In Proceedings of RTAS'97: The 1997 IEEE Real-Time Technology and Applications Symposium, Montreal, Canada, June 1997.

Admission Control for Soft-Deadline Transactions in ACCORD*

Sue Nagy†
The Open Group
Research Institute
Cambridge, MA 02142
s.nagy@opengroup.org

Azer Bestavros
Computer Science Department
Boston University
Boston, MA 02215
best@cs.bu.edu

Abstract

The use of admission control and overload management techniques in real-time systems has been shown to result in improved system performance—in terms of maximizing the value-added to the system by those transactions committing on time—in comparison to systems which do not employ such techniques. Continuing with our research in hard deadline Real-Time DataBase (RTDB) systems, we investigate the challenges associated with soft deadline transactions and describe a number of admission control and overload management techniques as well as scheduling algorithms appropriate for such systems.

1 Introduction

In our previous work reported in [3], we introduced ACCORD, an Admission Control and Capacity Overload management Real-time Database framework—an architecture and a transaction model—for hard deadline RTDB systems. The system architecture consists of admission control and scheduling components which provide early notification of failure to submitted transactions that are deemed not valuable or incapable of completing on time. The transaction model consists of two components: a *primary task* and a *compensating task*. The execution requirements for the primary task are *not* known *a priori*, whereas those for the compensating task are known *a priori*.

When a transaction is submitted to the system, an *Admission Control Mechanism* (ACM) is employed to decide whether to *admit* or *reject* that transaction. Once admitted, a transaction is guaranteed to *finish* executing before its deadline. A transaction is considered to have finished executing if exactly one of two things occur: either its primary task is completed, in which case we say that the transaction has *successfully committed*, or its compensating task is completed, in which case we say that the transaction has *safely terminated*. A committed transaction brings a *positive* profit to the system, whereas a terminated transaction brings *no* profit. The goal of the admission control and scheduling protocols employed in the system is to *maximize* the profit of those primary tasks which finish on time.

Admission control and overload management techniques preserve system resources by minimizing the likelihood of a transaction being accepted for execution, only to later miss its deadline. Obviously, such a situation cannot totally be eliminated in a system where the execution requirements of transactions are not known *a priori*. Therefore, missing a deadline is always a possibility with which the system must contend. Hence, there must exist some *compensating actions* that, when executed in a timely fashion, would allow the system to be "bailed out" from the consequences of missing a transaction's deadline.

When transactions have *hard* deadlines, transactions must successfully commit or else safely terminate by their deadlines (due to the prohibitive loss to be incurred if deadlines are missed). In this case, the scheduling of compensating tasks is relatively straightforward. We attempt to schedule each compensating task so that it starts at the latest possible time in order to give the corresponding primary task as much time as possible to complete on time. When transactions have soft deadlines, however, then it is possible for the system to finish (commit/terminate) a transaction past its deadline, which makes the problem of *compensating task scheduling* much harder. The question which we address in this research is the following: "How should compensating tasks be scheduled in soft deadline RTDB systems, so that the profit returned to the system is maximized?"

Our research is motivated by research problems in application areas such as robotics, telephone switching systems and the stock market where RTDB systems

*This work has been partially supported by NSF (grant CCR-9706685).

†This work was conducted as part of the author's Ph.D. thesis at Boston University.

are used to store the state of the world (*i.e.* physical components), directory information and financial data, respectively. Compensating actions, busy signals for example, in a telephony application, are needed if the system cannot handle the volume of call requests.

We start in section 2 with a brief overview of our transaction processing model and then in section 3 describe our time-varying value functions for transactions in soft RTDB systems. In section 4, we delineate the compensating task scheduling algorithms. Next, we present our initial simulation results in section 5. We then review in section 6 previous research work and highlight our contributions. We conclude in section 7 with a summary and a description of future research directions.

2 System model

Figure 1 shows the various components in our RTDB system. For a full explanation of all components as well as details of our admission control protocols (*i.e.* workload and concurrency) and scheduling algorithms (*i.e.* First Fit (FF), Latest Fit (LF), Latest Marginal Fit (LMF), Latest Adaptable Fit (LAF), Value Adaptable Fit (VAF)), please refer to [3].

