In Proceedings of RTDB'96: The 1996 Workshop on Real-Time Databases, Newport Beach, CA, March 1996.

Value-cognizant Admission Control Strategies for Real-Time DBMS[*]

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 *recovery block*. The execution requirements of the primary task are *not* known *a priori*, whereas those of the recovery block 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 recovery block 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 (RTDBMS) 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]

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 *recovery block*. 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 recovery block are known. Through the use of appropriate admission control policies, we show that it is possible for the system to maximize its profit dynamically.

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 recovery block*. The execution requirements for the primary task are *not* known *a priori*, whereas those for the recovery block are known *a priori*. Figure 1 shows the various components in our RTDBMS.

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 recovery block is completed, in which case we say that the transaction has *safely*

[*]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.

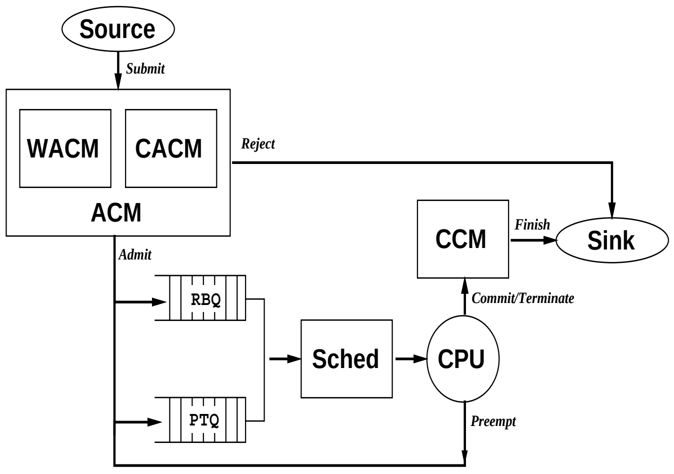

Figure 1: Major System Components

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 profit (to be gained only if the transaction 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.[2] 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 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 [25] or SCC-nS [3]), or Pessimistic Concurrency Control with Priority Abort (such as 2PL-PA [2]).

We adopt a 2-level priority scheme to schedule system resources (*e.g.,* CPU). In particular, all recovery blocks are assumed to have a higher priority than primary tasks. Thus, a primary task may be preempted (or aborted) by a recovery block, whereas a recovery block cannot be preempted by either a primary task or another recovery block under any condition.

[2]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].

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 recovery blocks finish). For example, if the percentage of the CPU bandwidth already committed to recovery blocks is high, then it may be prudent for the WACM to reject the submitted transaction. Another important function of the WACM is the scheduling of recovery blocks. A transaction is rejected if its recovery block cannot be scheduled, even if the current demand on the resources in the system is low.

2.2 Concurrency Admission Control Manager

In order to ensure that recovery blocks 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 recovery block 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), recovery blocks (which cannot be preempted) are guaranteed to finish execution without incuring 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 recovery blocks execute at a higher priority than primary tasks and, thus, are guaranteed to finish execution without incuring any blocking delays. This is not true in a multiprocessor system, where multiple recovery blocks may be executing concurrently. In such a system, the CACM ensures that only those recovery blocks that do not conflict with each other are allowed to overlap when executed.

2.3 Processor Scheduling Algorithm

There are two queues managed by the processor scheduler: the Primary Task Queue (PTQ) and the Recovery Block Queue (RBQ). 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 recovery block must wait for its start time, specified by the ACM. As indicated before, recovery blocks 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 recovery block 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) [22], which is optimal for a uniprocessor system with independent, preemptible tasks having arbitrary deadlines [10].

The RBQ is organized as a series of slots, one for each recovery block. Each slot contains the recovery block id as well as its start and end times. Slots are order according to ascending recovery block start time.

2.4 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 recovery block. 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 recovery blocks, 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 recovery blocks to conflict since the processor scheduler guarantees that recovery blocks do not overlap.[3] For primary tasks, such a request may result in blocking (if the read/write lock is not available).

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 recovery blocks. 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 recovery block of a transaction is inserted in the RBQ at the latest slot that satisfies its WCET. If no slot is big enough to fit the recovery block, then the transaction is rejected, otherwise it is admitted.

