An Admission Control Paradigm for 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 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.

Keywords: Admission control; real-time databases; concurrency control; scheduling; and resource management.

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

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 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.

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

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 RTDBS.

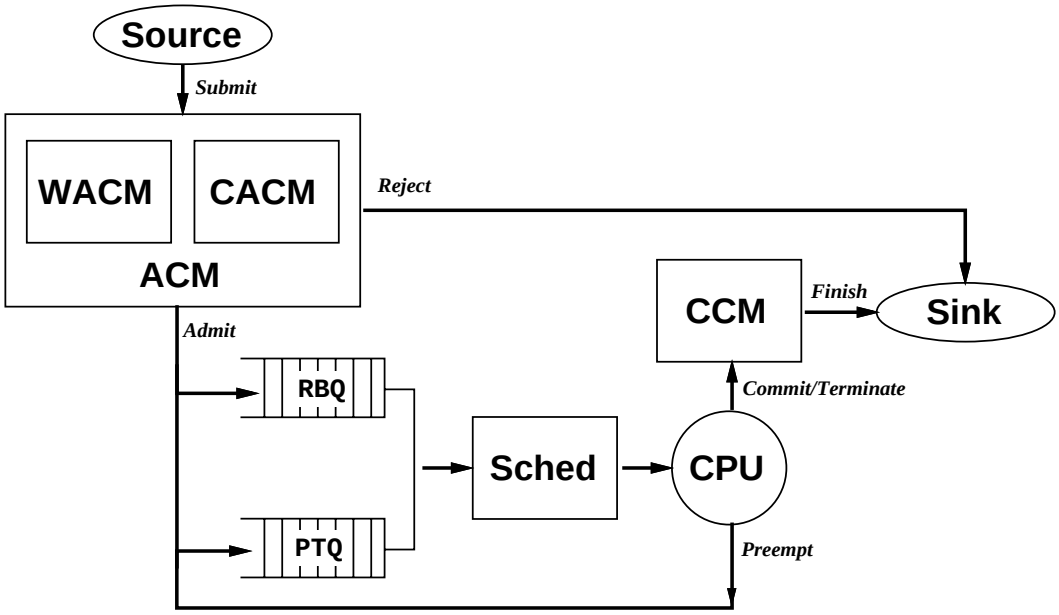

Figure 1: Major System Components

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 recovery block 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 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.

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 Algorithm with forward validation (such as OCC-BC [Mena82] or SCC-nS [Best94]) is used to ensure serializability. OCC techniques are better suited for systems with controllable utilization [Hari90], which is the case in a system with admission control like ours.[3]

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 by a recovery block, whereas a recovery block cannot be preempted.

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]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 [Best95].

[3]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).

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 OCC algorithm with forward validation, recovery blocks (which cannot be preempted) are guaranteed to finish execution without incuring any restart 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) [Liu73], which is optimal for a uniprocessor system with independent, preemptible tasks having arbitrary deadlines [Dert74].

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 recovery block, 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 transactions are restarted whereas with SCC-nS, we roll-back the transaction to a point preceding the conflicting action. 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).

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

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 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 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 recovery block completions (*i.e.* primary task abortions).

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 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 100 ms CPU time.[5] The recovery block of each transaction follows a normal distribution with a mean of 25 ms and standard deviation of 12.5 ms.[6] Transaction deadlines were related to the *serial execution time* through a *slack factor*, such that *(deadline time - arrival time)* = `SlackFactor` $\times$ *serial execution time*.

The transaction inter-arrival rate, which is drawn from an exponential distribution, is varied from 2 transactions per second up to 20 transactions per second in increments of 2, which represents a light-to-medium loaded system. We used two additional arrival rates of 30 and 40 transactions per second to experiment with a very heavy loaded system. Each simulation was run four times, each time with a different seed, for 500,000 ms. The results depicted are the average over the four runs.

[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 5 page accesses. This equivalence holds because recovery blocks cannot be interrupted once started and since a forward validation optimistic concurrency control algorithm is used.

Figure 2 shows the absolute number of successfully committed transactions, which is a measure of the *value-added* to (or *profit* of) the system, when the FF, LF, LMF (with a 0.167 threshold) policies are in use. Under light-to-medium loads (arrival rates < 8 TPS), the performance of FF and that of LF are identical. Under medium-to-heavy (arrival rates > 8 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 slightly worse than 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.

