ABSOLUTE AND RELATIVE TEMPORAL CONSTRAINTS IN HARD REAL-TIME DATABASES

N. C. Audsley, A. Burns, M. F. Richardson and A. J. Wellings

Department of Computer Science, University of York,
York Y01 5DD, England †

Abstract

Application of a conventional database to a hard real-time system requires two inter-related NP-complete problems to be solved: database concurrency control and task scheduling. Starting from temporal constraints on data objects in the database, the techniques described in this paper can be used to reduce the problem to a single NP-complete scheduling problem. Standard scheduling theory can then be applied.

1: Introduction

Traditionally, databases are used in systems which either do not operate in real-time, or where quick response is desirable but not strictly necessary. Incorporating databases into hard real-time systems necessarily imposes stricter timing constraints, as failure to meet a constraint can have catastrophic effects. Formally, the correct functioning of the system depends on timely operation as well as on functionally correct operation.

A number of algorithms have been proposed and used in order to improve the performance of conventional databases, for example *multiversion serialisability* [Pap84, Agr89], *semantic atomicity* [Cel88], *multilevel atomicity* [Lyn83] *weak correctness* [Sha88a, Sha88b] and *abort/restart* schemes [Kun81]. Generally, these rely on one of two techniques; either database consistency constraints are weakened, or advantage is make of known patterns of database access. However, the methods are characterised by the fact that the improvement is statistical in nature, and cannot be guaranteed. Hence, while they may be beneficial in (soft) real-time systems, they are not sufficient for use in hard real-time systems.

However, the type of systems in which conventional databases are used generally differ from hard real-time systems. We identify these differences, and note the consequences for hard real-time systems in section 2. A database model appropriate to a large class of hard real-time systems is then described in section 3, and the issues related to maintaining consistency in this model are discussed in section 4. Section 5 provides a simple example. Section 6 outlines the way in which transaction timing characteristics can then be derived.

For the purposes of this paper, we define the term *database* as below. Note that we do not include associated systems (such as database management systems) within the definition.

> **Defn (1):** *A database is a structured collection of data objects, where specified consistency constraints between data objects must be maintained.*

2: Database Characteristics

We note the following differences between typical conventional and hard real-time databases:

- Conventional databases have a very large, and in principal unbounded, number of data objects; further, the

† This work is supported in part by the UK Information Engineering Advanced Technology Programme Grant, GR/F 35920/4/1/1214.

number of objects can vary during the lifetime of the system. In a hard real-time database, however, the number must be statically bounded, and is generally much smaller.

- Many transactions in a conventional database may access an arbitrary subset of data objects. In a hard real-time database, transactions will access a fixed subset of data objects.

- Transaction invocation in a hard real-time database is much more predictable. Many (*periodic*) transactions will execute at regular intervals; those that do not will generally have minimum inter-arrival times (*sporadic* transactions).

- Conventional databases hold data which is essentially discrete and which is changed in a stepwise manner; there is often little scope for using anything other than the most recent data value. Hard real-time systems, however, are often embedded as the control element of continuous systems, and may well be able to use data values which are *sufficiently* recent.

- The *consistency* of a conventional database, and the *correctness* of its transactions, are defined purely over the values stored in data objects (*functional* consistency). While these may also apply to a hard real-time database, there is an additional requirement concerning time (*temporal* consistency).

- Transaction scheduling in a conventional database allows any ordering which preserves functional consistency. In a hard real-time database, the scheduling must also preserve temporal consistency. These two problems are individually NP-complete [Kun79, Mok83].

As implied in the last item above, the introduction of hard real-time constraints introduces a second NP-complete dimension to the scheduling problem. On the other hand, since (a) the number of data objects is bounded; (b) transactions operate on fixed subsets of data objects; and (c) the frequency of invocation of transactions is known (or at least bounded), it is in principal possible to enumerate the computational requirements placed on the database, and hence to test for scheduling feasibility. Further, the ability to use sufficiently recent values of data objects helps to eliminate interference between transactions; from a scheduling point of view, this allows a larger number of possible schedules.