Latest-Fit (LF) Using this technique, the recovery block of a transaction is inserted in the RBQ at the latest slot. If the slot is not large enough, then the recovery blocks preceeding that slot are rescheduled to start at earlier times so as to "make room" for the new recovery block. If this rescheduling is not possible—because it leads to a recovery block 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 recovery block—and, if necessary, the ensuing rescheduling of other recovery blocks—is conditional on whether or not the percentage of CPU time allotted to recovery blocks[4] is below a preset margin or threshold. If recovery blocks 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 recovery blocks 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 recovery blocks, 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 recovery blocks 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 recovery blocks 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 either OCC-BC or 2PL-PA. 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 recovery block 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.[5] The recovery block of each transaction follows a normal distribution with a mean of 10 ms and standard deviation of 5 ms.[6] Transaction deadlines were related to the *serial execution time* through a *slack factor*, such that *(deadline time - arrival time)* = `SlackFactor` × *serial execution time.*

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

[3]This condition is true in any uniprocessor system where recovery blocks cannot be preempted.

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

[5]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.

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

Parameter	Meaning	Value
`ArrivalRate`	Transaction arrival rate	5 - 100 TPS
`DBsize`	Database size in pages	1,000
`Xsize`	Number of reads per transaction	16
`UpdateProb`	Update Probability	0.25
`RbCompTime`	Mean Recovery Block Time	10 ms
`RbStdDev`	St.Dev. of Recovery Block Time	5 ms
`SlackFactor`	Slack factor	2
`TaskSchd`	Task scheduling protocol	`EDF`
`RbSchd`	Rb scheduling protocol	FF, LF, LMF
`Thrsh`	Rb computation threshold	0.125
`CCntrl`	Concurrency Control protocol	OCC-BC

Table 1: Baseline Workload Parameters

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 recovery blocks 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 two additional simulations under the LMF policy. In the first, we changed the concurrency control protocol to 2PL-PA. In the second, we set the update probability to zero, thus simulating the performance of LMF in the absence of data conflicts (*i.e.* when all transactions are "read-only"). 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 [12], 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. Figure 4 shows the percentage of submitted transactions that were successfully committed by LMF for these threshold values and arrival rates.

Figure 4 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 4 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 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).

Figure 5 shows the performance gains achievable through admission control for various arrival rates (system load). For each arrival rate we perform two simulations of the system under the baseline parameters. The first utilizes LF whereas the second utilizes LAF (*i.e.* LMF at optimal thresholds as determined from figure 4). Figure 5 is a plot of the percentage increase in transaction commits achieved when LAF is used relative to when LF is used. As expected, the advantage of using LAF is much more pronounced when the system is overloaded.[7]

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 6 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 6 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 restric-

[7]It can be shown that the plot in figure 5 could be fitted to a hyperbola, suggesting a power-law relationship between the performance improvement achieved through LAF and the system load.

