In Proceedings of RTSS'96: The 16th IEEE Real-Time System Symposium, Washington, DC, Dec 1996.

Value-cognizant Admission Control for RTDB Systems*

Azer Bestavros
(best@cs.bu.edu)

Sue Nagy
(nagy@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215

Abstract

Admission control and overload management techniques are central to the design and implementation of Real-Time Database Systems. In this paper, we motivate the need for these mechanisms and present protocols for adding such capabilities to Real-Time Databases. In particular, we present a novel admission control paradigm, we describe a number of admission control strategies and contrast (through simulations) their relative performance.

1 Introduction

The main challenge involved in scheduling transactions in a Real-Time DataBase (RTDB) management system is that the resources needed to execute a transaction are not known *a priori*. For example, the set of objects to be read (written) by a transaction may be dependent on user input (*e.g.* in a stock market application) or dependent on sensory inputs (*e.g.* in a process control application). Therefore, the *a priori* reservation of resources (*e.g.* read/write locks on data objects) to guarantee a particular Worst Case Execution Time (WCET) becomes impossible—and the non-deterministic delays associated with the on-the-fly acquisition of such resources pose the real challenge of integrating scheduling and concurrency control techniques.

Current real-time concurrency control mechanisms resolve the above challenge by relaxing the *deadline* semantics (thus suggesting best-effort mechanisms for concurrency control in the presence of *soft* and *firm*, but not *hard* deadlines), or by restricting the set of acceptable transactions to a finite set of transactions with execution requirements that are known a priori (thus reducing the concurrency control problem to that of resource management and scheduling).[1]

Consider the huge body of research on real-time concurrency control, where complex time-cognizant concurrency control techniques are proposed for the sole purpose of maximizing the number of transactions that meet their deadlines (or other metrics thereof). A careful evaluation of these elaborate techniques reveals that their superiority is materialized only when the RTDB system is overloaded. However, when the system is not overloaded, the performance of these techniques becomes comparable to that of much simpler techniques (*e.g.* 2PL-PA). It is important to observe that when a RTDB system is overloaded, a large percentage of transactions end up missing their deadlines. This observation leads to the following question: How better would be the performance of the system if these same transactions (that ended up missing their deadlines) were not allowed into the system in the first place? The answer is obviously "*much better*" because with hindsight, the limited resources in the system would not have been wasted on these transactions to start with. While such a clairvoyant scheduling of transactions is impossible in a real system, admission control and overload management techniques could be used to achieve the same goal. In this paper, we introduce and evaluate such techniques.

Admission control and overload management techniques preserve system resources by minimizing the likelihood of a transaction being accepted for execution, and later not being able to meet 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. For transactions with firm deadlines, such a situation is tolerable because commitment past a firm deadline is of no value. However, for transactions with hard (soft) deadlines, such an abortion is disastrous because missing a hard (soft) deadline results in an (eventual) infinite loss.[2] Thus, to support transactions with hard deadlines without *a priori* knowledge of their execution requirements, there must exist some compensating actions that, when executed in a timely fashion, would allow the system to be "bailed out" from the disastrous consequences of missing a hard deadline.

*This work has been partially supported by NSF (grant CCR-9308344).

[1] In this paper, we do not consider approaches that attempt to relax ACID properties—serializability in particular.

[2] Most RTDB systems avoid dealing with the consequences of missing a hard deadline by restricting the class of transactions they manage to those with either firm or soft deadlines.

Our research is motivated by research problems in application areas such as the stock market and robotics. Consider, for example, industrial automation processes which commonly employ robots, typically in a hazardous environment. Here, a real-time database is used to represent the state of the world, *i.e.* the location of the robot arms and of the physical components which are manipulated by the robot's arms. The robot may be required to complete a transaction (an atomic set of actions) by a specified time before proceeding to the next one. Compensating actions are needed, for example, if a transaction that is about to miss its deadline must be terminated safely (requiring the clearing of the workspace, for example).

We start in section 2 with an overview of our transaction processing model and the different components therein. Next, in section 3 we describe the various Admission Control Strategies to be used in our simulations. Next, in section 4 we present and discuss our simulation baseline model and results as well as results of our value-cognizant protocol. In section 5, we review previous research work and highlight our contributions. We conclude in section 6 with a summary and a description of future research directions.

2 System Model

Each transaction submitted to the system 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*.[3] Figure 1 shows the various components in our RTDB system.

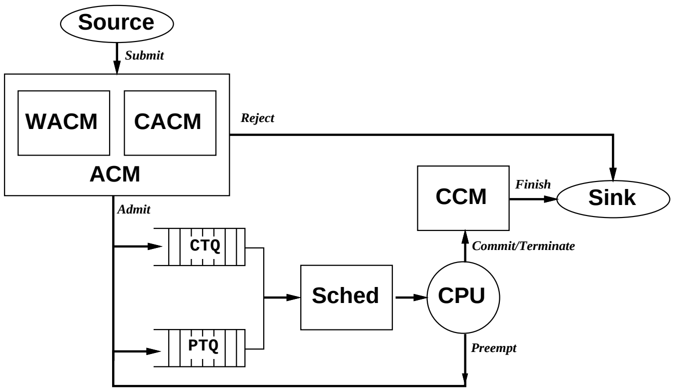

Figure 1: Major System Components

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* profit.