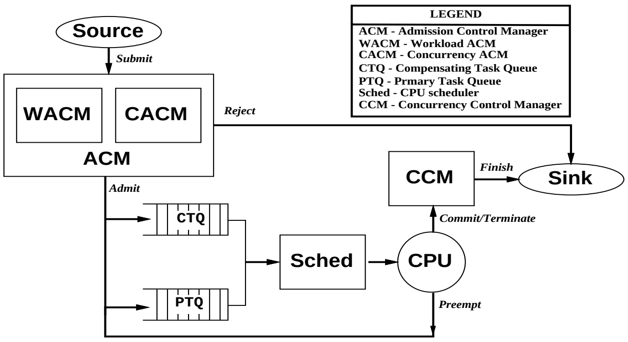

Figure 1: Major System Components

3 Value functions

Jensen, Locke and Tokuda [6] introduced the notion of transactions' values as a function of time. Each transaction T_i is associated with a value function $V_i(t)$ which represents the value of T_i at time t. In hard deadline systems, typically $V_i(t)$ is some constant value v_i until D_i, the deadline of the transaction. After this time, v_i tends towards negative infinity, indicating the catastrophic consequences of missing a hard deadline. For soft deadline transactions, the value of a transaction may decay until some point in time, denoted Z_i, at which the value of the transaction is zero, *i.e.* there is no benefit in continued execution of this transaction as it adds no value to the system. Moreover, execution of a transaction passed "zero point" results in a negative added to the system.

Similar to the work of Bestavros and Braoudakis in [2], we define the *penalty gradient* of a transaction to be the rate at which the transaction's value decays over time.

Definition 1 *The* penalty gradient *of a transaction T_i with a value function of $V_i(t)$ and a deadline D_i is defined as:*

$$\frac{d}{dt}V_i(t), \quad \text{for } t > D_i.$$

In soft RTDB systems, we use the penalty gradient as in indication of how soft deadlines are relative to one another. In order to characterize the rate at which transactions loose their value, we employ the following value function.

Definition 2 *The* value function $V_i(t)$ *of a transaction T_i with an arrival time of A_i and a soft deadline of D_i is defined as:*

$$V_i(t) = \begin{cases} v_i & \text{if } A_i \leq t \leq D_i \\ v_i - [(t - D_i)\tan\alpha_i] & \text{if } t > D_i \end{cases}$$

where v_i is the value-added to (profit of) the system iff T_i completes its execution before its deadline D_i, and $\tan\alpha_i$ *is its penalty gradient.*

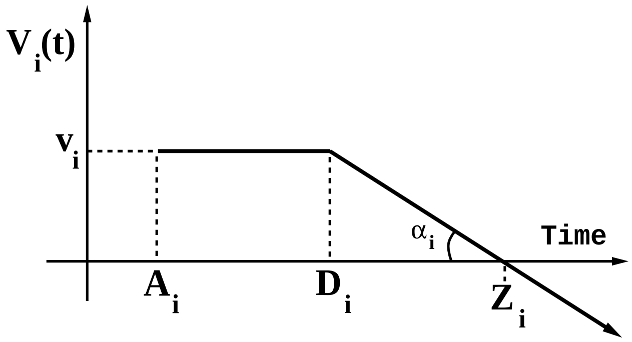

Figure 2: Typical Value Function

Figure 2 depicts a typical value function for transaction T_i. The value of T_i is v_i from time A_i up to D_i. After time D_i, the value decays at a rate corresponding to the $\tan\alpha_i$. The penalty gradient of T_i may vary from zero for non-critical (non-real-time) transactions ($\alpha_i = 0$) to infinity for very critical transactions ($\alpha_i = \pi/2$).

4 Compensating task scheduling

As stated previously, when transactions have soft deadlines rather than hard deadlines, the compensating task scheduling problem is more challenging. Unlike in hard deadline RTDB systems, where the only option available for scheduling a compensating task is so that it completes by the deadline of the transaction, with soft deadline RTDB systems, compensating tasks may be scheduled to start *at any time* between the arrival time of the transaction and an undetermined future time. The powerful sense of urgency associated with transactions having hard deadlines is no longer present.

