(\tau_{3,1}, X), R(\tau_{2,1}, X), R(\tau_{2,1}, Y), W(\tau_{2,1}, Y), \rightarrow R(\tau_{1,1}, X), W(\tau_{1,1}, X), W(\tau_{1,1}, Y)$$

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.

In general, we can show the following theorem:

Theorem 1. *Given a polynomial-time procedure for determining if any two states are similar, the problem of deciding whether a schedule is view Δ-serializable is NP-Hard.*

Proof. Follows directly from the view serializability problem [29]. □

3.3 Conflict Similarity and Δ-Serializability

The NP-hardness of determining view Δ-serializability motivates us to find a stronger correctness criterion. In this section, we shall extend the notion of conflict serializability to include the similarity concept. Unlike view serializability, the extension to conflict Δ-serializability must deal with a new problem, namely, the intransitivity of some similarity relations in order to define equivalent relations (i.e., relation $free(\pi)$ in Section 3.3.2) of "similar" events. The definition of such equivalent relations is essential in defining the equivalent relations of "similar" conflict Δ-serializable schedules.

3.3.1 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, in which case it is an equivalence relation, 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. In this section, 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., they 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

1. A schedule is *final-state similar* to another schedule if the first two conditions of the following view similarity definition are satisfied.

2. An *interpretation* of a set of transactions is a collection of transaction definitions and data domain definitions [29].

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.

For example, suppose we want to compute the bearing of a vehicle to a reference point and this distance data can be computed with acceptable precision only if the position data is accurate to within 10 meters, where the position data is updated from a sensor every second. Assumptions are often made in practice that the time derivative of the position data is bounded. Thus, if we need a precision of 10 meters and every sample of the position data differs from the previous one by no more than 1 meter, then it is acceptable to use any one of the last 10 samples, assuming that sensor updates are perfectly accurate. This idea can be captured in terms of the data dependency of an event in a schedule as follows:

Recall that a schedule over a transaction set is a partial order of events issued by the transaction instances in the set. Given a schedule π, we define its *data dependency graph* $G(\pi)$ as follows: Corresponding to every read/write event in π is a read/write node in $G(\pi)$. There is a directed edge(arc) from a read node to a write node in $G(\pi)$ if they are issued by the same transaction instance and the read event precedes the write event in π. There is a directed edge from a write node to a read node in $G(\pi)$ if the corresponding read event reads the output value of the corresponding write event in π. We note that data dependency graphs are acyclic since the time-stamp of the node at the head of an edge is always greater than that of the node at the tail.

We define the *w-length* of a path in a data dependency graph to be k if there are exactly k write nodes in the path, $k \geq 1$. A path with infinitely many write nodes in a data dependency graph has w-length ∞. We say that a data dependency graph has *depth* k if the maximum w-length of its paths is k. A schedule has *ddg depth* k if its data dependency graph has depth k.

Suppose Δ is a similarity relation and Δ^i is the ith power of Δ, $i \geq 1$. (Since Δ is reflexive, $\Delta^i \subseteq \Delta^j$ if $i \leq j$.) We define a similarity relation $\Delta_\pi^\#$ with respect to a schedule π as follows:

1. If the ddg depth of π is ∞, then $\Delta_\pi^\#$ is defined to be Δ^*, the transitive closure of Δ.
2. If the ddg depth of π is k, $k \geq 1$, then $\Delta_\pi^\#$ is defined to be Δ^{k+1}.

We say that a schedule π *strongly preserves* a similarity relation Δ (or Δ is a *strong similarity* relation for a schedule π) if every transaction that has an instance in π preserves $\Delta_\pi^\#$ (i.e., for every transaction τ that has an instance in π, if τ maps database state s to t and state s' to t', then t and t' are related by $\Delta_\pi^\#$ if s and s' are related by $\Delta_\pi^\#$).

In this paper, we shall assume that every similarity relation Δ is strongly preserved by all serial schedules of the transaction set in a system. In practical real-time systems, this assumption can often be weakened since the schedules that are produced by a system are often reducible to only a subset of the set of all serial schedules of the transaction set of the system (e.g., as a result of the time constraints imposed by the application). The results reported herein can be strengthened by requiring a similarity relation to be strongly preserved by only those serial schedules that are reduced from the set of schedules produced by the system (e.g., the time constraints may be such that each update on an object is an instance of a transaction that modifies the value of an object in small increments). Without making use of application-specific information, the above assumption is the weakest we can make. From now on, we shall refer to a strong similarity relation with the assumption that it is strongly preserved by all serial schedules.

We say that two data values v_1, v_2 are *weakly similar* under a similarity relation Δ with respect to a schedule π if π strongly preserves Δ and v_1, v_2 are related by $\Delta_\pi^\#$. When there is no ambiguity, we shall say that v_1, v_2 are weakly similar and omit the reference to Δ and π. Notice that two values that are similar must be weakly similar, but the converse is not necessarily true.

Let $RE(\pi)$ be the set of all read events in a schedule π and $WE(\pi)$ be the set of all write events in π. Suppose $init$ is a distinct write event which creates the initial state of schedule π. Let DB be the set of data objects in the database under discussion. $DSPACE$ is the space of all possible database states. Suppose $domain(obj)$ denotes the domain of data object obj and $access(e)$ is the data object accessed by event e. We define three functions on the events in schedule π as follows:

- $Wr_\pi : RE(\pi) \rightarrow WE(\pi) \cup \{init\}$: $Wr_\pi(e_r)$ returns the write event from which a read event e_r reads in π. If e_r reads from the initial state, $Wr_\pi(e_r)$ returns $init$.
- $Upd_\pi : DB \rightarrow WE(\pi) \cup \{init\}$: $Upd_\pi(obj)$ returns the last write event which updates the data object obj in π. If no write event accesses obj, $Upd_\pi(obj)$ returns $init$.
- $Out_\pi : (DSPACE, WE(\pi)) \rightarrow domain(access(e_w))$:

  Suppose $istate$ is an initial database state of schedule π. $Out_{\pi_i}(istate, e_w)$ returns the write-value of e_w (the value of the accessed data object $access(e_w)$) in schedule π if π starts with initial state $istate$.

When there is no ambiguity, we shall use $Out_\pi(e_w)$ and omit the reference to the initial state. Given any similarity relation Δ, we write $x \approx_\Delta y$ iff two data values x and y of the same data object are defined and are similar under Δ. For convenience, let $Out_\pi(init) = Out_{\pi'}(init)$ for any two schedules π and π' (although the special event $init$ is not in any schedule).

With the introduction of strong similarity relation, the definitions of final-state and view Δ-serializability ought to be modified. Two schedules are final-state similar if they transform strongly similar states (under Δ) into similar states (under $\Delta^\#$). Corresponding transaction views in two view-similar (under $\Delta^\#$) schedules remain similar (under

$\Delta^{\#}$). These modifications enlarge the collection of schedules which are final-state or view similar to a given schedule. Notice that requiring final-state similar schedules to transform similar states (under $\Delta^{\#}$) into similar states (under $\Delta^{\#}$) unnecessarily restricts a class of final-state or view similar schedules and reduces its applicability in practice.

Suppose the schedules π and π' have the same event set E and Δ is a strong similarity relation for π and π'. We say that π' is a *derived* schedule of π if

$$\forall e_r \in E, Out_{\pi}(init, Wr_{\pi}(e_r)) \approx_{\Delta} Out_{\pi}(init, Wr_{\pi'}(e_r))$$

and

$$\forall obj \in DB, Out_{\pi}(init, Upd_{\pi}(obj)) \approx_{\Delta} Out_{\pi}(init, Upd_{\pi'}(obj)),$$

where initial state $init$ can be omitted for convenience. (DB is the set of data objects in the database.)