3: A System Model

To analyse the requirements of a hard real-time database, we propose a general model which provides applications with a suitable and useful set of abstractions.

3.1: Database Object Structure

The database is modelled as an acyclic directed graph, where nodes represent data objects, and arcs represent the notion *depends upon*. The data objects may be partitioned into *input* objects, *output* objects, and *derived* objects. This partition is described by Song [Son90]. Input objects model inputs from the environment (*sensors*), and do not depend on any other data objects; output objects model outputs to the controlled system (*actuators*); and derived objects hold values which are internally computed.

Clearly, input objects represent measurements of the environment. The derived objects may also be considered to represent quantities in the environment, but which are inferred rather than measured directly. We use the term *real-world value* to refer to the actual value of the quantity in the environment, and say that the value of a data object *corresponds* to its real-world value if it differs by no more than a bounded error.

3.2: Database Transaction Structure

All software components interact via the database; they read and write subsets of the data objects. In this paper, we use the term *transaction* to identify such components, by analogy with conventional databases. The transactions implement the *depends upon* relationships amongst the data objects.

In a conventional database, an event in the environment may require the exection of a transaction which updates some data objects. This involves the use of some protocol in order to prevent conflict with other concurrently executing transactions. However, the completion of the transaction does not ordinarily imply any further activity, since all changes that follow directly from the event will have been made by the transaction.

In the model presented here, however, a transaction which is triggered by an event in the environment need not update all data objects which are dependent on the event, leaving some changes to be made by other transactions. In effect, the event triggers the first of a series of transactions. This reduces the potential for conflict between transactions.

However, as a result of the model, the hard real-time database may, by comparison with a conventional database, be functionally inconsistent for intervals of time. This is tolerable so long as data values remain *sufficiently* up to date. We note that, using this model, if all inputs were to freeze in value, then functional consistency would *eventually* [She90] appear. This will occur within bounded time if all *depends upon* relationships are restored in bounded time.

3.2: (Non-)Perishable Objects

Ideally, a data object should exactly track its real-world value, but this may not be possible for two reasons. Firstly, the data object is limited to taking one of a finite set of values, which may not be sufficient to represent all possible real-world values; this leads to a *value* error, which must however be bounded. Secondly, some real-world values may change too rapidly for the data object to track, leading to *temporal* errors. We therefore define the notion, which is used later, of a *perishable* data object.

> **Defn (2):** *A data object is perishable if it cannot track the corresponding real-world value within the value error.*

We refer to any data item which is not perishable as *non-perishable.* Clearly, the division is dependent on the speed with which transactions can be executed, as well as on the rate at which real-world values change.

4: Maintaining Consistency

We now consider the issue of specifying consistency in our database model. We start by defining *absolute* and *relative* temporal consistency, and define a version of *functional* consistency. The terms are derived from those used by Song [Son90], however we use *consistency* rather than *coherence* by analogy with conventional databases.

4.1: Absolute Temporal Consistency

In order that a transaction can execute meaningfully, it requires that the values of data objects which it reads are both accurate within some bounded value error, and that they are sufficiently recent. We quantify *sufficient* by the *absolute* temporal consistency (abbreviated *ATC*) of the data object, as follows:

> **Defn (3):** *If data object X has an ATC a_x then, at time t_{now}, the value v_x stored in X must correspond to the real-world value of X at some time within the interval $[t_{now}-a_x,t_{now}]$.*

Note that, if a data object is updated by a single transaction, then the responsibility for maintaining the ATC falls solely on that transaction.

3.2: Relative Temporal Consistency

If a transaction reads a set of data objects, then it may be necessary that the values correspond to real-world values at similar times. This type of consistency is referred to as *relative* temporal consistency (abbreviated *RTC*).

> **Defn (4):** *If a set of data objects* **S**$=\{V_i\}$ *has an RTC r_s, then the values v_i must correspond to real-world values of the V_i at times which fall within an interval of length r.*

An RTC requirement on its own is not useful; however, it becomes particularly useful if the RTC requirement is significantly shorter than the ATC requirements of the individual data objects.