When submitted to the system, each transaction is associated with a *deadline* and a *value*. The value of a transaction represents the profit that the system makes if the transaction is successfully committed (*i.e.* its primary task is committed by its deadline). In this paper we consider only hard deadlines and thus assume that no transaction will finish (*i.e.* successfully commit or safely terminate) past its deadline.[4] We initially assume that all transactions bring in equal profit when committed on time, and then consider the case where the profits of transactions differ. Moreover, once admitted to the system, a transaction is absolutely guaranteed (as opposed to conditionally guaranteed) to finish and cannot now be rejected in order to accommodate a newly submitted transaction.

The ACM consists of two major components: a *Concurrency Admission Control Manager* (CACM) and a *Workload Admission Control Manager* (WACM). The CACM is responsible for ensuring that admitted transactions do not overburden the system by requiring a level of concurrency that is not sustainable. The WACM is responsible for ensuring that admitted transactions do not overburden the system by requiring computing resources (*e.g.* CPU time) that are not sustainable.

Compensating tasks are executed when a transaction with a hard deadline is deemed incapable of committing by its deadline. Due to the urgency associated with the execution of such compensating tasks, we assume a 2-tier priority scheme for CPU scheduling purposes. In particular, all compensating tasks are assumed to have a higher priority than primary tasks. Thus a primary task may be preempted (or aborted) by a compensating task, whereas a compensating task cannot be preempted by either a primary task or another compensating task under any condition. Notice that this 2-tier priority assumption still allows primary tasks (compensating tasks) to be prioritized amongst themselves.

In this paper we study our admission control mechanism in conjunction with two types of concurrency control protocols, namely Optimistic Concurrency Control with forward validation (such as OCC-BC [20] or SCC-nS [3]), or Pessimistic Concurrency Control with Priority Abort (such as 2PL-PA [1]).

Workload Admission Control Manager: The source contains a set of transactions which are generated

[3] While the execution time of a transaction's primary task is not known *a priori*, we assume that this execution time cannot exceed the difference between the transaction's deadline and its submission time.

[4] Our current research involves extending our results to soft and firm deadline systems by allowing for a profit/loss past a transaction's deadline. This is similar to our work in [4].

off-line. Each enters the system at a random time and is first processed by the ACM. The decision of whether to admit or reject a transaction submitted for execution is based upon a feedback mechanism that takes into consideration the current demand on the resources in the system. This decision is motivated by the overall goal for maximizing profit by maximizing the number—or sum of the values—of successful commitments (when primary tasks finish) and minimizing the number of safe terminations (when compensating tasks finish). For example, if the percentage of the CPU bandwidth already committed to compensating tasks (of admitted primary tasks), within the interval from the current time to the deadline of the submitted transaction is high, it may be prudent for the WACM to reject the submitted transaction. For transactions which successfully pass through the admission control process, the WACM attempts to schedule the compensating task in the Compensating Task Queue (CTQ) whose organization is discussed later in this section. Even if the current demand on the system's resources is low, a transaction is rejected if it is not feasible to schedule its compensating task (*e.g.* it cannot be accommodated in the CTQ).

Concurrency Admission Control Manager: In order to ensure that compensating tasks can execute unhindered (and thus complete within their WCETs) the CACM must guarantee that the admission of a transaction into the system does not result in data conflicts between the compensating task of that transaction and other already admitted transactions. In a uniprocessor system employing an Optimistic Concurrency Control (OCC) algorithm with forward validation (*e.g.* OCC-BC), compensating tasks (which cannot be preempted) are guaranteed to finish execution without incurring any restart delays. The same is true of a uniprocessor system employing a Pessimistic Concurrency Control (PCC) algorithm with priority abort (*e.g.* 2PL-PA) because compensating tasks execute at a higher priority than primary tasks and, thus, are guaranteed to finish execution without incurring any blocking delays. This is not true in a multiprocessor system, where multiple compensating tasks may be executing concurrently. In such a system, the CACM ensures that only those compensating tasks that do not conflict with each other are allowed to overlap when executed.

Processor Scheduling Algorithm: There are two queues managed by the processor scheduler: the Primary Task Queue (PTQ) and the Compensating Task Queue (CTQ). Each admitted transaction contributes one entry in each of these queues. A primary task is ready to execute as soon as it is enqueued in the PTQ, whereas a compensating task must wait for its start time, specified by the ACM. As indicated before, compensating tasks execute at a priority higher than that of the primary tasks. Thus, the scheduling algorithm will always preempt a primary task in favor of a compensating task which is ready to execute.

Since all tasks in the PTQ are ready to execute, a scheduling algorithm must be used to apportion the CPU time amongst these tasks. We use the *E*arliest *D*eadline *F*irst algorithm (EDF) [17], which is optimal for a uniprocessor system with independent, preemptible tasks having arbitrary deadlines [9].

The CTQ is organized as a series of slots, one for each compensating task. Each slot contains the compensating task id as well as its start and end times. Slots are order according to ascending start time. The CPU continues to service primary tasks until all are finished or a compensating task must begin executing, *i.e.* its start time has arrived. In the later case, the primary task currently using the CPU is preempted and enqueued back into the PTQ where it awaits further processing, if the compensating task is associated with a different primary task. Otherwise, the primary task is aborted and its compensating task executes.

Concurrency Control Manager The function of the CCM is to enforce the concurrency control protocol in use. For OCC techniques, this enforcement is done at the time a transaction finishes its execution, either by the commitment of its primary task or by the safe termination of its compensating task. In the case of OCC-BC, conflicting (primary tasks of) transactions are restarted, whereas in the case of SCC-nS, conflicting (primary tasks of) transactions are rolled back to a point preceding the conflicting action. For PCC techniques, this enforcement is done at the time of each read/write request. For compensating tasks, which execute at a higher priority, such a request is always granted. This may result in aborting/restarting conflicting primary tasks. Notice that it is impossible for two compensating tasks to conflict since the processor scheduler guarantees that compensating tasks do not overlap.[5] For primary tasks, such a request may result in blocking (if the read/write lock is not available).