Suppose k is the maximum of the ddg depths of π and π' and let $\Delta^{\#} = \Delta^{k+1}$. We shall show that π and π' are view-similar under the relation $\Delta^{\#}$. That is, given two strongly similar initial states $init$ and $init'$,

$$\forall e_r \in E, Out_{\pi}(init, Wr_{\pi}(e_r)) \approx_{\Delta^{\#}} Out_{\pi'}(init', Wr_{\pi'}(e_r))$$

and

$$\forall obj \in DB, Out_{\pi}(init, Upd_{\pi}(obj)) \approx_{\Delta^{\#}} Out_{\pi'}(init', Upd_{\pi'}(obj)).$$

In other words, suppose we swap the events in a schedule π and obtain another schedule π'. If every one of the read events in π' reads from write events that are similar under Δ in π, then the swapped schedule π' will be view-similar to π under the similarity relation $\Delta^{\#}$. Thus, all we need to do to maintain consistency is to ensure that the rules for swapping events preserve Δ, as we shall do in the next section.

Theorem 2. *Suppose a schedule π' is a derived schedule of another schedule π and $\Delta^{\#} = \Delta^{k+1}$, where k is the maximum ddg depths of π and π', then π and π' are view-similar under $\Delta^{\#}$.*

Proof. Given a real-time database system, suppose DB is a collection of data objects in the database and Δ is a strong similarity relation for schedule π and its derived schedule π'. Let E be their event set and let k be the maximum ddg depth of π and π' so that $\Delta^{\#} = \Delta^{k+1}$. We shall prove that π and π' are view-similar under $\Delta^{\#}$. The proof is by induction on the "depth" of "changed" write events, as defined below.

Let $G(\pi)$ be the data dependency graph of a schedule π. $G_e(\pi)$ is a subgraph of $G(\pi)$ which consists of an event e and all ancestors of e in $G(\pi)$. An event e in the event set E is *unchanged* (with respect to π and π') if it satisfies any one of the following conditions: (See Fig. 2). Otherwise, it is *changed*.

1. e is a read event. e reads from initial state in both π and π'.
2. e is a write event. No preceding read event is issued by the same transaction instance.
3. e is a read event. e reads from the same write event e_w in π and π' and e_w is *unchanged*.

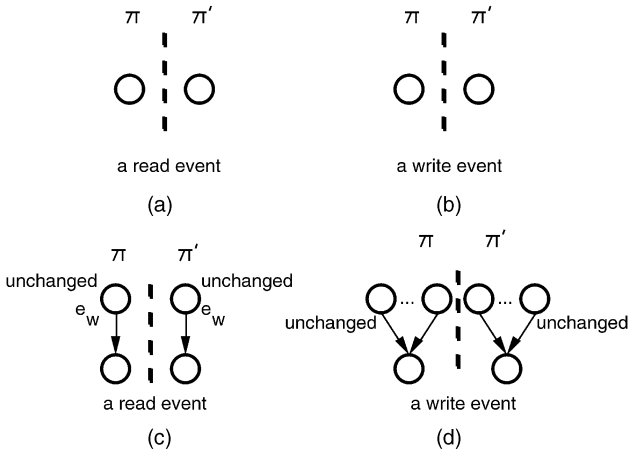

Fig. 2. Unchanged events.

4. e is a write event and all preceding read events of the same transaction instance are *unchanged*.

Suppose $init$ and $init'$ are two strongly similar initial states, where $init$ and $init'$ are initial states for schedules π and π', respectively. It is trivial to show that an unchanged read event receives strongly similar inputs in π and π'. An unchanged write event produces strongly similar outputs in π and π'. (It is because every unchanged event has the same data dependence graph in π and π' and transactions preserve the strong similarity relation.)

Let e_w be a write event in E. e_w^1 is the *changed* write event which has the maximum w-length between itself and e_w in $G_{e_w}(\pi')$. (If there is more than one candidate for e_w^1, select one arbitrarily and apply the same argument below.) Let m be the w-length of the path from e_w^1 to e_w minus one. If $e_w^1 = e_w$, $m = 0$. If e_w^1 does not exist, e_w is an *unchanged* write event and produces strongly similar outputs in π and π'.

We argue that $m \leq (k-2)$ (if e_w^1 exists). Suppose $L = e_1, e_2, \cdots, e_n$ is a maximum w-length path in $G(\pi')$ where there is an arc from e_i to e_{i+1} for $i < n$. We shall first show that the first write event closest to e_1 (it is e_1 if e_1 is a write event) on L is *unchanged*. Notice that read events and write events alternate on any path in $G(\pi')$.

If e_1 is a write event and there exists an arc from a read event e to e_1 in $G(\pi')$, e must be *unchanged* and, thus, e_1 is *unchanged* because L is a path with the maximum w-length. If e_1 is a write event and no preceding read event is issued by the same transaction instance, e_1 is *unchanged* according to the "*unchanged*" definition. Suppose e_1 is a read event and e_2 is a write event. e_1 is *unchanged* because L is a path with the maximum w-length. Suppose there is another read event e_r having an arc from e_r to e_2. e_r is also *unchanged* with the same argument for the read event e_1. Since all preceding read events issued by the same transaction instance are *unchanged*, e_2 must be *unchanged*.

The above arguments ensure that the first write event closest to e_1 on L is *unchanged*. Given a write event e_w and its e_w^1 on L, m, which is the w-length of the path from e_w^1 to e_w minus one, is no larger than $(k-2)$ because e_w^1,

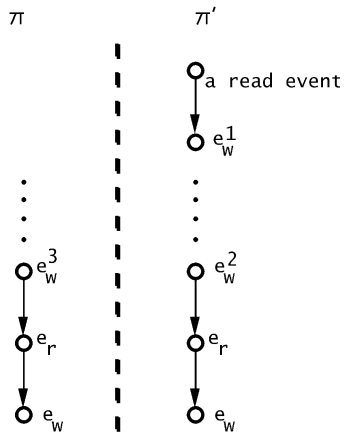

Fig. 3. $e_w^1 - e_w^2 - e_w$.

which must be *changed*, cannot be the first write event closest to e_1 on L. Since L has the maximum w-length in $G(\pi')$, given any write event e_w and its e_w^1 on any path, $m \leq (k-2)$.

If e_w^1 exists, we shall show that, when $m \leq (k-2)$ for $m \geq 0$, $Out_\pi(init, e_w) \approx_{\Delta^{m+2}} Out_{\pi'}(init', e_w)$. The proof is by induction on m.

Induction basis: $m = 0$ and $e_w^1 = e_w$. Because e_w is changed, some e_r of the preceding read events of the same transaction instance is *changed*. Because $m = 0$, $Wr_{\pi'}(e_r)$ is unchanged, i.e.,

$$Out_{\pi'}(init', Wr_{\pi'}(e_r)) \approx_\Delta Out_\pi(init, Wr_{\pi'}(e_r)).$$

Because of the assumption

$$\forall e_r \in E, Out_\pi(init, Wr_\pi(e_r)) \approx_\Delta Out_\pi(init, Wr_{\pi'}(e_r)),$$
$$Out_{\pi'}(init', Wr_{\pi'}(e_r)) \approx_{\Delta^2} Out_\pi(init, Wr_\pi(e_r)).$$

That is, $Out_{\pi'}(init', e_w) \approx_{\Delta^2} Out_\pi(init, e_w)$.

