In Proceedings of AAAI'96: The Fall Symposium on Flexible Computation in Intelligent Systems, Cambridge, MA, November 1996.

An Admission Control Paradigm for Value-cognizant Real-Time Databases*

Azer Bestavros (best@cs.bu.edu)
Sue Nagy (nagy@cs.bu.edu)

Computer Science Department
Boston University
Boston, MA 02215

Abstract

We propose and evaluate an admission control paradigm for RTDBS, in which a transaction is *submitted* to the system as a pair of processes: a *primary task* and a *compensating task*. The execution requirements of the primary task are *not* known *a priori*, whereas those of the compensating task are known *a priori*. Upon the submission of a transaction, an *Admission Control Mechanism* 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 (*successful commitment*), or its compensating task is completed (*safe termination*). Committed transactions bring a profit to the system, whereas a terminated transaction brings *no* profit. The goal of the admission control and scheduling protocols (*e.g.* concurrency control, I/O scheduling, memory management) employed in the system is to *maximize* system profit. 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 Management System (RTDBS) 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 nondeterministic 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]

In this paper, we propose and evaluate, through simulation experiments, a paradigm that preserves the hard deadline semantics without assuming complete a priori knowledge of transaction execution requirements. 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.

Our research is motivated by research problems in application areas such as the stock market and robotics. Consider, for example, automated financial trading [32, 33] in which security transactions are made to buy and sell stocks (as well as other types of securities). Information concerning each stock (*e.g.* stock id, description, current market price, broker dealers, broker dealers' shares, etc.) is stored in a database. Since a transaction is dependent upon user input, the set of database objects to be read and written cannot be determined a priori.

Security transactions are submitted to the system for execution and for those which are accepted, the trade, also known as the purchase or sale of securities, is executed. The settlement date or deadline of a security transaction is the time by which the transfer of the securities (for the seller) or the cash (for the buyer) must be completed. The primary task of each security transaction performs the following operations: verifies the trade order, transmits the order, checks for an order match between the seller and buyer, and performs any necessary follow-up actions. If any of these operations fail, the compensating task, which is scheduled to start[2] prior to the settlement date, is executed and the corresponding primary task is aborted. The compensating task, which possess knowledge (*i.e.* buyer/seller name, address, phone number, email address, etc.) concerning the buyer and seller, will notify both parties that the trade has not gone through.

*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] The exact start time is determined by calculating the time to process the read/write set of the compensating task and subtracting that sum from the transaction's deadline.

Another motivating application is industrial automation processes [11] which commonly employ robots, typically in hazardous environments. 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 moved by the robots arms. The robot may be required to complete certain actions by a specified time before proceeding to the next set of actions. Compensating actions are needed, for example, if a robot arm drops the object that it is holding (*e.g.* mechanical grasp failure) or if one arm is moving but its path is obstructed by another arm; we must be able to recover from these potentially precarious situations. We assert that the assumption of *a priori* knowledge of compensating actions for these types of applications is a fair one.

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. 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 RTDBS.

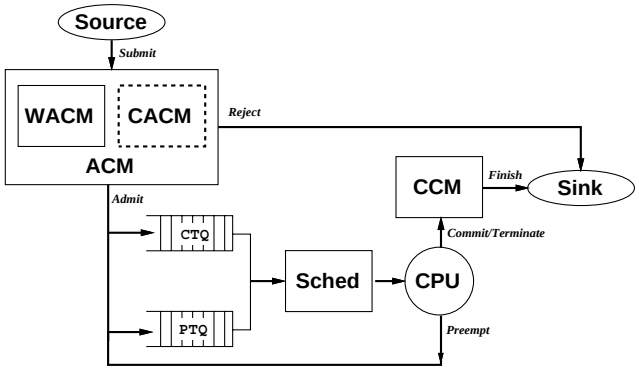

Figure 1: Major System Components

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] Also, we assume that all transactions bring in equal profit when committed on time. 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.

In this paper we assume that an Optimistic Concurrency Control (OCC) Algorithm with forward validation (such as OCC-BC [27] or SCC-nS [2]) is used to ensure serializability. OCC techniques are better suited for systems with controllable utilization [12], which is the case in a system with admission control like ours.[5]

We adopt a 2-level priority scheme to schedule system resources (*e.g.* CPU). 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.