3 Optimizing Profit through ACM

In order to maximize the value added to the system from the successful commitment of transactions, the ACM must admit *"enough"* transactions—but not too many—to make use of the system capacity. Admitting too many transactions results in the system being overloaded, which results in having to be content with most transactions safely terminating (*i.e.* not successfully committing), which minimizes the profit to the system. We use the term *thrashing* to coin this condition (*i.e.* the system is busy, yet doing nothing of value).

As indicated before, the main determinant of whether transactions are admitted into the system is the schedulability of compensating tasks. In this section we present a number of techniques that could be used by the WACM and contrast their performance.

First-Fit (FF) Using this technique, the compensating task of a transaction is inserted in the CTQ at the

[5]This condition is true in any uniprocessor system where compensating tasks cannot be preempted.

latest slot that satisfies its WCET. If no slot is big enough to fit the compensating task, then the transaction is rejected, otherwise it is admitted.

Latest-Fit (LF) Using this technique, the compensating task of a transaction is inserted in the CTQ at the latest slot. If the slot is not large enough, then the compensating tasks preceding that slot are rescheduled to start at earlier times so as to "make room" for the new compensating task. If this rescheduling is not possible—because it leads to a compensating task having to be rescheduled before the current time—then the transaction is rejected, otherwise it is admitted.

Latest-Marginal-Fit (LMF) This technique is identical to Latest-Fit, except that the scheduling of a compensating task—and, if necessary, the ensuing rescheduling of other compensating tasks—is conditional on whether or not the percentage of CPU time allotted to compensating tasks[6] is below a preset margin or threshold. If compensating tasks scheduled so far utilize CPU bandwidth above that margin, then the transaction is rejected, otherwise Latest-Fit (as described before) is attempted.

Latest-Adaptable-Fit (LAF) This technique is identical to Latest-Marginal-Fit, except that the threshold used to gauge the CPU bandwidth allotted to compensating tasks is set dynamically, based on measured variables, such as arrival rate of transactions, distribution of computation times for successfully committed primary tasks as it relates to the distribution of computation times for compensating tasks, probability of conflict over database objects (*e.g.* transaction read/write mix).

Both FF and LF continue to admit transactions into the system as long as compensating tasks are schedulable. In other words, there is no feedback mechanism (admission control) that would prevent thrashing. LMF implements such a mechanism by refraining from admitting new transactions, once the percentage of CPU bandwidth allocated to compensating tasks reaches a preset *static* threshold. LAF does the same, but allows that threshold to be determined dynamically using a table lookup procedure. The table is computed off-line (using simulations) to determine the *optimum* quiescent value for the threshold under a host of other parameters.

Both LMF and LAF, however, do not take into consideration transactions' values during the admission control process. When transactions return different profits to the system upon their successful completion, the ACM must be *value-cognizant.* We describe below VAF, a value-cognizant admission control protocol. Like LAF, VAF utilizes a threshold to estimate the CPU bandwidth allocated to compensating tasks, but VAF allows that threshold to be adjusted according to the submitted transaction's value.

[6]within a window of time determined by the current time and the deadline of the submitted transaction

Value-Adaptable-Fit (VAF) Given no *a priori* knowledge of primary tasks' WCET and read/write sets, and that only (1) the accumulated CPU time used by each transaction and (2) the execution requirements of the compensating tasks are known, the admission control mechanism must use a heuristic to guide it in determining whether to admit or reject a transaction. In using a heuristic, we attempt to predict the likelihood of a newly submitted transaction being able to successfully commit by its deadline, given the competition for system resources with previously admitted transactions (with both their primary and compensating tasks). Initially, we consider the CPU resource only.

Specifically, our technique consists of a 2-tiered admission control decision based on: (1) the profit-margin for the submitted transaction, and (2) the bandwidth-margin for the system.

The *profit-margin component* evaluates the expected profit from admitting a transaction into the system. This is done by weighing the potential gain achievable by admitting the transaction against the potential loss incurred by previously admitted transactions as a result of such an admission. To estimate the potential gain (loss) for a transaction T_i, we introduce the Commit Likelihood Index, $CLI(T_i)$, which reflects the confidence of the system in being able to successfully commit transaction T_i by its deadline.

The $CLI(T_i)$ is composed of the product of two indicators. The first indicator is the Accumulated Primary Task execution time (APT) and the second indicator is the Expected Compensating Task execution time (ECT). The $APT(T_i)$ measures the CPU bandwidth accumulated so far in the original window utilized by T_i. A value close to 1 is indicative of a transaction that is likely to have already executed most of its primary task, while a value close to 0 is indicative of a transaction that is not likely to have already executed much of its primary task. The $ECT(T_i)$ anticipates the effect of the future execution of compensating tasks of other transactions on the execution of T_i. Specifically,

$$\begin{aligned} CLI(T_i) &= APT(T_i) * ECT(T_i) && (1)\\ APT(T_i) &= \frac{CW_i}{OW_i - PT_i} && (2)\\ ECT(T_i) &= \left(1 - k * \frac{\sum_{j,j\neq i} CT_j}{CW_i}\right) && (3) \end{aligned}$$

where

- CW_i is the current window of T_i, *i.e.* the difference between the *current* starting time of the compensating task for T_i and the *current* time,[7]
- OW_i is the original window of T_i, *i.e.* the difference

[7]The compensating task for T_i may have been rescheduled in the CTQ after T_i is admitted.

between the *original* starting time of the compensating task for T_i and its admission time,

- PT_i is the CPU time used so far by the primary task of T_i,
- CT_j is the future CPU time reserved for the compensating task of transaction T_j, and
- k is a parameter that speculates as to the likelihood of an admitted transaction having to execute its compensating task. In particular, we set $k = 0$ if we adopt the *optimistic* assumption that *all* admitted transactions will successfully commit, and we set $k = 1$ if we adopt the *pessimistic* assumption that *no* admitted transactions will successfully commit. A value of $0 < k < 1$ denotes a *speculative* assumption that only a fraction k of *all* admitted transactions will successfully commit.