Induction step: The induction hypothesis assumes that, when $m \leq i$ for some $0 \leq i \leq (k-3)$, $Out_\pi(init, e_w) \approx_{\Delta^{m+2}} Out_{\pi'}(init', e_w)$. Let $m = i+1$. Suppose e_w^2 is the write event closest to e_w on the path from e_w^1 to e_w, as shown in Fig. 3. Let a read event e_r be between e_w and e_w^2. By the induction hypothesis, $Out_\pi(init, e_w^2) \approx_{\Delta^{i+2}} Out_{\pi'}(init', e_w^2)$. Let a write event $e_w^3 = Wr_\pi(e_r)$, as shown in Fig. 3. If $e_w^3 = e_w^2$, then $Out_\pi(init, Wr_\pi(e_r)) \approx_{\Delta^{i+2}} Out_{\pi'}(init', Wr_{\pi'}(e_r))$ because $Out_\pi(init, e_w^2) \approx_{\Delta^{i+2}} Out_{\pi'}(init', e_w^2)$. If $e_w^3 \neq e_w^2$, then $Out_\pi(init, e_w^2) \approx_\Delta Out_\pi(init, e_w^3)$ because of the assumption

$$\forall e_r \in E, Out_\pi(init, Wr_\pi(e_r)) \approx_\Delta Out_\pi(init, Wr_{\pi'}(e_r)).$$

Since

$$Out_\pi(init, e_w^2) \approx_{\Delta^{i+2}} Out_{\pi'}(init', e_w^2),$$
$$Out_\pi(init, e_w^3) \approx_{\Delta^{i+3}} Out_{\pi'}(init', e_w^2),$$

i.e., $Out_\pi(init, Wr_\pi(e_r)) \approx_{\Delta^{i+3}} Out_{\pi'}(init', Wr_{\pi'}(e_r))$. Since e_w^1 is the *changed* write event which has the maximum w-length between itself and e_w in $G_{e_w}(\pi')$, for every preceding read event e_r of the same transaction instance, $Out_\pi(init, Wr_\pi(e_r)) \approx_{\Delta^{i+3}} Out_{\pi'}(init', Wr_{\pi'}(e_r))$. Because transactions preserve these similarity relations, $Out_\pi(init, e_w) \approx_{\Delta^{i+3}} Out_{\pi'}(init', e_w)$.

Since, when $m \leq (k-2)$ for $m \geq 0$,

$$Out_\pi(init, e_w) \approx_{\Delta^{m+2}} Out_{\pi'}(init', e_w)$$

and $m \leq (k-2)$,

$$Out_\pi(init, e_w) \approx_{\Delta^k} Out_{\pi'}(init', e_w)$$

for any write event e_w in E. Furthermore, because $\forall e_r \in E, Out_\pi(init,\ Wr_\pi(e_r)) \approx_\Delta Out_\pi(init, Wr_{\pi'}(e_r))$ and

$$\forall obj \in DB, Out_\pi(init, Upd_\pi(obj)) \approx_\Delta Out_\pi(init, Upd_{\pi'}(obj)),$$
$$\forall e_r \in E, Out_\pi(init, Wr_\pi(e_r)) \approx_{\Delta^{k+1}} Out_{\pi'}(init', Wr_{\pi'}(e_r))$$

and

$$\forall obj \in DB, Out_\pi(init, Upd_\pi(obj)) \approx_{\Delta^{k+1}} Out_{\pi'}(init', Upd_{\pi'}(obj)).$$

Therefore, from strongly similar initial states (under Δ), the transaction view of any transaction instance and the final states of π and $'$ are similar (under $\Delta^\#$). □

3.3.2 Conflict Δ-Serializability

Suppose a schedule π consists of a set E of events and a set T of transaction instances. Let Δ be a given strong similarity relation. A relation $free(\pi)$ over E is a set $\{(e_i, e_j)\}$ of event pairs in which e_i and e_j satisfy any of the following conditions, where $i \neq j$:

1. e_i and e_j do not conflict with each other.
2. e_i and e_j are conflicting write events, but they are similar under Δ.
3. e_i and e_j are conflicting events and one of them is a read event. Suppose e_r and e_w are the read and write events, respectively, and e_r precedes e_w. Suppose e'_w ($e'_w \neq e_w$) is the write event which writes the value read by e_r in π and e_w and e'_w are similar under Δ.

Notice that the $free(\pi)$ relation is a collection of swappable events in schedule π, which can be falsified if a read event and some preceding conflicting write event also satisfy the relation. For example, let events in a schedule $\pi = e_w^1, e_w^2, e_w^3, e_r$ conflict with each another, where e_w^i and e_r are a write event and a read event, respectively. Suppose e_w^2 is strongly similar to both e_w^1 and e_w^3 and $free(\pi) = \{(e_w^1, e_w^2), (e_w^2, e_w^3)\}$. If (e_r, e_w^3) satisfy the $free(\pi)$ relation, then π can be transformed into e_w^2, e_w^1, e_r, e_w^3 by swapping event pairs (e_w^1, e_w^2) and (e_r, e_w^3). As a result, e_r reads from e_w^1, which is not strongly similar to e_w^3 in schedule π.

Suppose π is a schedule and R is a reflexive and symmetric binary relation over the events in π. Let π' be a schedule which has the same events as π. We say that $\pi \rightarrow_R \pi'$ if π' can be obtained from π by changing the order of two consecutive events, e_1 and e_2 in π and $(e_1, e_2) \in R$. Let $\rightarrow_R^*$ be the transitive closure of the relation $\rightarrow_R$. A schedule π_1 is *conflict similar* to another schedule π_n iff

$\pi_1 \rightarrow^*_{free(\pi_1)} \pi_n$. Notice that the relation $free(\pi_1)$ is computed with respect to π_1.

A schedule is *conflict Δ-serializable* iff it is conflict similar to a serial schedule. Obviously, if a schedule is conflict equivalent to another schedule, it is conflict similar to that schedule, but the converse may not hold. The conflict similarity relation is reflexive, but not necessarily symmetric or transitive.

Example 2: Conflict Δ-serializability. The schedule

$$\pi_1 = R(\tau_{1,1}, Y), R(\tau_{2,1}, Y), W(\tau_{1,1}, Y), W(\tau_{1,1}, X), \rightarrow W(\tau_{2,1}, X)$$

conflict similar to the schedule

$$\pi_2 = \tau_{2,1}, \tau_{1,1} = R(\tau_{1,1}, Y), R(\tau_{2,1}, Y), W(\tau_{1,1}, Y), \rightarrow W(\tau_{2,1}, X), W(\tau_{1,1}, X)$$

if $W(\tau_{1,1}, X)$ and $W(\tau_{2,1}, X)$ are strongly similar. Since π_2 is a serial schedule, π_1 is conflict Δ-serializable.

Example 3: View Δ-serializable but not conflict Δ-serializable schedules. The schedule

$$\pi = R(\tau_{1,1}, Y), R(\tau_{2,1}, Y), W(\tau_{1,1}, Y), W(\tau_{1,1}, X), \rightarrow W(\tau_{2,1}, X), W(\tau_{3,1}, X)$$

is view-similar but not conflict-similar to $\tau_{2,1}, \tau_{1,1}, \tau_{3,1}$ unless $W(\tau_{1,1}, X)$ and $W(\tau_{2,1}, X)$ are strongly similar.

In general, we can show the following.