2.1 Workload Admission Control Manager

The source contains a set of transactions which are generated 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 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 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 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 in section 2.3. Even if the demand on the system's

[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 [3].

[5] Our choice of an OCC-based algorithm is not crucial for the purpose of this paper. In particular, all of our algorithms could be adapted to a Pessimistic Concurrency Control (*e.g.* 2PL-HP).

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).

2.2 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 OCC algorithm with forward validation, compensating tasks (which cannot be preempted) are guaranteed to finish execution without incurring any restart 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. The dotted box around the CACM in figure 1 denotes that it has not yet been implemented.

2.3 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) [24], 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.

2.4 Concurrency Control Manager

As each transaction finishes its execution, either by the commitment of its primary task or by the safe termination of its compensating task, the CCM must ensure that all other active transactions (*i.e.* primary tasks admitted to the system) that have data conflicts with the finished transaction are handled according to the concurrency control protocol in effect. In the case of OCC-BC, conflicting (primary tasks of) transactions are restarted whereas with SCC-nS, we rollback the (primary tasks of) transactions to a point preceding the conflicting action. There is no need to check for data conflicts with other compensating tasks in either a uni or multiprocessor system. In the former case, only 1 compensating task is active at a time, and in the later case, the CACM prohibits compensating tasks which access the same data to overlap their executions. Upon the successful commitment of a primary task, the CCM removes the corresponding compensating task from the CTQ and marks its slot as free. All transactions, whether finished or rejected, are removed from the system and sent to the sink which generates statistical information used to evaluate the system performance.

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 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 alloted 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

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

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.

4 Performance Evaluation

We have implemented the above ACM policies for a uniprocessor system using OCC-BC. In 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).

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 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.[7] The compensating task of each transaction follows a normal distribution with a mean of 10 ms and standard deviation of 5 ms. This amounts to an average of 4 page accesses. Transaction deadlines are related to the *serial execution time* through a *slack factor*, such that *(deadline time - arrival time)* = `SlackFactor` × *serial execution time*.

Parameter	Meaning	Value
`ArrivalRate`	Transaction arrival rate	5 - 100 TPS
`DBsize`	Database size in pages	1,000
`Xsize`	Number of reads/transaction	16
`CPUTime`	CPU time per page access	2.5 ms
`UpdateProb`	Update Probability	0.25
`CTCompTime`	Mean Compensating Time	10 ms
`CTStdDev`	St. Dev. of CT Time	0.5 `CTCompTime`
`SlackFactor`	Slack factor	2
`TaskSchd`	Task scheduling protocol	`EDF`
`CTSchd`	CT scheduling protocol	FF, LF, LMF
`Thrsh`	CT computation threshold	0.125
`CCntrl`	Concurrency Control protocol	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. 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 2 shows the absolute number of successfully committed transactions, which is a measure of the *value-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.

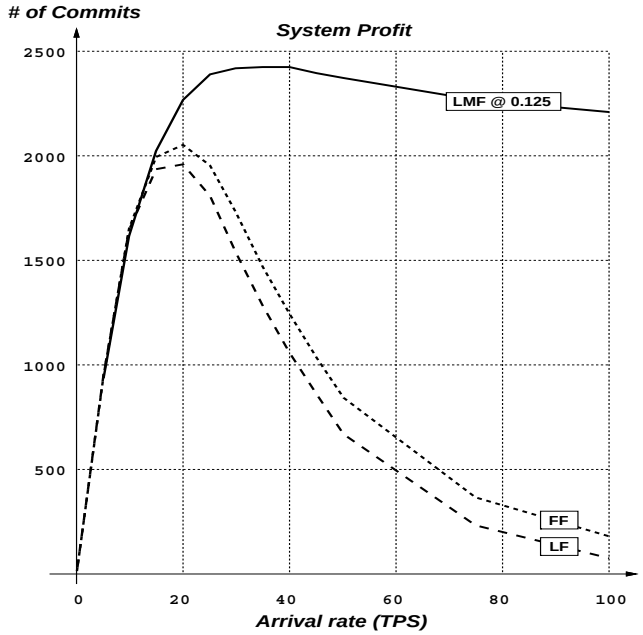

Figure 2: Performance of FF, LF, and LMF

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. Figure 3 shows the percentage of submitted transactions that was successfully committed by LMF.