4.3: Functional Consistency

A *depends upon* relationship from data object X to data object Y can be expressed either as Y=**f**(X), or as **P**(Y,X) where **P** is a predicate which must hold for the database to be consistent. The latter form is that which is used to define consistency in conventional databases.

However, in our model, the transaction which implements **f** may not be executed for some time after data object X changes, and hence **P**, as applied to the current values of X and Y, will be false. We therefore define functional consistency in the following manner.

> **Defn (5):** *If predicate* **P** *applies to data objects X and Y, having ATC requirements a_x and a_y, then X and Y are functionally consistent at time t_{now} if* **P**(v_y,v_x) *is true, where v_y is the value of Y at time t_{now}, and v_x is some value held by X in the time interval $[t_{now}-(a_y-a_x),t_{now}]$.*

This is justified as follows. We assume that in order for v_y to satisfy the ATC of Y, it must be equal to **f**(v_0) where v_0 was a value of X which itself corresponded to the real-world value of X in the time interval $[t_{now}-a_y,t_{now}]$. In the worst case, if definition (5) holds, then v_0 could have been read from X at time $t_f=t_{now}-(a_y-a_x)$ and, at time t_f, v_0 could have corresponded to the real-world value of X at a time a_x earlier. Hence, the value v_y could in the worst case be based on a value of X corresponding to time t_f-a_x. This is equal to $t_{now}-a_y$, which is the ATC requirement of Y.

This definition can be trivially generalised to the case where Y is dependent on a set of data objects $\mathbf{S}=\{X_i\}$. In this case, there may also be an RTC requirement on **S**.

5: A Simple Example

In this section we present a trivial example, illustrated in figure 1. Circles represent database objects, boxes represent transactions, and arrows represent data flow.

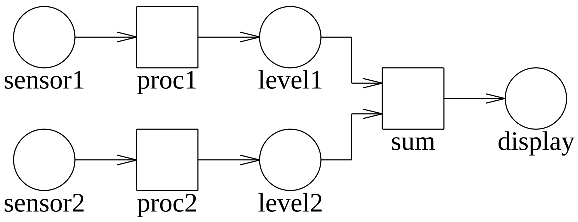

Figure 1: A Simple Example

An aircraft uses two fuel tanks *tank1* and *tank2*, whose levels are measured by sensors *sensor1* and *sensor2*. The sensor values are processed by two transactions (*proc1* and *proc2*) to check for sensor failure, and to generate values *value1* and *level2* which have short-term fluctuations (caused by fuel slopping about) removed. A third transaction *sum* adds the processed levels and outputs the sum to a fuel level display *display*.

It may be adequate that the display can be up to 10 seconds out of date, hence $a_{display}=10$. If we assume that the sensors inputs show the instantaneously correct value, then $a_{sensor1}=a_{sensor2}=0$. The values for a_{level1} and a_{level2} are essentially arbitrary, and for convenience we may set $a_{level1}=a_{level2}=5$.

Now, suppose the fuel may be pumped between tanks for trimming purposes. If a significant amount can be moved within a 5 second period, the ATC requirements alone on *level1* and *level2* are insufficient. If *sum* reads the levels at time t_{now}, then *level1* could contain the level at time t_{now}-5, and *level2* at time t_{now}. If fuel was pumped from *tank1* to *tank2* in the inteval [t_{now}-5,t_{now}], then the sum will be too high; conversely, if fuel is pumped from *tank2* to *tank1*, then the sum will be too low.

An RTC requirement must therefore be imposed on data objects *level1* and *level2*. For example, if fuel can be pumped at a rate of 1gallon/sec, and the error due to this must be limited to 1gallon, then we require $r_{level1,level2}=1$.

6: Transaction Scheduling

The definition of functional consistency given above can be used to derive timing requirements for the transactions. In this section we outline this derivation for three cases: ATC requirements for perishable and non-perishable data objects, and for RTC requirements. The full derivations of these results are given by Audsley *at al* [Aud].