With regard to the $APT(T_i)$, as the CW_i shrinks with the passage of time, either the PT_i steadily increases—T_i is being serviced by the CPU and hence we are *more confident* that T_i will finish by its deadline—or PT_i remains constant (or increases rather slowly)—T_i is receiving little, if any, CPU time and hence we are *less confident* that T_i will finish by its deadline. Concerning the $ECT(T_i)$, an optimistic assumption that all admitted transactions will successfully commit results in the $ECT(T_i)$ not having any effect on the $CLI(T_i)$ while pessimistic and speculative assumptions decrease the $CLI(T_I)$, *i.e.* as a result of the compensating tasks of admitted transactions executing, we are less confident that the newly submitted transaction T_i will be able to successfully commit by its deadline.

During the admission control process, for each submitted transaction T_i, we compute the value of $CLI(T_i)$. In addition, we compute $CLI(T_j)$ for each previously admitted T_j whose current window CW_j would intersect CT_i. These indices, which incorporate transactions' values, are used to estimate the potential profit (loss) from admitting T_i as follows:

$$VGain = CLI(T_i) * V(T_i) \quad (4)$$

$$VLoss = \sum_{j, j \neq i} (CLI(T_j) - CLI^*(T_j)) * V(T_j) \quad (5)$$

where

- $V(T_i)$ is the profit (value) gained by the system if T_i successfully commits, and
- $CLI^*(T_j)$ is the new commit index of T_j if T_i is admitted.

In computing the $VLoss$, we take into consideration the difference between the previous value of $CLI(T_j)$ and the current value of $CLI^*(T_j)$, *i.e.* is the likelihood of T_j successfully committing less now than previously as a result of (possibly) admitting a new transaction T_i? If the $VLoss > VGain$, we reject T_i since it doesn't promise a positive overall profit to the system if admitted. Alternately, transactions that successfully pass through this profit-margin component of VAF admission control are deemed useful to the system (since they promise a positive profit if admitted) and thus are moved on to the *bandwidth-margin component* of VAF, which is presented below.

Similar to LAF, the threshold of CPU bandwidth allocated to compensating tasks is obtained using a table lookup procedure. However, unlike LAF, VAF takes $OriginalThreshold$, returned by the table lookup procedure, and dynamically computes $NewThreshold$ according to $V(T_i)$, as follows:

if $(V(T_i)/MeanValue > 1.0)$
    $NewThreshold = 1.0$
else
    $NewThreshold = OriginalThreshold$

where $MeanValue$ is the average value (profit) of the transaction mix.

For those transactions which are not rejected by the profit-margin component and have higher value with respect to the transaction set (are more profitable to the system), we raise $NewThreshold$ to 1.0, *i.e.* admit this transaction. Transactions which pass through the profit-margin component but are less profitable use $OriginalThreshold$.

For less profitable transactions, in computing $NewThreshold$, we use the $OriginalThreshold$ making it more difficult for these transactions to be accepted. On the other hand, we raise $OriginalThreshold$ to 1.0 for more profitable transactions as we stand to gain more by successfully completing these transactions. If compensating tasks scheduled so far utilize CPU bandwidth above $NewThreshold$, then the transaction is rejected. Otherwise LF scheduling is attempted.

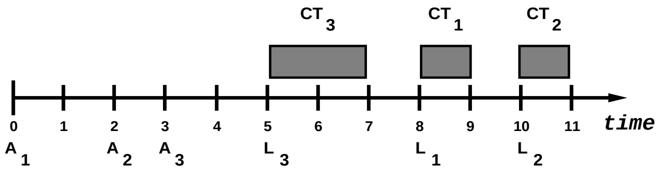

Figure 2: VAF example

Figure 2 shows how the VAF admission control protocol works. Suppose that A_i is the arrival time of transaction T_i, and L_i is the latest time at which CT_i can begin its execution (so as to complete by the deadline of T_i). Three transactions are submitted to the system: T_1 with value 1 at time 0, T_2 with value 10 at time 2 and T_3 with value 1 at time 3. Note that although their compensating tasks, CT_1, CT_2, and CT_3, are all shown in the figure, each would be added at the time of the corresponding primary task's arrival. We assume that k is set to 1, *i.e.* pessimistic VAF, and that EDF scheduling is followed.

Time	T_i	*CW*	*OW*	*PT*	$\sum CT$	*CLI*	CLI^*	*VGain*	*VLoss*
0	T_1	8	8	0	0	0	1	1	0
2	T_1	6	8	2	0	1	1		
	T_2	8	8	0	1	0	0.875	8.75	0
3	T_1	5	8	3	2	1	0.6		
	T_2	7	8	0	3	0.875	0.5		
	T_3	2	2	0	0	0	1	1	4.15

Both T_1 and T_2 pass through the profit-margin component of VAF since $VGain$ is greater than the $VLoss$. However, T_3 is rejected from the system basically due to its compensating task's effect on higher valued transaction T_2. Consider, however, T_3 to have value 10 like T_2. $VGain$ would now be 10 resulting in T_3 proceeding to the bandwidth-margin component of VAF.