Figure 3 shows 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, Figure 3 indicates 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

[7] 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.

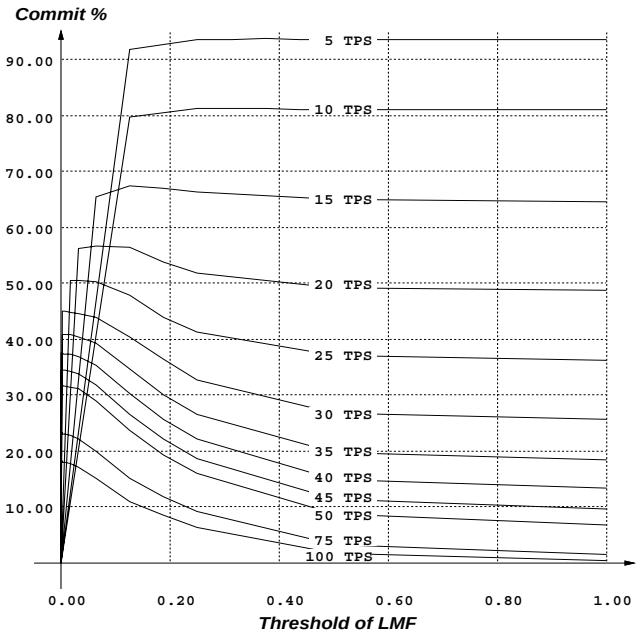

Figure 3: Effect of threshold setting on LMF performance

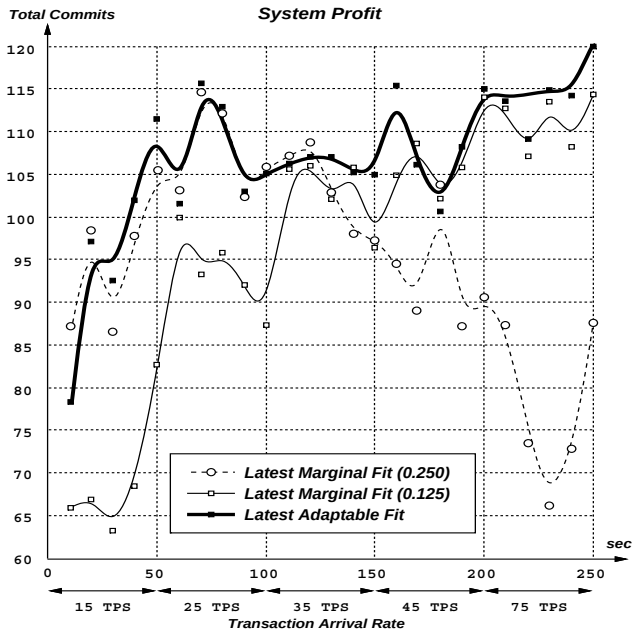

Figure 4: Dynamic Performance of LMF and LAF

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 the 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, respectively.

Figure 4 shows the performance of LAF against that of LMF for two threshold values: 0.125 and 0.25. 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.

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 (such as the one portrayed in figure 3), 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.[8] 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 dimension. 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`,

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

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 [17] 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 (WrkLd) for each experiment was constructed by fixing the value along one dimension to emulate a different workload as described in table 3.

WrkId	Description	Constant Parameter
WrkLd 0	Random	none
WrkLd 1	Lax Deadlines	SlackFactor - 4.0
WrkLd 2	Tight Deadlines	SlackFactor - 1.5
WrkLd 3	High Arrival Rate	ArrivalRate - 100 TPS
WrkLd 4	Low Arrival Rate	ArrivalRate - 10 TPS
WrkLd 5	Long Compensating Tasks	CTCompTime - 40 ms
WrkLd 6	Short Compensating Tasks	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 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.