Direct extensions of hard deadline transaction management

There are a number of compensating task scheduling options to choose from which we discuss below.

1. AD: Schedule the compensating task so that it starts At the Deadline, D_i.
2. AZ: Schedule the compensating task so that it starts At the Zero value time, Z_i.
3. FZ: Schedule the compensating task so that it starts Following the Zero value time, Z_i.

With AD, primary tasks which successfully commit will do so by the deadline of the transaction, and hence they will return the maximum value to the system. In addition, transactions will generally remain in the system for shorter periods of time, and consequently have less chances of wasting system resources (*e.g.* CPU) only to abort. However, consider transactions which are aborted and return no value to the system. If these same transactions were kept in the system past their deadlines, as with AZ or FZ, they would have had more opportunity to successfully commit and return some diminished value to the system. The trade-off, though, is that transactions will typically remain in the system for longer periods of time. AD scheduling basically transforms soft deadlines into hard deadlines.

With AZ, although transactions may remain in the system past their deadlines, there is the chance of gaining some diminished value. Even though this value may be less than that if the transaction had committed before its deadline, the value returned to the system is greater than if the compensating task had safely terminated by the deadline, a possible scenario with AD.

With FZ, in the event that a primary task commits past the zero value time, the result would be the return of a *negative* value to the system, given the time-varying value function as described in figure 2. The actual value lost by the system, when a transaction T_i completes past its zero point Z_i, depends on how far past the deadline the transaction completes. The time that a transaction completes (*i.e.* successfully commits/safely terminates) is in part determined by where the corresponding compensating task is scheduled. Certainly compensating tasks could be scheduled at distant future times, using FZ, thereby allowing primary tasks to *eventually* commit. The price that is paid for these successful commitments is the potentially large, negative value incurred by the system. However, the return of negative value is not necessarily an undesirable outcome especially in situations where there is a corresponding *large, positive profit* to be gained by the commitment(s) of other high-valued transaction(s). Rather than execute the compensating tasks of low-valued transactions—which takes away processor time from the primary tasks of other high-valued transactions—we permit less profitable transactions to remain in the system and execute their primary tasks when doing so will not negatively effect the more profitable transactions.

Generalized framework for soft deadline transaction management

There are a number of major differences between soft and hard deadline systems which dictate the appropriate manner to schedule compensating tasks. With hard deadline systems, compensating tasks must have higher priority than primary tasks and also cannot be preempted since all admitted transactions *must* either successfully commit or safely terminate by their deadlines. With soft deadline transactions, compensating tasks do not necessarily have higher priority than primary tasks and may also be preempted. In evaluating the performance of soft deadline RTDB systems, our focus shifts from maximizing the number (or sum of values) of transactions which complete on-time to maximizing the value that is returned to the system by transactions which successfully commit, both before as well as *after* their deadlines. Consequently, the scheduling algorithm used to apportion the CPU time to primary tasks must be changed as EDF [8], which we employed for hard deadline transactions, is no longer appropriate.

The Primary Task Queue (PTQ) is now organized according to Dynamic Highest Value (DHV) which operates as follows. Periodically we determine, for each transaction T_i whose PT_i is queued in the PTQ, $V_i(t)$, *i.e.* the value of the primary task of transaction T_i at the current time t, and reorganize the PTQ, as necessary. The periodicity of this CPU reorganization can be initiated every δ time units by a daemon process or upon the occurrence of a certain event, such as a transaction submission. Compensating tasks are still maintained in their own separate queue, the Compensating Task Queue (CTQ) which is again ordered according to ascending start time. However, the value, which is also the priority for scheduling purposes, associated with each compensating task is 0—the same as the value returned to the system upon their safe termination.

As each transaction T_i is submitted to the system, we must determine where in the processor's schedule to place its compensating task. As an initial solution to this problem which has an infinite number of solutions, we schedule each compensating task using AZ so that it starts at Z_i, the time at which the value of T_i equals 0, *i.e.* $V_i(t) = Z_i$. When workload admission control (WACM) is employed, we then calculate the processor load from A_i, the arrival time of T_i up to Z_i. If the amount of time occupied by currently scheduled compensating tasks in this interval of time violates the WACM threshold, we reject T_i, otherwise, we admit T_i.