Lemma 1. *A schedule π is conflict similar to another schedule π' iff every two conflicting events e and e' occur in the same order in π and π' unless e and e' satisfy condition 2 or 3 in the definition of the $free$ relation.*

Proof. The only-if part is based on the fact that π' can be obtained from π by swapping every two consecutive events which satisfy the $free$ relation (according to the definition of conflict similarity). Therefore, every two conflicting events e and e' must occur in the same order in π and π' unless e and e' satisfy condition 2 or 3 in the definition of the $free$ relation.

The if part can be proven in a similar way as the proof of conflict equivalence in [29]: Let k be the number of pairs of events of the two schedules which occur in different order in π than in π'. We shall show, by induction on k, that π is conflict similar to π'. If $k = 0$, then π and π' are identical (Induction Base). Let $k = m$ and $m \geq 0$ and π is conflict-similar to π' (Induction Hypothesis). Suppose that $k = m + 1$. Let e_1 and e_2 be two consecutive events which occur in different order in π than in π'. Because every two conflicting events e and e' occur in the same order in π and π', unless e and e' satisfy condition 2 or 3 in the definition of the $free$ relation, e_1 and e_2 can be swapped without violating the definition of conflict similarity. It follows that, after the swapping, there are only m pairs of events of the two schedules which occur in different order in π than in π'. We conclude that π is conflict similar to π'. □

Lemma 2. *Suppose two schedules π_1 and π_n satisfy the relation $\pi_1 \rightarrow^*_{free(\pi_1)} \pi_n$ and Δ is a strong similarity relation for both π_1 and π_n. Then, π_n is a derived schedule of π_1, i.e., either there is no write event from which a read event e_r reads in π_1 and π_n or the write events from which e_r reads in π_1 and π_n, respectively, are strongly similar in π_1. If e_w and e'_w are the last write events of certain data object in π_1 and π_n, respectively, then e_w and e'_w are strongly similar in π_1.*

Proof. Suppose e_r is a read event in π_1 and π_n. We shall show that either there is no write event from which e_r reads in π_1 and π_n or the write events from which e_r reads in π_1 and π_n, respectively, are strongly similar in π_1.

Suppose there is no write event from which e_r reads in π_1 and e_r reads from a write event e'_w in π_n. It implies that e'_w is after e_r in π_1. Since there is no write event from which e_r reads in π_1, there is no event in π_1 with which e_r can satisfy any condition in the definition of the $free(s_1)$ relation. According to Lemma 1, the order of e_r and e'_w must be the same in π_1 and π_n. This is contradictory to the assumption.

Suppose e_r reads from e_w in π_1, but there is no write event from which a read event e_r reads in π_n. Because e_w and e_r can never satisfy any condition in the definition of the $free(s_1)$ relation such that their order can be changed, according to Lemma 1, the order of e_w and e_r is the same in π_1 and π_n, again a contradiction.

Suppose e_r is a read event which reads from different write events e_w and e'_w in π_1 and π_n, respectively. If e'_w is after e_r in π_1, then the order of e'_w and e_r are different in π_1 and π_n because e'_w is before e_r in π_n. According to Lemma 1, e'_w and e_r must satisfy condition 3 in the definition of the $free(s_1)$ relation, i.e., e_w and e'_w are strongly similar in π_1.

Suppose e'_w is before e_r in π_1. It is impossible to have e'_w after e_w in π_1 because e_r reads from e_w in π_1. So, e'_w is before e_w and e_r in π_1. Because e_w and e_r can never satisfy any condition in the definition of the $free(s_1)$ relation such that their order can be changed, according to Lemma 1, the order of e_w and e_r is the same in π_1 and π_n. This implies that the order of e_w and e'_w is different in π_1 and π_n because e_r reads e'_w in π_n and e'_w is before e_w in π_1. According to Lemma 1, e_w and e'_w must satisfy condition 2 in the definition of the $free(s_1)$ relation, i.e., e_w and e'_w are strongly similar in π_1.

Therefore, either there is no write event from which e_r reads in π_1 and π_n or the write events from which e_r reads in π_1 and π_n, respectively, are strongly similar in π_1.

Suppose e_w and e'_w are the last write events of certain data object in π_1 and π_n, respectively. We shall show that e_w and e'_w are strongly similar in π_1. If e_w and e'_w are the same write event, the proof is complete. Otherwise, the orders of e_w and e'_w are different in π_1 and π_n because e_w is after e'_w in π_1 and e'_w is after e_w in π_n. According to Lemma 1, e_w and e'_w must satisfy condition 2 in the definition of the $free(s_1)$ relation, i.e., e_w and e'_w are strongly similar in π_1. □

Lemma 3. *Any conflict Δ-serializable schedule is view Δ-serializable.*

Proof. Follows from Theorem 2 and Lemma 2. □

Lemma 4. *There is a view Δ-serializable schedule that is not conflict Δ-serializable.*

Proof. From Example 3. □

Given a schedule π, we define its *transaction dependency graph* $TG(\pi)$ as follows: Corresponding to every transaction instance in π is a node in $TG(\pi)$. There is an arc from a node τ to another node τ' in $TG(\pi)$ if there are two events e and e' in τ and τ', respectively, which do not satisfy the $free(\pi)$ relation and e precedes e' in π. For simplicity, let us call any two events in π which do not satisfy the $free(\pi)$ relation being *ordered* in π.

Theorem 3. *The problem of determining whether a schedule is conflict Δ-serializable can be solved in $O(n^2)$ time, where n is the number of events in the given schedule.*

Proof. Given a schedule π, let $TG(\pi)$ be its transaction dependency graph. By Lemma 1, π is conflict Δ-serializable iff $TG(\pi)$ is acyclic. The cycle detection problem can be solved in $O(n^2)$ time. □

4 Extensions

4.1 Continual Operation

Since real-time applications usually run continually, database consistency may not be maintained at points when some transaction instances have not completed their executions. However, it is sufficient if every transaction instance sees a consistent view of the database. The purpose of this section is meant to extend the definitions of Δ-serializability to schedules with incomplete transaction instances (to be defined later). A similar idea can be found in [29].

Definition: Incomplete Transaction Instance. *An instance of a transaction is* incomplete *in a schedule if the schedule contains all events of the transaction instance which occur before some specified event in the instance.*

Note that a transaction instance is *complete* in a schedule if the schedule contains all events of the transaction instance.

Example 4: Incomplete Transaction Instances. Let $\tau_{1,1}$ and $\tau_{2,1}$ be instances of transactions τ_1 and τ_2, respectively, where τ_1 reads and writes data object X and τ_2 reads and writes data object Y. Consider the following schedule:

$$\pi = R(\tau_{1,1}, X), R(\tau_{2,1}, Y), W(\tau_{1,1}, X).$$

$\tau_{1,1}$ and $\tau_{2,1}$ are a complete transaction instance and an incomplete transaction instance in π, respectively. $\tau_{2,1}$ is incomplete in π because $\tau_{2,1}$ has not yet issued a write event on Y in π.

Intuitively, a schedule is acceptable at some observation point if it can be extended for the database to reach a consistent state, where extending a schedule by making transactions complete does indicate adding more operations. For convenience, we introduce the following definitions: A schedule π' is an *extended schedule* of another schedule π if π' is the schedule π appended with events which complete all incomplete transaction instances in π. A schedule π' is a *prefix schedule* of another schedule π if π' consists of all events in π which occur before or concurrently with some specified event in π. Because there may be more than one way to complete incomplete transactions, a schedule may have more than one extended schedule.