5 Related Work

The performance objective in most previous RTDBS studies has been to minimize the number of transactions that miss their deadlines in a hard or firm deadline environment, or to minimize tardiness, *i.e.* the time by which late transactions miss their deadlines, in a soft deadline environment. The assumption in these systems is that all transactions are of equal value. In many systems, this assumption 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 notion of transaction values and value functions [16, 26] have been utilized in both general real-time systems [4, 6] as well as in RTDBS [1, 15, 34]. In [4, 6], the value of a task is evaluated during the admission control process. The decision to reject a task or remove a previously guaranteed task is based upon tasks' values. A task which

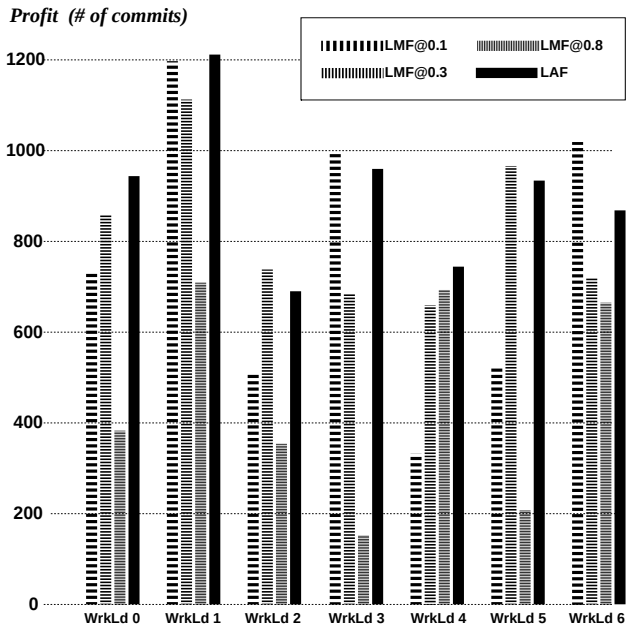

Figure 5: Profits achievable by LMF and LAF in a dynamic environment

is accepted into the system is *conditionally* guaranteed[9] to complete its execution provided that no higher valued (critical) task (with which it conflicts) arrives. In all cases, the WCET of the tasks is assumed to be known *a priori*. Huang *et al.* [15], continuing with the work of [34] use transactions' values to schedule system resources (*e.g.* CPU) and in conflict resolution protocols in a soft real-time environment.

Two recent PhD theses have proposed novel transaction processing frameworks that allow RTDBS to apportion their resources in a value-cognizant fashion. In [19, 18], Kim establishes a RTDBS model which includes both hard and soft real-time transactions, maintains temporal and logical consistency of data [31], and supports multiple guarantee levels. Under this model, an integrated transaction processing scheme is devised, providing both predictability and consistency for RTDBS such that every application in the system is assured to achieve its own performance goal (the guarantee level) and maintain consistency requirement. A simulation study shows that higher guarantee levels require more system resources and therefore cost more than non-guaranteed transactions.

In [5, 3], Braoudakis takes a different approach, whereby transactions are associated with value functions that identify the nature of their timing constraints, as well as their overall importance to the system's mission. Under this framework a whole spectrum of transactions could be specified, including transactions with no timing constraints, as well as transactions with soft, firm, and hard deadlines. The novelty of this approach is that it allows a single transaction processing protocol to be carried uniformly on all types of transactions. The efficacy of this approach has been demonstrated by applying it to the concurrency control problem in RTDBS. In particular, speculative concurrency control algorithms [2] were extended to work under this framework

[9] This is in contrast to an *absolute* guarantee, which specifies that once admitted to a system, the task will complete its execution by its deadline.

and were shown—in detailed simulation studies—to yield superior performance.

Our work differs from previous research in that our system model incorporates not only primary tasks, with unknown WCET, but also compensating tasks. The admission control mechanism used admits transactions into the system with the *absolute guarantee* that either the primary task will successfully commit or the compensating task safely terminate. There have been a number of similar models suggested in the literature. These are contrasted to our model below.