With soft deadline RTDB systems, compensating tasks do not necessarily have to be executed by the deadline of the transaction nor by the point in time at which the value of the transaction is zero or even negative. When exactly should compensating tasks be executed? Given that compensating tasks have lower priority than all primary task, compensating tasks will never be executed. Hence, we need some process which will trigger the possible execution of compensating tasks. As we periodically determine the current value of each transaction for scheduling purposes, we can also periodically evaluate, for each admitted transaction, whether to execute its compensating task at the *current time* or to wait to execute its compensating task until *a later time*. The approach here is similar to one used by Bestavros and Braoudakis in [2] in which the net value of committing a transaction as soon as it validates (with OCC-BC concurrency control) is compared with net value of deferring its commitment until a future time. If the value-added to the system by committing the transaction now is greater than the value-added by committing it later, the transaction is committed now; otherwise, it is committed later. We discuss an overview of our method below.

In executing the compensating task of a transaction at the current time, we neither gain nor loose any value from this safe termination. However, we do *potentially* loose the value that the primary task could have return upon its successful commitment (or correspondingly could have incurred additional loss if the primary task had committed past the zero value point in time), should the compensating task not have been executed. By removing this transaction from the system, though, other admitted transactions will have less competition for system resources, and as a result, may potentially return more value to the system should they successfully commit. On the other hand, waiting to execute the compensating task of a transaction till a later time results in 1) the possibility of this transaction successfully committing and returning some diminished value to the system, and 2) admitted transactions having to continue to compete for system resources with this transaction—possibly lessening their value.

By quantitatively analyzing these two options—Execute Compensating Task Now (ECTN) and Wait to execute Compensating Task until Later (WCTL)—the final decision of when to execute a compensating task is influenced by the potential value-added to the system. If the value of ECTN is greater than WCTL, then we execute the compensating task now, otherwise we execute the compensating task at a later point in time.

5 Performance evaluation

The RTDB system model used in our experiments consists of a uniprocessor system with a 1000-page, memory-resident database. A second CPU is dedicated to supporting both admission and concurrency control protocols. Our baseline simulation parameters are as follows. The primary task of each transaction reads 16 pages selected at random with a 25% update probability. The CPU time needed to process a read or a write is 2.5 ms. Thus, in the absence of any data or resource conflicts, the primary task of each transaction would need a *serial execution time* of 50 ms CPU time.[1] The compensating task of each transaction follows a normal distribution with a mean of 10 ms and standard deviation of 5 ms—amounting to an average of 4 page accesses. Transaction deadlines were related to the *serial execution time* through a *slack factor*, such that *(deadline time - arrival time)* = *slack factor* $\times$ *serial execution time*.

The transaction inter-arrival rate, which is drawn from an exponential distribution, is varied from 5 transactions per second up to 50 transactions per second in increments of 5, which represents a light-to-medium loaded system. We used two additional

[1] Notice that these figures (*i.e.* number of pages accessed and serial execution time) are only needed to generate the workload fed to the simulator. They are *not* known to the ACM.