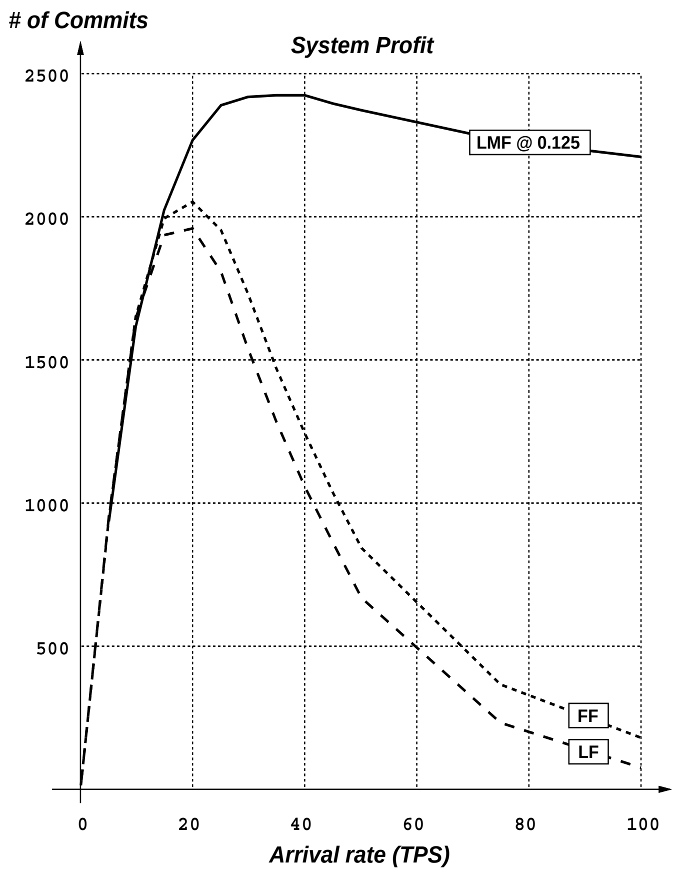

Figure 2: Performance of FF, LF, and LMF

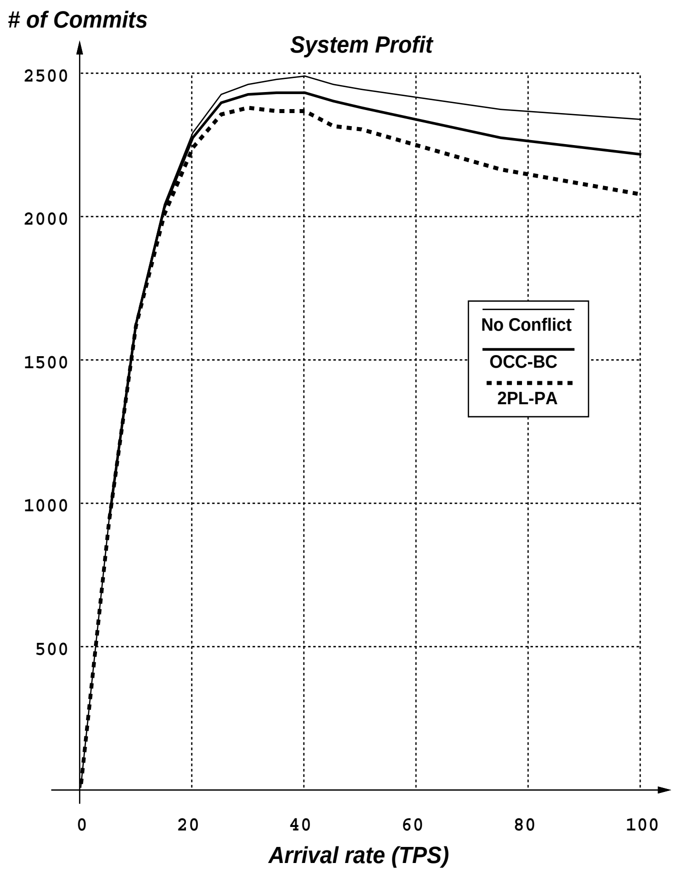

Figure 3: LMF under various concurrency conditions

tive 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.

5 Related Work

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

Liu *et al.* [21, 20, 23] 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 recovery block must execute to completion but only in the event that the primary task incurs a timing failure. Given our goal of maximizing the profit, our priority is to execute primary tasks rather than recovery blocks.

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 [19, 9] and processor failures in [26, 27, 18]. Our notion of a recovery block is indeed similar to that of an alternative; execution of a recovery block provides less attractive quality of service in comparison to the execution of the primary task. The similarities end here, however. The alternatives in Liestman and Campbell are not subject to timing failures, i.e. they have soft deadlines, whereas recovery blocks have hard deadlines. Moreover, in Chetto and Chetto, the alternatives are periodic in nature, unlike recovery blocks which are not.

Most previous RTDBMS 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 desir-