Liu *et al.* [23, 22, 25] describe the *imprecise computation* model which decomposes each task into two subtasks, a mandatory part and an optional part. The mandatory part, which has a hard deadline, must be completed in order for the task to produce an *acceptable* result. The optional part, which has a soft deadline and executes upon completion of the mandatory part, refines the result produced by the mandatory part. The *error* in the result produced by a task is zero if the optional part completes its execution; otherwise, it is equal to the unfinished processing time of the optional part. The goal in this model is to minimize the average error incurred by all tasks. Our work differs from that of Liu *et al.* in that the WCET requirements of the mandatory and optional parts are assumed, and *both* must complete in order to obtain a precise result. Like the mandatory part, a compensating task must execute to completion but only in the event that the primary task incurs a timing failure. Given that a profit is returned to the system only by the successful completion of primary tasks, our preference is to *solely* execute primary tasks. In effect, our model is the transposed version of the imprecise computation model.

A number of papers have employed the *primary / alternative* model in which the primary task provides good quality of service and is preferable to the alternative which produces an acceptable quality of service. Alternatives handle timing faults in [21, 8] and processor failures in [28, 29, 20]. 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. The alternatives in [21] are not subject to timing failures, i.e. they have soft deadlines, whereas compensating tasks have hard deadlines. Moreover, in [8], the alternatives are periodic in nature, unlike compensating tasks which are not.

Admission control protocols and feedback mechanisms have been employed in a variety of RTDBS components: transaction scheduling [13, 14, 7], memory allocation for queries [30], and B-tree index concurrency control [10]. Haritsa *et al.* [13] developed Adaptive Earliest Deadline (AED) into order to stabilize the overload performance of EDF in firm RTDBS. As transactions are submitted to the system, they are either placed into a *Hit Group* or *Miss Group*. The Hit Group is the largest set of transactions which can be completed by their deadline when scheduled according to EDF while the Miss Group contains those transactions whose deadlines are expected to not be met. The key to AED is the determination of the dynamic control variable *Hit Capacity* which demarcates the Hit Group from the Miss Group. The Hit Capacity is calculated by using a feedback mechanism which collects output system measurements from previously completed transactions in order to derive the new size of the Hit Group.

Hong *et al.* [14] introduce the Cost Conscious Approach (CCA) to scheduling transactions in a soft RTDBS. CCA takes into account both static (*i.e.* deadline) and dynamic (*i.e.* effective service time, restart cost) aspects of a transaction's execution when dynamically computing the priority of a transaction. Chakravarthy *et al.* [7] extend CCA to adapt to the system load— CCA-ALF—Cost Conscious Approach with Average Load Factor. Like CCA, CCA-ALF uses both static and dynamic information in calculating the priority of a transaction. In addition, through a feedback mechanism, CCA-ALF incorporates the average load factor of the most recent N completed transaction. Simulation experiments of a multiclass system are performed in which 3 transaction classes are specified based upon the cpu time needed per page access (*i.e.* transaction length is varied). Since only soft deadline transactions are considered, there is no need for an admission control protocol.

The focus of Pang *et al.* [30] is on admission control and memory management of queries requiring large amounts of computational memory in a firm RTDBS. Their Priority Memory Management (PMM) algorithm consists of two components: admission control and memory allocation. The admission control component dynamically sets the target MPL by using a feedback process based upon information from previously completed queries. The memory allocation component also utilized feedback obtained from previously completed queries in order to determine the memory allocation strategy to follow (*i.e.* Max or MinMax).

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 (LAB-link), a real-time version of index (B-tree) concurrency control algorithms in firm-deadline RTDBS. LAB-link ensures that the root of the B-tree (disk) does not become a bottleneck by rejecting transactions when the percentage of transactions missing their deadlines is above a preset threshold. By tuning the system based on the percentage of missed deadlines, their technique does not guarantee a maximum profit. Also, the notion of a guarantee (whether for commitment or safe termination by the deadline) is non-existent in their work.

In all of the above research, the basic system model is one of transactions (or queries) accessing information in the database, after which, each transaction either completes by its deadline or is aborted when its deadline is missed. The only two possible transaction execution outcomes are *commitment* and *abortion*. In [30], when the number of transactions admitted to the system exceeds the MPL, new transactions are made to wait. This non-zero admission waiting time is detrimental to the progress of these transactions completing by their deadlines. The situation is analogous in [10]. When the load control mechanism is active and the utilization of the bottleneck resource is above the preset threshold, new transactions are not allowed to enter the system. Eventually, these transactions are aborted when it is discovered that their deadlines have passed. When AED [13] scheduling algorithm is used, transactions which cannot be assigned to the Hit Group are placed in the Miss Group and are likely to miss their deadlines, especially in moderately to heavily loaded systems.