Example 5: Extended schedules and prefix schedules. Consider the following three schedules:

$$\pi_1 = R(\tau_{1,1}, X), R(\tau_{2,1}, Y)$$
$$\pi_2 = R(\tau_{1,1}, X), R(\tau_{2,1}, Y), W(\tau_{1,1}, X), W(\tau_{2,1}, Y)$$
$$\pi_3 = R(\tau_{1,1}, X), R(\tau_{2,1}, Y), W(\tau_{2,1}, Y), W(\tau_{1,1}, X),$$

π_1 is a prefix schedule of π_2 and π_3. π_2 and π_3 are both extended schedules of π_1 where τ_1 and τ_2 reads and updates X and Y, respectively.

A (finite) schedule is *potentially final-state/view/conflict Δ-serializable* if at least one of its extended schedules is final-state/view/conflict Δ-serializable. An infinite schedule is *final-state/view/conflict Δ-serializable* if every one of its prefix schedules is potentially final-state/view/conflict Δ-serializable.

Example 6: Extension of a prefix schedule to preserve consistency. Let $\pi = W(\tau_{1,1}, X), R(\tau_{2,1}, X)$ be a prefix schedule of another schedule $\pi' = W(\tau_{1,1}, X), R(\tau_{2,1}, X), W(\tau_{1,1}, Y), R(\tau_{2,1}, Y)$. In other words, π' is an extended schedule of π. Because π' is conflict similar to a serial schedule $\pi'' = \tau_{1,1}, \tau_{2,1}$, π is potentially conflict Δ-serializable. Note that another schedule

$$\pi''' = W(\tau_{1,1}, X), R(\tau_{2,1}, X), R(\tau_{2,1}, Y), W(\tau_{1,1}, Y),$$

which is not final-state/view/conflict Δ-serializable, is also an extended schedule of π.

This justifies our treatment of final-state/view/conflict Δ-serializability, even though there may not be a time instant at which a real-time database system has no incomplete transaction instances. Note that the complexity of determining whether a parital schedule or a complete schedule is final-state/view/conflict Δ-serializable are the same if missing events of incomplete transaction instances in a partial schedule are ignored. We shall consider only those transactions that have been completed unless otherwise stated.

4.2 Real Parallelism—Δ-Synchronizability

With the advent of parallel processing, true parallelism instead of interleaving [23] is likely to become more realistic in modeling the concurrent execution of real-time transactions. Correctness criteria based on the idea of interleaving (such as Δ-serializability described in previous sections) cannot capture the semantics of true parallelism. For example, to model the operation of the digital watch in Fig. 4, the notion of serializability cannot capture the semantics of the triggering of the time adjustment function which is invoked by pressing two buttons on the watch simultaneously. The purpose of this section is to extend the idea of Δ-serializability to justify the correctness of real-time transaction executions with true parallelism.

For real-time transaction execution, although simultaneity can be explicitly defined by proximity in time, the

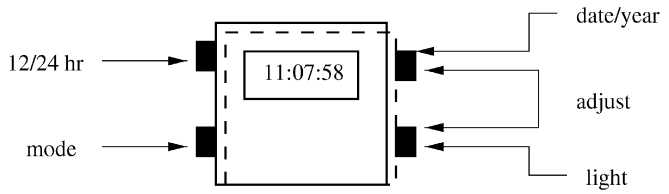

Fig. 4. A digital watch.

consistency of a schedule is usually checked by the relative ordering of events and not their time of occurrence. Simultaneous actions have to be grouped to form parallel transitions when we strip off time in consistency checks. Otherwise, there is no way to distinguish two simultaneous closely timed events from two unrelated events. A *parallel action* is defined as a partial order of events which denotes a parallel execution of a set of transaction instances. Semantically, transaction instances in a parallel action see the same database state and update the same database state. We require that no w/w-conflict events exist in a parallel action and every read event should take effect before all its conflicting write event in a parallel action.

A parallel action is consistent if it preserves all consistency predicates. In fact, a serial action is a special case of a parallel action with a single transaction instance. A *synchronous schedule* is a sequence of consistent parallel actions. A timed schedule is *synchronous* if its corresponding schedule is synchronous.

We say that a schedule is *final-state/view/conflict Δ-synchronizable* if it is final-state/view/conflict similar to some synchronous schedule.

Theorem 4. *The problem of deciding whether a schedule is conflict Δ-synchronizable can be solved in polynomial time if there exists a polynomial time algorithm for determining whether a parallel action is consistent.*

Proof. Given a schedule π, let $TG(\pi)$ be its transaction dependency graph, as defined in Section 3. By Lemma 1, all transaction instances in π involved in a cycle should be in a consistent parallel action and merged into a single node in $TG(\pi)$ if π is conflict Δ-synchronizable. Since there exists a polynomial time algorithm for determining whether a parallel action is consistent or not, the problem of deciding whether a schedule is conflict Δ-synchronizable can be solved by executing a sequence of cycle detections and deletions until an inconsistent parallel action is found or no more cycles exist. π is conflict Δ-synchronizable if no inconsistent parallel action can be found.

Since there are no more than n cycles in $TG(\pi)$, where n is the number of transaction instances in π, and there exists a polynomial time algorithm for determining whether a parallel action is consistent, the problem of deciding whether a schedule is conflict Δ-synchronizable can be solved in polynomial time. □

The problem of deciding whether a schedule is final-state/view Δ-synchronizable follows from the NP-hardness of the final-state/view serializability problem.

4.3 Logical Schedules vs. Physical Schedules

The concept of physical schedules is motivated by the observation that requests for the same database operation from several transactions that occur close in time can be and are often satisfied by the execution of a single database operation. To make a distinction between the actual physical execution of database operations and the logical events in individual transactions, we shall refer to the schedules that we have discussed so far as *logical schedules*. In this section, we define the correctness of the *physical schedules* that result from the actual execution of database operations. With the concept of physical schedules, it is possible to have more flexibility in implementing logical schedules and to justify the correctness of "actual execution." Furthermore, under the concept of physical schedules, a real-time database scheduler may skip unimportant computation or updates to meet time constraints and/or satisfy some safety requirements on the system.

An *occurrence* is a pair (op, obj) which represents an event op involving a data object obj. A *timed occurrence* is an occurrence with a time tag which indicates its occurrence time. A *timed physical schedule* is a collection of timed occurrences. A *physical schedule* is a partial order of occurrences.

Definitions: *A physical schedule π_p is a* realization *of a logical schedule π_l if there exists a* surjective (onto) mapping *which:*

1. *Maps each read event in π_l to a read occurrence in π_p,*
2. *Maps each write event in π_l either to a unique write occurrence in π_p or to the event* null,

such that:

1. *For any interpretation, π_l and π_p transform the same initial state into the same final state.*
2. *Each transaction instance has the same view in π_l and π_p.*

Note that each read event in a logical schedule must be mapped to a physical occurrence, but some write events in a logical schedule may be mapped to *null*, in which case they are not performed physically. Each write occurrence in a physical schedule is the image of exactly one write event in the logical schedule, but a read occurrence may be the image of several read events in a logical schedule.