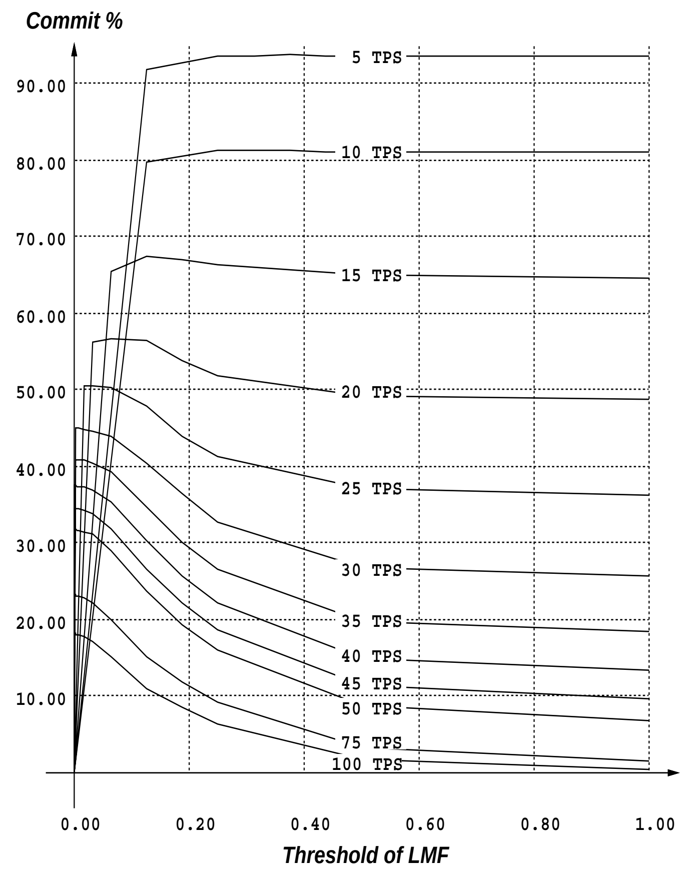

Figure 4: Effect of threshold setting on LMF performance

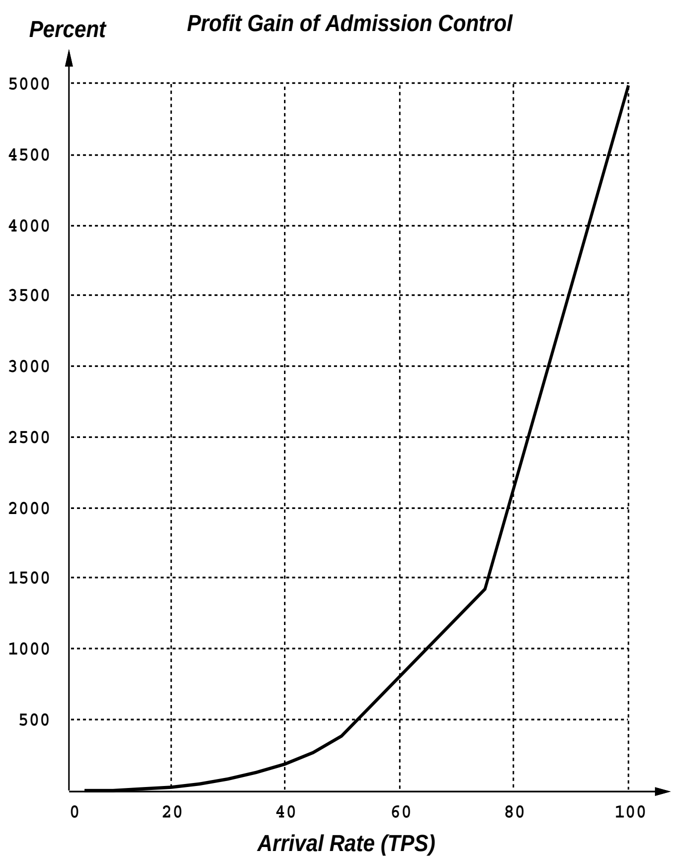

Figure 5: Percentage Profit Gain for LAF

able. 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.

Haritsa *et al.* [13] incorporate a feedback mechanism into an Adaptive Earliest Deadline (AED) scheduling strategy for transactions in a firm deadline environment. Goyal *et al.* [11] 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 RT-DBMS. 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.

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 [15, 24] have been utilized in both general real-time systems [5, 7] as well as in RTDMBS [1, 14, 29]. In [5, 7], 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 is accepted into the system is *conditionally* guaranteed[8] 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*.

This notion of "cost consciousness" is similar to that investigated by Chakravarthy, Hong, and Johnson in [8], where a Cost Conscious Approach with Average Load Factor (CCA-ALF) is proposed and evaluated. CCA-ALF is a

[8]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.