Most previous RTDBS studies have assumed that the only possible outcome of a transaction execution is either the *commitment* or the *abortion* of the transaction. In many systems, a third outcome of an outright *rejection* may be desirable. For example, in a process control application, the outright rejection of a transaction may be safer then attempting to execute that transaction, only to miss its deadline. Our work allows the system to reject a transaction, thus making it possible for compensating actions to be taken in a timely fashion (possibly by the outside mechanism that

submitted that very same transaction). Also, this flexibility allows the system to ration its resources in the most *profitable* way, by only admitting high-value transactions when the system is overloaded, while being less choosy when the system is underloaded.

6 Conclusion and Future Work

In this paper, we proposed a new paradigm for the execution of transactions in a RTDBS. 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.

In this paper we singled out concurrency control and CPU scheduling as representative activities within a RTDBS. In that respect, we showed how an admission control strategy could be composed with these activities to optimize the system performance dynamically. In a typical RTDBS, other activities must be considered as well. In particular, the admission control decisions may depend not only on the CPU capacity and/or on the CCM capacity to deal with data conflicts, but also on the capacity of other RTDBS components, such as the I/O scheduler, memory manager, and index concurrency control manager. Such a generalized admission control manager is under development.

References

[1] Robert Abbott and Hector Garcia-Molina. Scheduling real-time transactions. *ACM, SIGMOD Record*, 17(1):71–81, 1988.

[2] 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.

[3] 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.

[4] 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.

[5] Spyridon Braoudakis. *Concurrency Control Protocols for Real-Time Databases*. PhD thesis, Computer Science Department, Boston University, Boston, MA 02215, November 1994.

[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] S. Chakravarthy, D. Hong, and T. Johnson. Incorporating load factor into the scheduling of soft real-time transactions. Technical Report TR94-024, University of Florida, Department of Computer and Information Science, 1994.

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

[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] Mikell P. Groover. *Industrial Robotics: Technology, Programming, and Applications*. McGraw-Hill, 1986.

[12] 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.

[13] 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.

[14] R. Hong, T. Johnson, and S. Chakravarthy. Real-time transaction scheduling: A cost conscious approach. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pages 197–206, December 1993.

[15] 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.

[16] 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.

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

[18] Y. Kim and S. H. Son. An approach towards predictable real-time transaction processing. In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, pages 70–75, Oulu, Finland, June 1993.

[19] Young-Kuk Kim. *Predictability and Consistency in Real-Time Transaction Processing*. PhD thesis, Department of Computer Science, University of Virginia, May 1995.

[20] C. M. Krishna and K. G. Shin. On scheduling tasks with a quick recovery from failure. *IEEE Transactions on Computers*, 35(5):448–455, May 1986.

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

[22] K. J. Lin, S. Natarajan, and J. W.-S. Liu. Imprecise results: Utilizing partial commputations in real-time systems. In *Proceedings of the 8th IEEE Real-Time Systems Symposium*, December 1987.

[23] K. J. Lin, S. Natarajan, J. W.-S. Liu, and T. Krauskopf. Concord: A system of imprecise computations. In *Proceedings of the IEEE Compsac*, October 1987.

[24] 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.

[25] 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.

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

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

[28] D. Mosse, R. Melhem, and S. Ghosh. Analysis of a fault-tolerant multiprocessor scheduling algorithm. *IEEE Fault Tolerant Computing*, pages 16–25, 1994.

[29] Y. Oh and S. Son. An algorithm for real-time fault-tolerant scheduling in multiprocessor systems. In *Fourth Euromicro Workshop on Real-time Systems*, 1992.

[30] 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.

[31] Krithi Ramamritham. Real-time databases. *International journal of Distributed and Parallel Databases*, 1(2):199–226, 1993.

[32] Michael T. Reddy. *Securities Operations: A Guide to Operations and Information Systems in the Securities Industry*. New York Institute of Finance, 1990.

[33] David L. Scott. *Wall Street Words*. Houghton Mifflin, 1988.

[34] John Stankovic and Wei Zhao. On real-time transactions. *ACM, SIGMOD Record*, 17(1):4–18, 1988.