Techniques other than the one above could also be used to calculate $NewThreshold$. For example, instead of using $V(T_i)$, we could factor in the net profit, *i.e.* the differential between the the $VGain$ and the $VLoss$. In situations where the difference is large (*i.e.* $VGain \gg VLoss$), $NewThreshold$ would be raised, while when the difference is small (*i.e.* $VGain \approx VLoss$), $NewThreshold$ would be lowered.

4 Performance Evaluation

We have implemented the above ACM policies for a uniprocessor system using either OCC-BC or 2PL-PA. In the first part of this section, we show the value of admission control by comparing the performance achievable through FF, LF, LMF, and LAF. Since we assume that all transactions bring in equal profit when committed before their deadlines, we desire to maximize the number of primary task completions while minimizing the number of compensating task completions (*i.e.* primary task abortions). In the second part of this section, we show the performance of the value-cognizant VAF technique in comparison to the non-value-cognizant technique of LAF. The superior results of VAF demonstrate the advantage of utilizing the value of a transaction in the admission control process and as well as in the computation of the CPU bandwidth threshold. For all simulation experiments performed, we assumed the existence of a second CPU dedicated to supporting the admission control and concurrency control protocols.

4.1 Baseline Experiments

Table 1 shows the baseline parameters for our simulations. We assume a 1000-page memory-resident database. The primary task of each transaction reads 16 pages (`Xsize`) 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.[8] The compensating task of each transaction follows a normal distribution with a mean (`CTCompTime`) of 10 ms and standard deviation (`CTStdDev`) of 5 ms.[9] Transaction deadlines were related to the *serial execution time* through a *slack factor*, such that *(deadline time - arrival time)* = `SlackFactor` × *serial execution time.*

Parameter	Value
`ArrivalRate`	5 - 100 TPS
`DBsize`	1,000
`Xsize`	16
`CPUTime`	2.5 ms
`UpdateProb`	0.25
`CTCompTime`	10 ms
`CTStdDev`	0.5 `CTCompTime`
`SlackFactor`	2
`TaskSchd`	EDF
`CTSchd`	FF, LF, LMF
`Thrsh`	0.125
`CCntrl`	OCC-BC

Table 1: Baseline Workload Parameters

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 arrival rates of 75 and 100 transactions per second to experiment with a very heavy loaded system. `TaskSchd`, the primary task scheduling protocol, was EDF while `CTSchd`, the compensating task scheduling protocol was FF, LF, and LMF. The threshold used with LMF, `Thrsh`, was 0.125. 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.

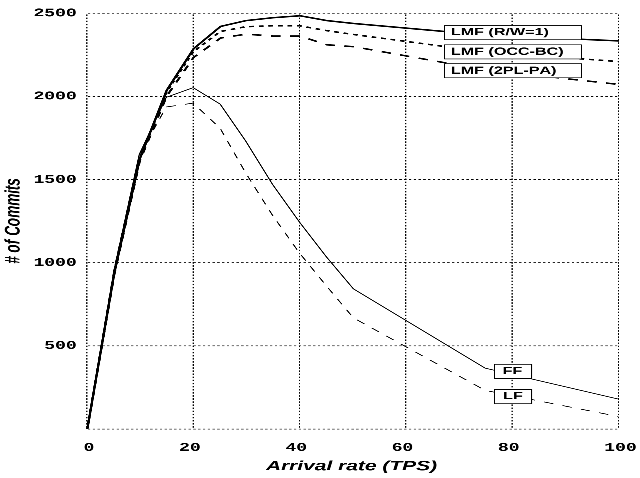

Figure 3: Performance of FF, LF, and LMF

Figure 3 shows the absolute number of successfully committed transactions, which is a measure of the *value-*

[8]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.

[9]This amounts to an average of 4 page accesses.

added to (or *profit* of) the system, under the baseline parameters shown in table 1. Under light-to-medium loads (arrival rates < 15 TPS), the performance of FF and that of LF are identical. Under medium-to-heavy (arrival rates > 15 TPS) loads FF performs slightly better. This is expected due to LF's tighter packing of compensating tasks via rescheduling, which results in the admission of more transactions, thus resulting in a more pronounced thrashing behavior. Under light-to-medium loads, the performance of LMF is indistinguishable from that of FF or LF, but under medium-to-heavy loads LMF manages to avoid thrashing, thus keeping the system's profit in check with its capacity.

We performed three simulations under the LMF policy. In the first, we used OCC-BC as the concurrency control protocol. In the second, we used 2PL-PA as the concurrency control protocol. In the third, we set the write probability to 0 (*i.e.* read/write mix = 1; all transactions are "read-only"), thus simulating the performance of LMF in the absence of data conflicts. These simulations, illustrated in Figure 3, show that LMF is most beneficial when data conflicts are least. Also, it shows that LMF is more beneficial with OCC-BC than it is with 2PL-PA. This could be explained by noting that OCC techniques are better suited for systems with controllable utilization [11], which is the case in a system with admission control like ours.