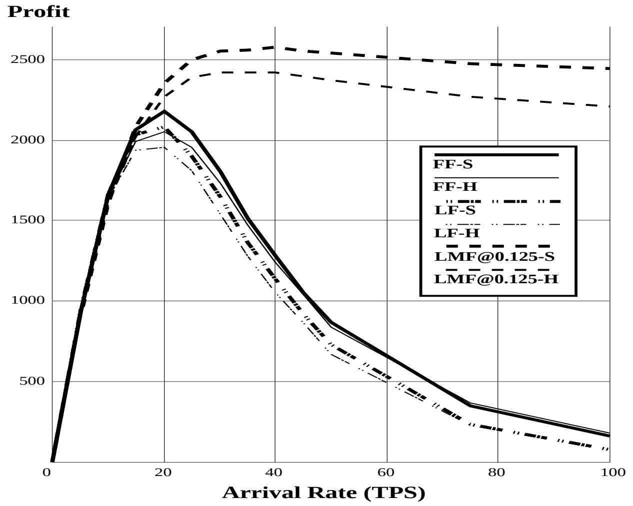

Figure 3: Total Value Realized

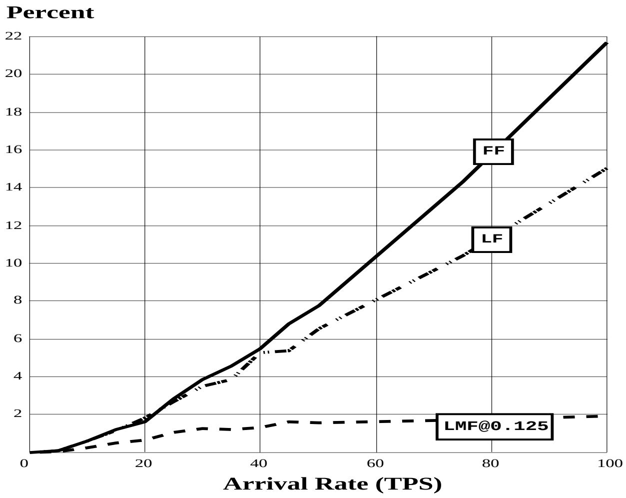

Figure 4: Percentage Missed Deadlines

arrival rates of 75 and 100 transactions per second to experiment with a very heavy loaded system. The primary task scheduling protocol was EDF while the compensating task scheduling protocol was FF, LF, and LMF. The threshold used with LMF was 0.125. We employed Optimistic Concurrency Control with forward validation—OCC-BC [10]. All transactions were from the same transaction class, *i.e.* have the same transaction characteristics such as constant value of 1 before the deadline (all are equally important) and penalty gradient of $\alpha_i = 45$ degrees $\forall\ T_i$. For our initial baseline results, we scheduled compensating tasks using AZ in order to see the performance gains achievable with the simplest scheduling technique. Each simulation was run four times, each time with a different seed, for 200,000 ms. The results depicted are the average over the four runs.

In figure 3, we see the total value (profit) realized by the system by those transactions which successfully committed. Specifically, we compare our previous hard deadline results (denoted by -H) with our soft deadline results (denoted -S). With the non-admission control protocols of FF and LF, the results in the soft deadline case are slightly better in light-to-medium loaded systems and nearly the same in heavy loaded systems. However, with LMF@0.125, which employs an admission control mechanism, the results for the soft deadline system are markedly improved, especially in moderately-to-heavily loaded systems . Since transaction deadlines are soft, we are also interested in the number of transactions which missed their deadlines out of the total number of transactions which successfully committed. We see in figure 4 that the percentage of transactions which successfully committed between D_i (the deadline) and Z_i (zero value) (*i.e.* missed their deadlines) is very small over all system loads for LMF@0.125. Almost all transactions which successfully committed were able to do so by their deadlines with LMF@0.125. As a result of admission control and our compensating task scheduling technique, we have seen that in soft deadline RTDB systems, we are better able to utilize system resource for those transactions admitted to the system, we realize more profit and miss fewer deadlines (*i.e.* complete more transactions on time).

6 Related work

Our work differs from previous research in that our transaction model incorporates not only primary tasks, with unknown WCET, but also compensating tasks. There have been a number of similar transaction models suggested in the literature, and these are contrasted with our model below.

Liu *et al.* [9] developed the *imprecise computation* model which decomposes each task into two subtasks, a mandatory part and an optional part. Others employing this model include Audsley *et al.* [1] and Davis *et al.* [5]. Our model differs from the imprecise computation model in that the WCET requirements for the mandatory *and* optional parts are assumed in [9, 1, 5], whereas they are assumed only for the compensating tasks in our model. Also, unlike the imprecise computation model, we start off with the execution of the optional component (the primary task), leaving the mandatory component (the compensating task) to a later time (if needed). In a sense, our paradigm is complementary to the imprecise computation paradigm.