Example 7: Logical schedules vs. physical schedules. The physical schedule: $(W, X), (R, X)$ is the realization of the logical schedule

$$W(\tau_{1,1}, X), W(\tau_{2,1}, X), R(\tau_{3,1}, X)$$

through the mapping

$$W(\tau_{1,1}, X) \rightarrow null$$
$$W(\tau_{2,1}, X) \rightarrow (W, X),$$
$$R(\tau_{3,1}, X) \rightarrow (R, X).$$

Lemma 5. *Every physical schedule π_p of a logical schedule π_l preserves the final-state/view/conflict Δ-serializability/ Δ-synchronizability property of π_l.*

Proof. Follows directly from the fact that the realization definition preserves transaction views in logical schedules and state transformations of logical schedules. □

Our notion of physical vs. logical schedules is different from the *Thomas Write Rule* (TWR) [40]. TWR ignores obsolete write events instead of rejecting them to prevent obsolete information from entering a database. Together with the basic r/w synchronization rules of a timestamp ordering protocol, serializability of schedules is enforced in TWR. Our concept of physical schedule is different. The unused write events in a schedule can be ignored and several read events from the same write event can be merged to maintain transaction views. Of course, the TWR can be used as to implement physical schedules; however, they were proposed for different applications.

5 A Sufficient Condition for Achieving Synchronization for Free

Similarity is an inherently application-dependent concept and we expect the application engineer to define it for specific applications. In many real-time applications, it is often acceptable to use an older value of a sensor as input to a calculation, instead of waiting for a more up-to-date value. This is possible because the physics of the application may be such that changes in sensor reading over a short interval of time are so small as to be insignificant to the calculation. This observation provides us with the needed connection between similarity and time constraints governing data access.

Specifically, 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, i.e., all instances of write events on the same object that occur in any interval shorter than the similarity bound can be swapped in the (untimed) schedule without violating consistency requirements. Notice that the existence of a similarity bound does not imply that the similarity relation is transitive since event swapping is based on (wall-clock) time values and not on the relative positions of events in a schedule.

5.1 Basic Idea

The basic idea is that transactions should not block one another as long as meeting time constraints guarantees the strong similarity of their conflicting events. The event conflicts are resolved by appealing to the similarity bound in the following discussion, which refers to Fig. 5.

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 Fig. 5, these two write events are strongly similar and it does not matter which write value 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 Fig. 5. 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 Fig. 5, then it does not matter which write value

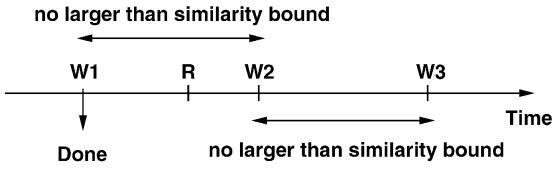

Fig. 5. Similarity of conflicting events.

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.

5.2 A Sufficient Condition: Time Constraints vs. Data Similarity

Suppose sb_i is a similarity bound for a data object x_i. Any two writes on x_i within an interval shorter than sb_i are interchangeable because they are strongly similar. Let p_i^{max}, p_i^{nxt}, and p_i^{min} be the maximum, the second largest, and the minimum periods of transactions updating x_i, respectively. If there is only one transaction updating x_i, then p_i^{max} is equal to p_i^{min} and p_i^{nxt}. Suppose p_i^r is the maximum period of transactions reading x_i. In the following, we shall derive a sufficient condition which guarantees the "strong similarity" of any concurrently executing transaction instances.

For simplicity of discussion, we assume in this paper that the deadline of a transaction instance is equal to the end of its period. Extension of our results to relax this restriction will be discussed later.

Write vs. Write Condition: $(p_i^{max} + p_i^{nxt}) \leq sb_i$. By our definition of strong similarity, two conflicting write events are interchangeable if they are strongly similar. In other words, conflicting write events of any overlapping transaction instances are interchangeable if these write events are strongly similar. (We say that two transaction instances overlap if their execution overlap in time.) If no transaction misses its deadline, the maximum temporal distance between any two conflicting write events of overlapping transaction instances on data object x_i is $(p_i^{max} + p_i^{nxt})$. Obviously, if $(p_i^{max} + p_i^{nxt}) \leq sb_i$, conflicting write events of any overlapping transaction instances are strongly similar and interchangeable (please see Fig. 6).

Notice that if the deadline of a transaction instance is before the end of its period, then the Write vs. Write condition can be modified as follows: Let d_i^{nxt} be the maximum (relative) deadline of transactions updating x_i (excluding transaction τ_i^{max}). The Write vs. Write condition becomes $(p_i^{max} + d_i^{nxt}) \leq sb_i$. Furthermore, the Write vs.

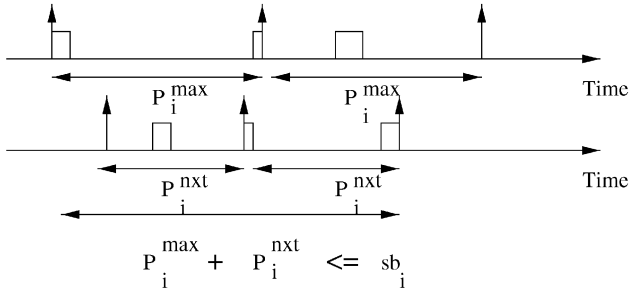

Fig. 6. Write vs. write condition.

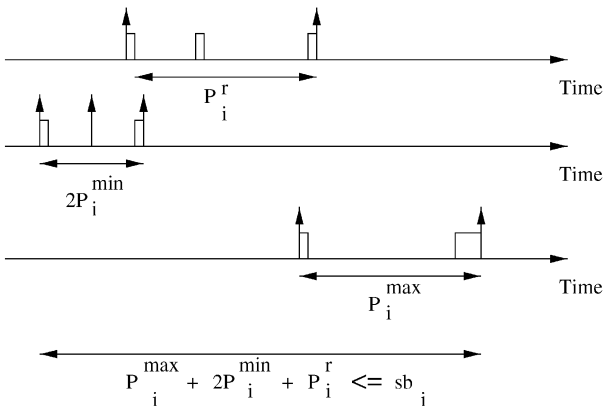

Fig. 7. Read vs. write condition.

Write condition for data object x_i can be ignored if there is only one transaction updating x_i. This is because no two instances of the same transaction will overlap if the transaction never misses its deadline.

Read vs. Write Condition: $(p_i^{max} + 2p_i^{min} + p_i^r) \leq sb_i$. Suppose τ is a transaction with period p_i^r and reads data object x_i. To ensure correctness, conflicting write events which might be read by an instance of τ must be strongly similar (thus interchangeable) so that any instance of τ will not block or be blocked by transaction instances which may update x_i.

If no transaction updating x_i misses its deadline, then no read event e_r can read from a conflicting write event which occurs more than $2p_i^{min}$ ago. Let this oldest write event be called $write^{old}$ of e_r. For ease of argument, we assume without loss of generality that the initial database state is determined by a fictitious set of write events so that an oldest write event always exists. On the other hand, a transaction instance which overlaps with the transaction instance issuing e_r may issue a conflicting write event almost p_i^{max} later than the end of the period of the transaction instance issuing e_r. Let this write event be $write^{young}$ of e_r. Obviously, this transaction instance of τ (which issues e_r) should not block or be blocked by any transaction instance because of read-write access conflict on x_i, assuming that the maximum temporal distance of $write^{old}$ and $write^{young}$ of e_r is no more than the similarity bound sb_i of x_i. In order words, read-write access conflict of x_i can be resolved if $(p_i^{max} + 2p_i^{min} + p_i^r) \leq sb_i$ (please see Fig. 7).

Notice that if the deadline of a transaction instance is before the end of its period, then the Read vs. Write condition can be modified as follows: Let d_i^{max} be the maximum (relative) deadline of transactions updating x_i and d_i^r the maximum (relative) deadline of transactions reading from x_i. The Read vs. Write condition becomes $(d_i^{max} + 2p_i^{min} + d_i^r) \leq sb_i$. When there is only one transaction updating x_i, then $p_i^{max} = p_i^{min}$. (In the last case, further optimization is possible.)