The value of the threshold to be used in LMF is key to its performance. As we explained before, the optimal value for this threshold depends on many parameters, most of which cannot be estimated *a priori*. One such parameter is the arrival rate of transactions. To demonstrate this, we ran a set of experiments using LMF, in which we varied the value of the threshold and the transaction arrival rates. Our results show that for lightly-loaded systems (arrival rates less than 10 TPS), the performance is unimodal, thus any threshold less than 1 is not optimal. This implies that at such low loads all transactions should be admitted, making the performance of LMF identical to that of LF. For moderately-loaded and heavily-loaded systems, our results indicate that an optimum threshold exists for each arrival rate. Setting the threshold to that optimal value yields the highest percentage of successful commitments, and thus yields the highest possible profit. The sensitivity of the profit to the value of that threshold is much more pronounced under heavy loads (*e.g.* 30-100 TPS) than it is under more moderate loads (*e.g.* 15-25 TPS).

To evaluate the effect of dynamically changing the threshold in LAF, we ran a simulation of the system, in which we varied the arrival rate. The parameters used were identical to those in table 1, except that the update probability was set to zero (thus making these results independent of the concurrency control protocol in use). Our simulation consisted of 5 consecutive epochs, each running for 50,000 ms, for a total of 250 seconds. The arrival rate of transactions in these epochs was set to 15, 25, 35, 45, and 75 TPS, respectively.

Figure 4 shows the performance of LAF against that of LMF for two threshold values: 0.125 and 0.250. For each one of the three mechanisms, we plotted the mean number of successful commitments observed over periods of 10,000 ms, thus yielding five measurements per epoch for each mechanism (shown in Figure 4 as a scatter plot). These data points were used to fit a curve to characterize the performance of each mechanism over the full 250 seconds of simulation. Overall, the performance of LAF is better than both LMF (@ 0.125) and LMF (@ 0.25). As expected, when the system is lightly loaded, the performance of LMF (@ 0.25) is close to that of LAF, whereas the performance of LMF (@ 0.125) is meager as a result of its unduly restrictive admission control. When the system is heavily loaded, the performance of LMF (@ 0.125) is close to that of LAF, whereas the performance of LMF (@ 0.25) is meager as a result of its excessively lax admission control. When the system is moderately loaded, the performance of all three techniques is similar.

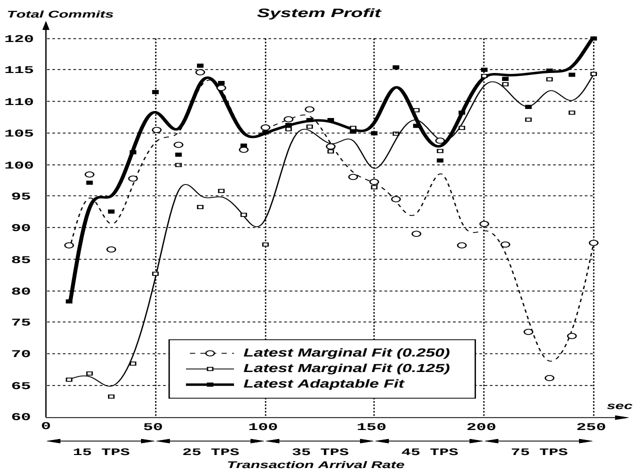

Figure 4: Dynamic Performance of LMF and LAF

In the above experiment, only the arrival rate of transactions changes from one epoch to the other, and as a result, LAF was allowed to adapt its threshold value to a single parameter, namely the arrival rate of transactions. In other words, LAF optimized the value of its threshold along a single dimension.

In a typical system, more than one parameter is likely to change over time. LAF could be easily used in such systems by allowing it to optimize the value of its threshold along multiple dimensions. In particular, assuming n different dimensions (*e.g.* observed average arrival rate, average slack factor, average read/write mix, and average compensating task length, among others), then using off-line simulation experiments, the optimum threshold value for each node in an n-dimensional mesh could be evaluated for later use by LAF in a manner similar to that shown in figure 4. The identification of the appropriate dimensions for this optimization process is an interesting research problem.

To illustrate the above process, consider the case in which three parameters—namely, the arrival rate, the slack factor and the compensating task computation time—are likely to change and that LAF has to adapt to these changes dynamically.[10] The first step involves the evaluation of the optimum threshold value for each node in a 3-dimensional mesh. Table 2 shows the different values we considered along each one of these dimensions. All other parameters were identical to those in table 1 except that the update probability was set to zero, *i.e.* all transactions were "read-only".

Parameter	Value
`ArrivalRate`	5 - 50 by 5's, 60, 80, 100 TPS
`CTCompTime`	4, 8, 20, 40 ms
`SlackFactor`	1.5, 2.0, 3.0, 4.0

Table 2: Parameters' Ranges for 3-dimensional mesh

We ran 4 simulations for each setting of `ArrivalRate`, `CTCompTime`, and `SlackFactor`—a total of 208 combinations, or 832 simulations. This process was repeated for a number of threshold values in order to compute the *optimal* value per setting. The bisection method [15] was used in order to determine the optimal threshold value for each `ArrivalRate`, `CTCompTime`, `SlackFactor` triplet.

To evaluate the relative performance of LAF, we ran a set of experiments in which LAF optimized the value of its threshold along all 3 dimensions using the results from the above experiments. The workload for each experiment was constructed by fixing the value along one dimension to emulate a different workload (wrkld) as described in table 3.