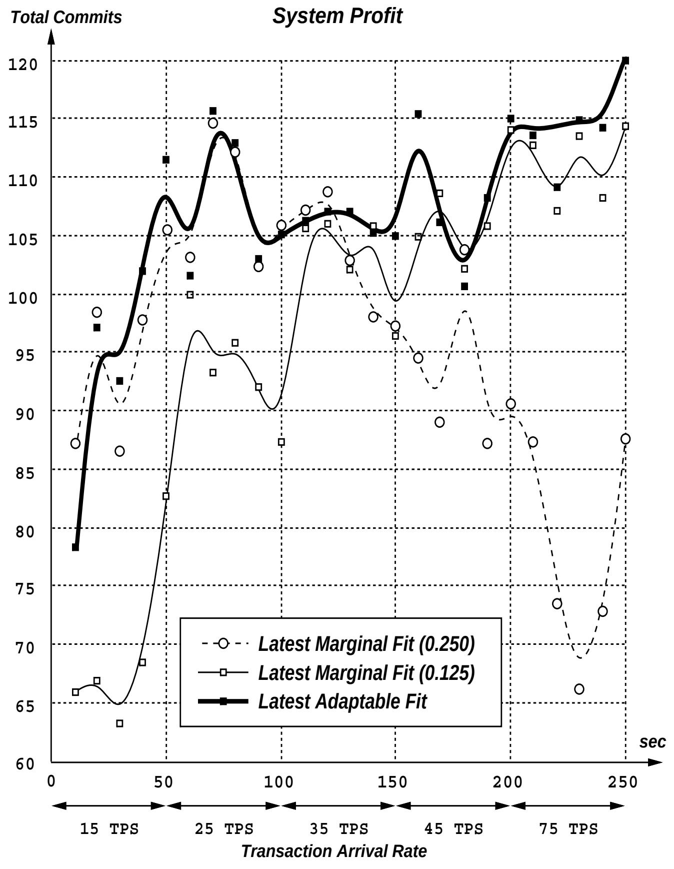

Figure 6: Dynamic Performance of LMF and LAF

best-effort scheduling strategy (*i.e.* no guarantees are given) that takes into account the dynamic aspects of transaction execution (*e.g.,* system load) in addition to its static aspects (*e.g.,* soft/firm deadlines) when making scheduling decisions. Huang *et al.* [14] use transaction values to schedule system resources (*e.g.,* CPU) and in conflict resolution protocols in a soft real-time environment. Bestavros and Braoudakis [4] also employs value functions in a soft real-time system to determine whether it is more advantageous (*i.e.* adds more value to the system) to commit a transaction or to delay that commitment for a period of time.

Two recent PhD theses have proposed novel transaction processing frameworks that allow RTDBS to apportion their resources in a value-cognizant fashion. In [17, 16], Kim establishes a RTDBS model which includes both hard and soft real-time transactions, maintains temporal and logical consistency of data [28], 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 [6, 4], 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 [3] were extended to work under this framework and were shown—in detailed simulation studies—to yield superior performance. The notion of value functions is a generalization of the earlier work of Biyabani *et al.* [5], Huang *et al.* [14], and Chakravarthy *et al.* [8].

6 Conclusion and Future Work

In this paper, we proposed a new paradigm for the execution of transactions in a RTDBMS. 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 recovery block. 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 recovery block 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 incured 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 *recovery block 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.[9] Conversely, the CACM could use information about the read/write sets of primary tasks to determine whether or not to accept a particular recovery block.

Our current work involves extending our transaction model to include transactions with different values. New admission control techniques which take into account a transaction's value (*i.e.* relative importance with respect to the transaction set) are being designed. When a transaction is submitted to the system, the profit to be gained by admitting this transaction is compared against the profit to be lost by previously admitted transaction. If the *profit-gain* is greater than the *profit-loss*, we admit the new transaction; otherwise, we reject it. In addition, CPU scheduling of primary tasks by value will also be studied.

[9] In particular, the read/write sets of recovery blocks could be used by an SCC-nS [3] algorithm to determine when shadow transactions are to be created.

References

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

[2] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

[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] 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.

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

[16] 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.

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

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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

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

[27] 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.

[28] Krithi Ramamritham. Real-time databases. *International journal of Distributed and Parallel Databases*, 1(2), 1993.

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