We claim that if a transaction set satisfies the *Read vs. Write* and *Write vs. Write* conditions, then these transactions can be scheduled independently as if they do not share data (with the usual assumption that individual read and write events are atomic). Formal justification of this claim is stated in Theorem 5 below.

Theorem 5. *If a transaction set satisfies both the Read vs. Write and Write vs. Write conditions, then any schedule that satisfies all transaction deadlines is view Δ-serializable.*

Proof. The proof follows directly from Theorem 2 if there exists a serial schedule π' which is a derived schedule of any schedule π that satisfies all transaction deadlines. According to the definition of "derived schedule," π and π' must satisfy the following two requirements: 1) All write events read by the same read event in π and π' must be strongly similar in π and 2) the last write events on every data object in π and π' must be strongly similar in π. In the following, we shall prove that there exists a sequence of event swaps from π to some serial schedule π' such that the requirements of a derived schedule are preserved at every step in the sequence.

Since any conflicting write events of overlapping transaction instances in π are strongly similar (according to the *Write vs. Write* condition), they can be swapped in any way without violating the second requirement of a derived schedule. Likewise, a conflicting read event and a conflicting write event of two overlapped executing transaction instances in π can be swapped in any way without violating the first requirement of a derived schedule because they are "strongly similar" according to the *Read vs. Write* condition. In particular, instances of all write events on the same data object that occur in any interval shorter than the similarity bound can be swapped in a (untimed) schedule without violating consistency requirements. Thus, swapping such write events will not violate the first requirement of a derived schedule. Therefore, conflicting events of overlapping transaction instances can be swapped in any order. Since nonconflicting events can also be swapped in any order, events of overlapping transaction instances can be swapped in any order. In other words, overlapping transaction instances can be serialized in any order. Also, transaction instances which are not overlapped in π are already serialized. Therefore, π can be serialized by swapping events of overlapping transaction instances in any order. □

5.3 Extensions

Since different transactions may have different precision requirements for a data object, the *Read vs. Write* and *Write vs. Write* conditions can be weakened. Suppose sb_i' is the similarity bound of a data object x_i with respect to a transaction τ'. The *Read vs. Write* condition can be weakened to: $(p_i^{max} + 2p_i^{min} + p') \leq sb_i'$ if the period of τ is p'. The *Write vs. Write* condition can be weakened to: $(p_i^{max} + p_i^{nxt}) \leq sb_i'$.

Finally, we consider the situation where some transactions satisfy the *Read vs. Write* and *Write vs. Write* conditions, but others do not. In this case, the transaction system cannot be scheduled "fully" independently. A simple variation of Similarity Stack Protocol (SSP) [17] can be made to take care of this situation, as follows:

As in SSP, transactions are partitioned into interactive sets such that no two transactions in different interactive sets may share any data object. If all transactions in an

interactive set satisfy the *Read vs. Write* and *Write vs. Write* conditions, the recency bound of the interactive set can be set to ∞ such that transactions in the interactive set can be scheduled independently of one another. Here, the recency bound of an interactive set limits the length of any interval spanned by overlapping transaction instances in the set. If any transaction in an interactive set fails any one of the conditions, the recency bound of the interactive set is calculated as defined in [17]. The correctness of this approach can be justified by an argument similar to the last section.

6 Conclusion and Future Research

In this paper, we discuss the semantics of real-time database applications and propose new correctness criteria for concurrency control which exploits the interchangeability of input data values of sufficient precision. The concept of similarity is used to extend the usual correctness criteria for transaction scheduling: final-state, view, conflict serializability to their counterparts of final-state Δ-serializability, view Δ-serializability, and conflict Δ-serializability. We then generalize the criteria for infinite schedules and extend the criteria when true parallelism, instead of interleaving, is considered. We propose the idea of physical schedules in which a real-time database scheduler may skip unimportant computation or updates to meet time constraints and/or satisfy some safety requirements on the system. The correctness of physical schedules is justified by the notion of similarity. We also take a semantic approach and propose a sufficient condition for achieving data synchronization for free, which is based on the concept of similarity. Real-time transactions satisfying this condition can be scheduled correctly by any process scheduling discipline that is designed for the independent processes model [24] (e.g., RMS, EDF), where no locking of data is assumed. With our approach, the usually high utilization factor that can be achieved by these scheduling disciplines is also attainable for transactions satisfying our condition. The idea of similarity-based read and write operations was later implemented under a distributed Real-Time Object Management Interface (RTOMI) on an Intel Multiprocessor computer [19] and found effective in reducing a significant portion of remote data access in a typical industrial control application [5].

The integrity management of real-time database systems involves two issues: 1) External consistency and temporal consistency constraints must be specified and justified to be adequate for the similarity relations specific to the application. 2) Real-time transaction scheduling algorithms are needed to enforce these constraints. These issues give rise to many interesting real-time scheduling problems.

For future research, we shall investigate quantitatively how the concept of similarity may be used to improve the schedulability of real-time transaction workloads, e.g., how it can be used to exploit parallel processing hardware and to improve the achievable utilization of multiprocessor systems. We believe that there are many interesting research issues concerning the concept of similarity. To gain experience, it is important to investigate how to construct similarity relations systematically from application specifications, such as a stock market system [23]. A toolset which facilitates reasoning about similarity relations for typical real-time applications should be very useful.

Acknowledgments

This research was supported in part by research grants from the Republic of China National Science Council under Grant NSC85-2213-E-194-008 and from the US Office of Naval Research under ONR contract number N00014-89-J-1472. This paper is an extended version of papers that appeared in the *Proceedings of the IEEE 13th Real-Time Systems Symposium* and the *Proceedings of the IEEE 11th Workshop on Real-Time Operating Systems and Software*.

References

[1] R. Abbott and H. Garcia-Molina, "Scheduling Real-Time Transactions: A Performance Evaluation," *Proc. 14th VLDB Conf.*, pp. 1-12, 1988.

[2] A. Bestavros, "Timeliness via Speculation for Real-Time Databases," *Proc. IEEE 15th Real-Time Systems Symp.*, 1994.

[3] A. Bestavros, "Advances in Real-Time Database Systems Research," *Special Section on RTDB of ACM SIGMOD Record*, vol. 25, no. 1, Mar. 1996.

[4] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Reading, Mass.: Addison-Wesley, 1987.

[5] D. Chen and A.K. Mok, "SRDE—Application of Data Similarity to Process Control," *Proc. IEEE 20th Real-Time Systems Symp.*, pp. 136-145, Dec. 1999.

[6] L.B.C. Dipippo and V.F. Wolfe, "Object-Based Semantic Real-Time Concurrency Control," *Proc. IEEE Real-Time Systems Symp.*, Dec. 1993.

[7] W. Du and A.K. Elmagarmid, "Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase," *Proc. 15th Int'l Conf. Very Large Data Base*, pp. 347-355, 1989.

[8] S.B. Davidson and A. Watters, "Partial Computation in Real-Time Database Systems," *Proc. Fifth Workshop Real-Time Software and Operating Systems*, pp. 117-121, May 1988.

[9] M.H. Graham, "How to Get Serializability for Real-Time Transactions without Having to Pay for It," *Proc. IEEE 14th Real-Time Systems Symp.*, 1993.

[10] H. Garcia-Molina and K. Salem, "SAGAS," *Proc. 1987 ACM SIGMOD Conf. Management of Data*, pp. 249-259, 1987.