Wrkld	Description	Constant Parameter
`Wrkld 0`	Random	none
`Wrkld 1`	Lax Deadlines	`SlackFactor` - 4.0
`Wrkld 2`	Tight Deadlines	`SlackFactor` - 1.5
`Wrkld 3`	High `ArrivalRate`	`ArrivalRate` - 100 TPS
`Wrkld 4`	Low `ArrivalRate`	`ArrivalRate` - 10 TPS
`Wrkld 5`	Long `CTCompTime`	`CTCompTime` - 40 ms
`Wrkld 6`	Short `CTCompTime`	`CTCompTime` - 4 ms

Table 3: Workload Descriptions

Each experiment consisted of 20 consecutive epochs of 4 sec each for a total running time of 80 sec. At the beginning of each epoch, the values of the parameters were set according to the specifications above. For example, under Wrkld 3, at the beginning of each epoch, the `SlackFactor` and `CTCompTime` were chosen at random and used for transactions generated during that epoch, while the `ArrivalRate` remained at 100 TPS. All workloads were run 4 times—once for each of LMF (@ 0.1), LMF (@ 0.3), LMF (@ 0.8), and LAF. The profits achievable by each one of these compensating task scheduling techniques, for each workload is shown in figure 5.

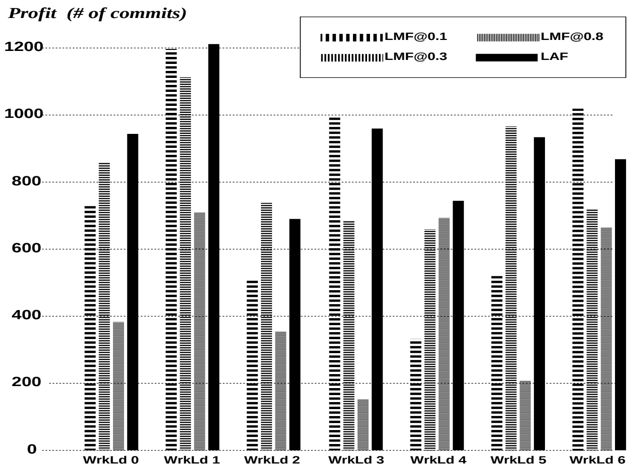

Figure 5: Profits achievable by LMF and LAF in a dynamic environment

Figure 5 shows that LAF achieves the most profit when all 3 parameters are allowed to change (wrkld 0). Under all other workloads, LAF achieved either the best profit or the second best profit. More importantly, unlike the other LMF techniques, LAF shows *consistent* performance.

4.2 Value-cognizant Results

The baseline parameters used for the value-cognizant VAF simulation experiments are identical to those in table 1 with the exception of two additional higher `ArrivalRate`s of 200 and 300 TPS. Moreover, for this set of experiments, transactions were grouped into two different classes based on their relative value. Transactions in the first (less critical) class have a value of 1, whereas those in the second class (more critical) have a value of 10. Transactions in Class-I made up 90% of the load while transactions in Class-II made up the remaining 10% of the load.

Figure 6 shows the results of our baseline simulations for VAF. Two sets of curves are shown. The first shows the profit realized using Pessimistic VAF and using LAF. The second shows the unrealized profit—profit that had to be given up by the admission control protocol—for Pessimistic VAF and for LAF. The results clearly show that VAF outperforms LAF, especially when the system is not underutilized. For example at an arrival rate of 100 TPS, the Pessimistic VAF admission control results in 55% more profit when compared to LAF admission control. The difference between the unrealized profits of VAF and LAF is even more compelling. In particular, VAF seems to be able to *choose*

[10] One could also vary other parameters, such as the transaction length (*i.e.* number of pages read), or the write probability.

the right transactions to admit so as to maximize the realized profit (and thus minimize the unrealized profit).

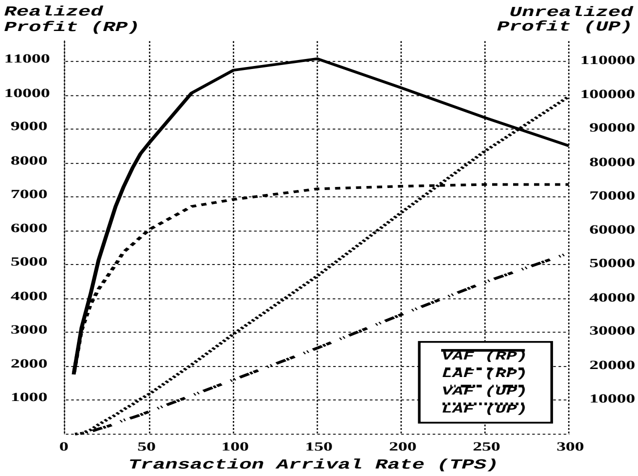

Figure 6: VAF versus LAF - Baseline

We have performed other experiments in which the differential between the value of transactions in Class-I and Class-II is increased to 100-fold (instead of 10-fold). In other words, 90% of the workload consisted of transactions with value 1 and 10% of the workload consisted of transactions with value 100. These experiments indicated that the increase in profit achievable through the use of VAF over LAF is larger (more than 100% greater profit in heavily-loaded systems).

5 Related Work

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

Liu *et al.* [18] developed the *imprecise computation* model which decomposes each task into two subtasks, a mandatory part and an optional part. Audsley *et al.* [2] and Davis *et al.* [8] developed mechanisms whereby mandatory, hard deadline tasks are given off-line guarantees of minimum quality of service while associated optional tasks are guaranteed on-line, if sufficient resources are available. Our model differs from the imprecise computation model in that the WCET requirements for the mandatory *and* optional parts are assumed in [18, 2, 8], 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.

A similar model was also considered by Liestman and Campbell [16] and by Chetto and Chetto [7]. In [16] *primary* tasks provide good quality of service and are preferable to *alternative tasks* which produce acceptable quality of service and handle timing faults. Our notion of a compensating task is indeed similar to that of an alternative, with one major difference. Alternative tasks in [16] are not subject to timing failures, whereas in our model compensating tasks have hard deadlines. In [7], alternative tasks are periodic in nature, unlike compensating tasks which are not.