The primary/alternative model was employed by Liestman and Campbell [7] and by Chetto and Chetto [4]. In [7] *primary* tasks provide good quality of service and are preferable to *alternative tasks* which produce acceptable quality of service and handle primary tasks' timing faults. Our notion of a compensating task is indeed similar to that of an alternative; execution of a compensating task provides less attractive quality of service in comparison to the execution of the primary task. The similarities end here, however. Alternative tasks in [7] are not subject to timing failures, whereas in our model compensating tasks may have hard, soft or firm deadlines. Moreover, in [4], alternative tasks are periodic in nature, unlike compensating tasks which are not.

In [11], Tew *et al.* introduce a task model with two components: a *load task* and an *execute task* whereby the load task first loads the task from disk into memory thereby making the execute task eligible to run (*i.e.* there is a precedence relation between the two tasks). The task model of Tew *et al.* is similar to our transaction model. Both models consist of a main task (primary task, execute task). However, the motivation for having the second component differs. Our compensating task is necessitated by the fact that the read/write sets and WCETs of primary tasks are nondeterministic, whereas Tew *et al.* are interested in accounting for loading a task into memory.

A number of papers utilize transaction values and value functions. Like [2, 12], we also use time-varying value functions to indicate how soft deadlines are relative to each other. In addition, we use value functions in order to determine where to schedule compensating tasks, which are not present in the transaction models of the other two papers.

7 Summary and future work

In this paper, we presented simple algorithms for scheduling compensating tasks in soft RTDB systems for ACCORD. Our initial results confirm even more firmly our earlier conclusions drawn in hard deadline systems: Admission control and overload management techniques improve system performance by rationing system resources and minimizing the likelihood of a transaction being accepted for execution, only to later miss its deadline. We presented the difficulties and challenges associated with scheduling compensating tasks for soft deadline systems, and we sketched a generalized framework for such systems. However, the methodology must still be further refined.

References

[1] N. C. Audsley, R. I. Davis, and A. Burns. Mechanisms for enhancing the flexibility and utility of hard real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 12–21, December 1994.

[2] Azer Bestavros and Spyridon Braoudakis. Value-cognizant speculative concurrency control. In *Proceedings of VLDB'95: The International Conference on Very Large Databases*, Zurich, Switzerland, Spetember 1995.

[3] Azer Bestavros and Sue Nagy. Value-cognizant admission control for rtdb systems. In *RTSS'96: The 17th Real-Time Systems Symposium*, pages 230–239, Washington, D.C., December 1996.

[4] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[5] R. I. Davis, S. Punnekkat, N. Audsley, and A. Burns. Flexible scheduling for adaptable real-time systems. In *Proceedings of the Real-Time Technology and Applications Symposium*, pages 230–239, May 1995.

[6] E. Jensen, C. Locke, and H. Tokuda. A time-driven scheduling model for real-time operating systems. In *Proceedings of the 6th Real-Time Systems Symposium*, pages 112–122, December 1985.

[7] A. Liestman and R. Campbell. A fault-tolerant scheduling problem. *IEEE Transaction on Software Engineering*, SE-12(11):1089–1095, November 1986.

[8] C. L. Liu and J. Layland. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the Assocation of Computing Machinery*, 20(1):46–61, January 1973.

[9] J. W.-S. Liu, K. J. Lin, and S. Natarajan. Scheduling real-time, periodic jobs using imprecise results. In *Proceedings of the 8th IEEE Real-time Systems Symposium*, December 1987.

[10] D. Menasce and T. Nakanishi. Optimistic versus pessimistic concurrency control mechanisms in database management systems. *Information Systems*, 7(1), 1982.

[11] Ken Tew, Panos K. Chrysanthis, and Daniel Mosse. Emperical evaluation of task and resource scheduling in dynamic real-time systems. In *Proceedings of RTSS'96 WIP Session: The 17^{th} IEEE Real-Time System Symposium*, pages 35–38, Washington, D.C., December 1996.

[12] S.-M. Tseng, Y.H. Chin, and W.-P. Yang. Scheduling real-time transactions with dynamic values: a performance evaluation. In *Proceedings Second International Workshop on Real-Time Computing Systems and Applications*, pages 60–67, October 1995.