[11] H. Garcia-Molina and G. Wiederhold, "Read-Only Transactions in a Distributed Database," *ACM Trans. Database Systems*, vol. 7, no. 2, pp. 209-234, June 1982.

[12] M. Gouda, *Beauty Is Our Business: A Birthday Salute to Edsger W. Dijkstra*, W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, eds., pp. 135-140, Springer-Verlag, 1990.

[13] J.R. Haritsa, M.J. Carey, and M. Livny, "On Being Optimistic about Real-Time Constraints," *Proc. Ninth ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems*, pp. 331-343, Apr. 1990.

[14] M.U. Kamath and K. Ramamritham, "Performance Characteristics of Epsilon Serializability with Hierarchical Inconsistency Bounds" *Proc. Int'l Conf. Data Eng.*, pp. 587-594, Apr. 1993.

[15] T.-W. Kuo, "Real-Time Database—Semantics and Resource Scheduling," PhD dissertation, Univ. of Texas at Austin, 1994.

[16] T.-W. Kuo and A.K. Mok, "Application Semantics and Concurrency Control of Real-Time Data-Intensive Applications," *Proc. IEEE 13th Real-Time Systems Symp.*, 1992.

[17] T.-W. Kuo and A.K. Mok, "SSP: A Semantics-Based Protocol for Real-Time Data Access," *Proc. IEEE 14th Real-Time Systems Symp.*, Dec. 1993.

[18] T.-W. Kuo and A.K. Mok, "Using Data Similarity to Achieve Synchronization for Free," *Proc. IEEE 11th Workshop Real-Time Operating Systems and Software*, pp. 112-116, 1994.

[19] T.-W. Kuo and A.K. Mok, "The Design and Implementation of Real-Time Object Management Interface," *Proc. IEEE 1995 Real-Time Technology and Applications Symp.*, 1995.

[20] Y.-K. Kim and S.H. Son, "Supporting Predictability in Real-Time Database Systems," *Proc. IEEE 1996 Real-Time Technology and Applications Symp.,* 1996.
[21] H.F. Korth and G.D. Speegle, "Formal Model of Correctness without Serializability," *Proc. 1988 ACM SIGMOD Conf. Management of Data,* pp. 379-386, 1988.
[22] H.F. Korth, N. Soparkar, and A. Silberschatz, "Triggered Real Time Databases with Consistency Constraints," *Proc. 16th VLDB Conf.,* Aug. 1990.
[23] K.-Y. Lam, T.-W. Kuo, and L. Shu, "On Using Similarity to Process Transactions in Stock Trading Systems," *Proc. IEEE 1998 Workshop Dependable and Real-Time E-Commerce Systems,* June 1998.
[24] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM,* vol. 20, no. 1, pp. 46-61, Jan. 1973.
[25] K.-J. Lin, S. Natarajan, and J.W.-S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," *Proc. IEEE Eighth Real-Time Systems Symp.,* pp. 210-217, Dec. 1987.
[26] K.-J. Lin and M.-J. Lin, "Enchancing Availability in Distributed Real-Time Databases," *ACM SIGMOD Record,* vol. 17, no. 1, pp. 34-43, Mar. 1988.
[27] Y. Lin and S.H. Son, "Concurrency Control in Real-Time Databases by Dynamic Adjustment of Serialization Order," *Proc. IEEE 11th Real-Time Systems Symp.,* Dec. 1990.
[28] A.K. Mok, "Fundamental Design Problems for the Hard Real-Time Environment," PhD dissertation, Massachusetts Inst. of Technology, Cambridge, Mass., 1983.
[29] C. Papadimitriou, *The Theory of Database Concurrency Control.* Computer Science Press, 1986.
[30] C.-S. Peng and K.-J. Lin, "A Semantic-Based Concurrency Control Protocol for Real-Time Transactions," *Proc. IEEE 1996 Real-Time Technology and Applications Symp.,* 1996.
[31] C. Pu and A. Leff, "Epsilon-Serializability," Technical Report CUCS-054-90, Dept. of Computer Science, Columbia Univ., Jan. 1991.
[32] C.-S. Peng, K.-J. Lin, and T. Ng, "A Performance Study of the Semantics-Based Concurrency Control Protocol for Air Traffic Control System," *Proc. Int'l Workshop Real-Time Databases,* Sept. 1997.
[33] K. Ramamritham and C. Pu, "A Formal Characterization of Epsilon Serializability," *IEEE Trans. Knowledge and Data Eng.,* vol. 7, no. 6, pp. 997-1,007, Dec. 1995.
[34] L. Sha, R. Rajkumar, and J.P. Lehoczky, "Concurrency Control for Distributed Real-Time Databases," *ACM SIGMOD Record,* vol. 17, no. 1, pp. 82-98, Mar. 1988.
[35] *ACM SIGMOD Record: Special Issue on Real-Time Databases,* S.H. Son, ed., Mar. 1988.
[36] X. Song and J.W.-S. Liu, "Maintaining Temporal Consistency: Pessimistic vs. Optimistic Concurrency Control," *IEEE Trans. Knowledge and Data Eng.,* pp. 787-796, Oct. 1995.
[37] L. Shu and M. Young, "A Mixed Locking/Abort Protocol for Hard Real-Time Systems," *Proc. IEEE 11th Workshop Real-Time Operating Systems and Software,* pp. 102-106, May 1994.
[38] L. Shu, M. Young, and R. Rajkumar, "An Abort Ceiling Protocol for Controling Priority Inversion," *Proc. First Int'l Workshop Real-Time Computing Systems and Applications,* pp. 202-206, Dec. 1994.
[39] J.A. Stankovic and W. Zhao, "On Real-Time Transactions," *ACM SIGMOD Record,* vol. 17, no. 1, pp. 4-18, June 1989.
[40] R.H. Thomas, "A Solution to the Concurrency Control Problem for Multiple Copy Data Bases," *Proc. 1978 COMPCON Conf.,* pp. 56-62, 1978.
[41] M. Xiong, K. Ramamritham, R. Sivasankaran, J.A. Stankovic, and D. Towsley, "Scheduling Transactions with Temporal Constraints: Exploiting Data Semantics," *Proc. IEEE Real-Time Systems Symp.,* pp. 240-251, Dec. 1996.

Tei-Wei Kuo received the BSE degree in computer science and information engineering from National Taiwan University in Taipei, Taiwan, in 1986. He received the MS and PhD degrees in computer sciences from the University of Texas at Austin in 1990 and 1994, respectively. He is currently an associate professor in the Department of Computer Science and Information Engineering of the National Taiwan University, Taiwan, Republic of China (ROC). He was an associate professor in the Department of Computer Science and Information Engineering of the National Chung Cheng University, Taiwan, ROC, from August 1994 to July 2000. His research interests include real-time databases, real-time process scheduling, real-time operating systems, and control systems. He is the program chair of IEEE Seventh Real-Time Technology and Applications Symposium, 2000 and has consulted for government and industry on problems in various real-time systems design. Dr. Kuo is a member of the IEEE and the IEEE Computer Society.

Aloysius K. Mok received the BS and MS degrees in electrical engineering and computer science, and the PhD degree in computer science in 1983, all from the Massachusetts Institute of Technology. He is currently a professor of computer sciences at the University of Texas at Austin. His current interests include design problems of robust, distributed real-time systems, real-time database, and performance issues of real-time rule-based programs. He has consulted for government and industry on problems in real-time system design and was the chair of the IEEE Technical Committee on Real-Time Systems (1995-1996). He is a member of the IEEE.