Admission control protocols and feedback mechanisms have been employed in a variety of RTDB system components: transaction scheduling [12], memory allocation for queries [21], and B-tree index concurrency control [10]. Haritsa *et al.* [12] incorporate a feedback mechanism into an Adaptive Earliest Deadline (AED) and Hierarchical Earliest Deadline (HED) scheduling strategies for transactions in a firm deadline environment. Both AED and HED attempt to stabilize the overload performance of EDF. The focus of Pang *et al.* [21] is on admission control and memory management of queries requiring large amounts of computational memory in a firm RTDB system. The admission control component of their Priority Memory Management algorithm dynamically sets the target MPL by using a feedback process based upon information from previously completed queries. Goyal *et al.* [10] describe an approach that allows transactions to be rejected as part of an optimization of the Load Adaptive B-link algorithm, a real-time version of index (B-tree) concurrency control algorithms in firm-deadline RTDB systems.

In many systems, the assumption that all transactions are of equal value is not valid, making it necessary to consider the worth of a transaction, when making resource allocation and conflict resolution decisions. In such systems, the performance objective becomes that of maximizing the system *profit*. The notions of transaction values and value functions [14, 19] have been utilized in both general real-time systems [5, 6] as well as in RTDB [4, 12, 13, 22]. In [5, 6], the value of a task is evaluated during the admission control process. Huang *et al.* [13] use transactions' values to schedule system resources (*e.g.* CPU) and in conflict resolution protocols in a soft real-time environment. Extending their AED scheduling algorithm to be value-cognizant, Haritsa *et al.* [12], developed Hierarchical Earliest Deadline (HED) for firm RTDB systems. All of the aforementioned research make use of transactions' values which are time-invariant.

A different approach is taken by Bestavros and Braoudakis [4] and Tseng *et al.* [22]. In [4], value functions are employed in a soft real-time system to determine whether it is more advantageous to commit a transaction or to delay that commitment for a period of time. Like [4], Tseng *et al.* use time-variant value functions in their Highest Reward First (HRF) scheduling algorithm for firm RTDB systems.

6 Summary and Future Work

In this paper, we proposed a new paradigm for the execution of transactions in a RTDB system. Our paradigm allows the system to *reject* a transaction that is submitted for execution, or else *admit* it and thus *guarantee* that one of two outcomes will occur by the transaction's deadline: either the transaction will successfully commit through the execution of a primary task, or the transaction will safely terminate through the execution of a compensating task. The system assumes no *a priori* knowledge of the execution requirements of the primary task, but assumes that the WCET and read/write sets of the compensating task are known. Through the use of appropriate admission control policies, we show that it is possible for the system to maximize its profit dynamically.

In this paper, we considered only hard-deadline transactions. This implied that once admitted, a transaction must be successfully committed, or else safely terminated by its deadline (due to the prohibitive loss to be incurred if that deadline is missed). If soft-deadline transactions are to be managed, 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 interaction between concurrency control and admission control is one of the main themes of this paper. Yet, many facets of this interaction have not been addressed. For example, the CCM could use information provided to the CACM to make better concurrency control decisions. Conversely, the CACM could use information about the read/write sets of primary tasks to determine whether or not to accept a particular compensating task.

References

[1] Robert Abbott and Hector Garcia-Molina. Scheduling real-time transactions: A performance evaluation. In *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 1–12, Los Angeles, Ca, 1988.

[2] 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.

[3] Azer Bestavros and Spyridon Braoudakis. Timeliness via speculation for real-time databases. In *Proceedings of RTSS'94: The 14^{th} IEEE Real-Time System Symposium*, San Juan, Puerto Rico, December 1994.

[4] 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.

[5] Sara Biyabani, John Stankovic, and Krithi Ramamritham. The integration of deadline and criticalness in hard real-time scheduling. In *Proceedings of the 9th Real-Time Systems Symposium*, December 1988.

[6] G. Buttazzo, M. Spuri, and F. Sensini. Value vs. deadline scheduling in overload conditions. In *Proceedings of the 16th Real-Time Systems Symposium*, December 1995.

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

[8] 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.

[9] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proceedings IFIP Congress*, pages 807–813, 1974.

[10] B. Goyal, J. Haritsa, S. Seshadri, and V. Srinivasan. Index concurrency control in firm real-time dbms. In *Proceedings of the 21st VLDB Conference*, pages 146–157, September 1995.

[11] Jayant R. Haritsa, Michael J. Carey, and Miron Livny. On being optimistic about real-time constraints. In *Proceedings of the 1990 ACM PODS Symposium*, April 1990.

[12] Jayant R. Haritsa, Miron Livny, and Michael J. Carey. Earliest deadline scheduling for real-time database systems. In *Proceedings of the 12th Real-Time Systems Symposium*, December 1991.

[13] J. Huang, J. A. Stankovic, D. Towsley, and K. Ramamritham. Experimental evaluation of real-time transaction processing. In *Proceedings of the 10th Real-Time Systems Symposium*, December 1989.

[14] 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 Symbosium*, pages 112–122, December 1985.

[15] Lee W. Johnson and R. Dean Riess. *Numerical Analysis.* Addison Wesley, 1982.

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

[17] 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.

[18] 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.

[19] C. Locke. *Best Effort Decision Making for Real-Time Scheduling.* PhD thesis, Carnegie-Mellon University, Department of Computer Science, May 1986.

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

[21] H. Pang, M. J. Carey, and M. Livny. Managing memory for real-time queries. In *Proceedings of the 1994 ACM SIGMOD Conference on Management of Data*, pages 221–232, 1994.

[22] 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.