As noted earlier, if a data object is written by a single transaction, then the responsibility for its functional consistency falls solely on that transaction, and execution of the transaction is not directly dependent on any other transaction. We therefore adopt this *single-writer* transaction.

This restriction also allows the use of algorithms described by Simpson [Sim90a, Sim90b] for access to data objects. Briefly, these work on both uniprocessor and shared-memory multiprocessor systems, and allow shared access to data objects without the need for synchronisation mechanisms such as *locking* [Esw76], hence eliminating blocking between transactions. Although hard real-time scheduling algorithms which handle blocking are known [Sha90], the general effect is to severly restrict schedulability. Also, Simpson's algorithms have the useful property that a data object cannot be left in an internally inconsistent state if the transaction which writes it should fail.

6.1: Perishable Objects

If a data object is perishable, a value written to it may be out of date even as it is written, and the transaction which writes to it must be executed again within a bounded time which is related to the ATC of the data object. The simplest solution is to schedule the transaction with a fixed period and deadline. If $Y=\mathbf{f}(X)$, where the ATC requirements are a_x and a_y, and the transaction has period T_p and deadline D_p, then we can derive the result:

$$T_p + D_p < a_y - a_x$$

This can be generalised to the case where the transaction reads and writes multiple data objects. If it reads data objects $\mathbf{X}=\{X_1,...,X_n\}$, and writes data objects $\mathbf{Y}=\{Y_i,...,Y_m\}$, then:

$$T_p + D_p < \min_{i,j}(a_{y_j} - a_{x_i})$$

6.2: Non-Perishable Objects

If data object Y is non-perishable, then, once it has been written, it may be a considerable time before it needs to be written again. In this case, the transaction which implements Y=**f**(X) can be sporadic. If we assume that the transaction is released as soon as X changes value, then if the deadline of the transaction is D_s:

$$D_s < a_y - a_x$$

Similarly, for multiple data objects:

$$D_s < \min_{i,j}(a_{y_j} - a_{x_i})$$

In order to guarantee schedulability, consecutive releases of the transaction must be separated by a known minimum inter-arrival time. This must not be less than $\min_{i,j}(a_{y_j}-a_{x_i})$ in order for Y to be non-perishable.

6.3: RTC Requirements

Consider a set of data objects **S**=$\{X_i,...,X_n\}$ with RTC requirement r_s, and individual ATC requirements a_i. We assume that the transactions τ_i which write the X_i are periodic (with periods and deadlines T_i and D_i), and that they have a critical deadline. Also, we assume that when X_i is written by τ_i at time t_i, the value corresponds to the real-world value at a time not earlier than t_i-δ_i, where δ_i is a function only of τ_i.

We can show that:

$$\max_i(D_i - \delta_i) < r_s$$

If LCM is the least common multiple function, and the $a_{i,j}$ are the ATC requirements of the data objects read by transaction τ_i, then:

$$\underset{i}{LCM}(T_i) + \max_i(D_i) < \min_i(a_i - \max_j(a_{i,j}))$$

In addition, we require that each data object X_i can hold the $LCM_j(T_j)/T_i$+1 most recent data values. If this is not the case, and only a single value can be held, then all transactions τ_i must have the same period.

6.4: Scheduling Theory

The above results require periodic transactions which may have deadlines less than their period, and which lack critical instants. Standard scheduling theory, such as the rate-monotonic and earliest-deadline algorithms [Liu73], is not adequate in this respect. However, deadline-monotonic scheduling [Aud91] can provide the necessary functionality.

6: Conclusions

In this paper, we present a database model which is applicable to at least some hard real-time systems. Its essential property is that its functional consistency is defined in terms of both value and temporal consistency.

If a conventional database is used, then two interrelated NP-complete problems must be solved; transaction scheduling for conventional functional consistency, and transaction scheduling to meet hard deadlines. The model presented here transforms the problem so that only a single NP-complete problem, namely that of hard real-time scheduling, needs be solved.

Given the model, is it worthwhile to question whether it is indeed a database. To this end we may consider the notions of atomicity, consistency, isolation and durability. The single-writer restriction supplies atomicity; consistency is maintained via the definition of functional consistency; and isolation is provided via the ability to use sufficiently recent values. Although durability has not been addressed in detail, is it achievable; indeed, by eliminating locking the problem is made easier.

It is in the issue of consistency that the model is unconventional. One of the motivations for this formulation is as a system modelling technique. Within the appropriate application domain it is possible to capture all timing constraints within the database. From these, transaction timing requirements can be systematically derived, to give a transaction set that is ameanable to schedulability analysis. Hence, the absolute and relative temporal consistency requirements can be guaranteed.

References

Agr89. D. Agrawal and S. Sengupta, ‘‘Modular Synchronisation in Multiversion Databases: Version Control and Concurrency Control’’, *1989 ACM-SIGMOD Intl. Conf.* (1989).

Aud. N. Audsley, A. Burns, M. Richardson and A. Wellings, ‘‘Data Consistency in Hard Real-Time Systems’’, *submitted for publication*.

Aud91. N. C. Audsley, ‘‘Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times’’, YCS 164, Dept. of Computer Science, University of York (December 1991).

Cel88. W. Cellary, E. Gelenbe and T. Morzy, *Concurrency Control in Distributed Database Systems*, 1988.

Esw76. K. P. Eswaran, J. N. Gray and R. A. Lorie, ‘‘The Notions of Consistency and Predicate Locks in a

Database System'', *Communications of the ACM* **19**(11), pp. 624- (November 1976).

Kun79. H. T. Kung and C. H. Papadimitriou, ''An Optimal Theory of Concurrency Control for Databases'', *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 111-126 (1979).

Kun81. H. T. Kung and J. T. Robinson, ''On Optimistic Methods of Concurrency Control'', *ACM Trans. on Database Systens*, pp. 213-226 (June 1981).

Liu73. C. L. Liu and J. W. Layland, ''Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment'', *Journal of the ACM 20(1)*, pp. 174-189 (1973).

Lyn83. N. A. Lynch, ''Multi-Level Atomicity - A New Correctness Criterion for Database Concurrency Control'', *ACM Transactions on Data Base Systems Vol 8 No 4*, pp. 484-502 (Dec 1983).

Mok83. A. K. Mok, ''Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment'', *PhD. Theses, MIT/LCS/TR-297* (1983).

Pap84. C. H. Papadimitriou and P. C. Kanellakis, ''On Concurrency Control by Multiple Versions'', *ACM Trans. on Database Systems* (March 1984).

Sha88a. L. Sha, J. P. Lehoczky and E. D. Jensen, ''Modular Concurrency Control and Failure Recovery'', *IEEE Transactions on Computers*, pp. 146-159 (Feb 1988).

Sha88b. L. Sha, R. Rajkumar and J. P. Lehoczky, ''Concurrency Control for Distributed Real-Time Databases'', *SIGMOD Record*, pp. 82-96 (Vol, 17, No. 1, March 1988).

Sha90. L. Sha, R. Rajkumar and J. P. Lehoczky, ''Priority Inheritance Protocols: An Approach to Real-Time Synchronisation'', *IEEE Transactions on Computers* **39**(9), pp. 1175-1185 (September 1990).

She90. A. Sheth and M. Rusinkiewicz, ''Management of Interdependant Data: Specifying Dependency and Consistency Requirements'', *Proceedings of the Workshop on the Management of Replicated Data*, pp. 133-136 (Nov 1990).

Sim90a. H. Simpson, ''Four-Slot Fully Asynchronous Communication Mechanism'', *IEE Proceedings*, pp. 17-30 (Vol. 137, Pt. E., No. 1, Jan 1990).

Sim90b. H. Simpson, ''Correctness Analysis for a Class of Asynchronous Communication Mechanisms'', *BAe Report* (1990).

Son90. X. Song and J. W. S. Liu, ''Performance of Multiversion Concurrency Control Algorithms in Maintaining Temporal Consistency'', *Dept. of Computer Science, Uni. of Illinois at Urbana-Champaign* (Feb. 1990).