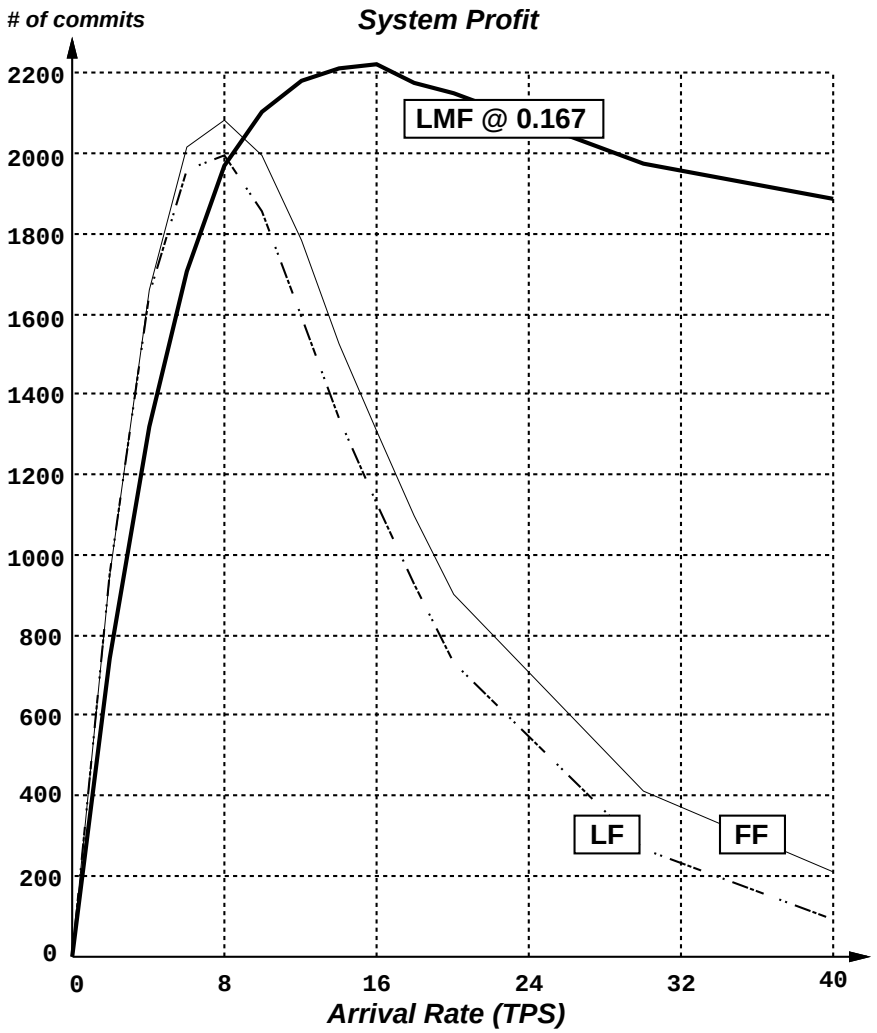

Figure 2: Performance of FF, LF, and LMF

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. Specifically, we used threshold values of 0.05, 0.1, 0.125, 0.167, 0.25, 0.5, 0.75, and 1.0 (*i.e.* accept all, which reduces LMF to LF), and we used arrival rates of 4, 6, 8, 10, 12, 16, 20, 30 TPS. Figure 3 shows the percentage of submitted transactions that was successfully committed by LMF for these threshold values and arrival rates.

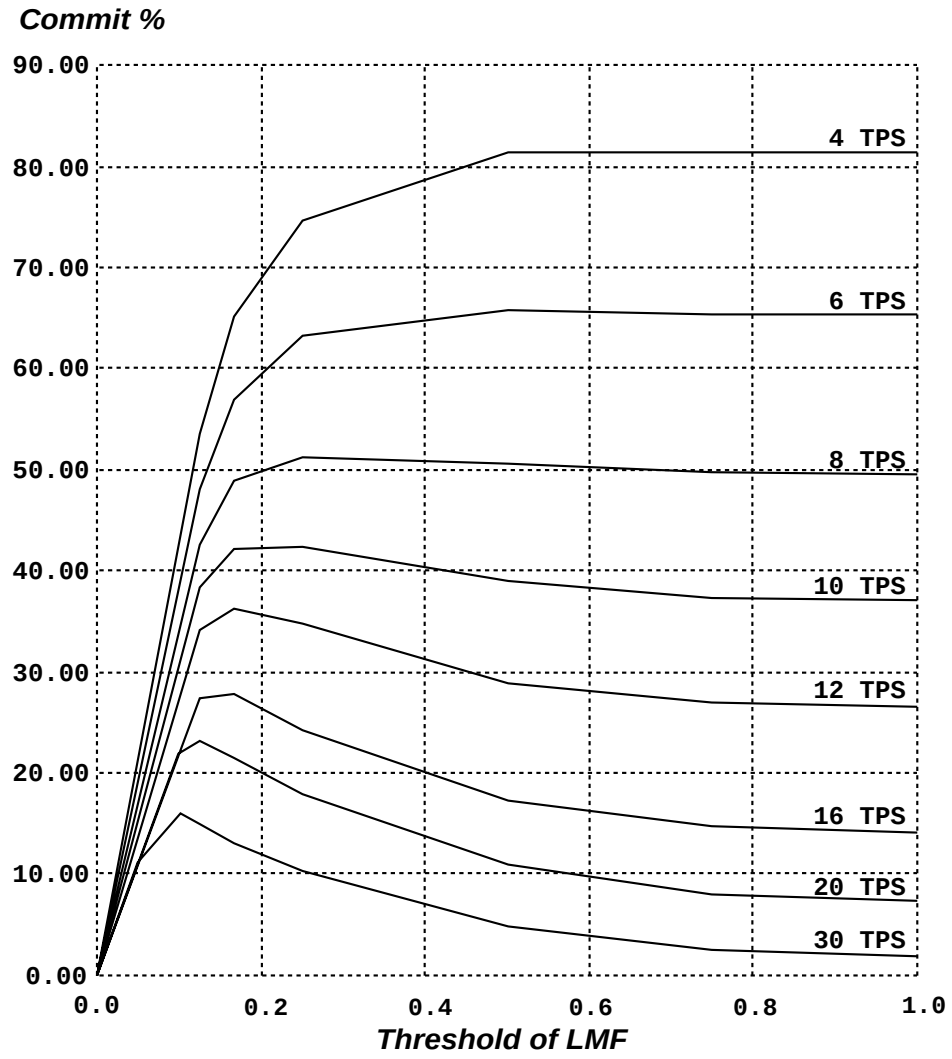

Figure 3: Effect of threshold setting on LMF performance

Figure 3 shows that for lightly-loaded systems (arrival rates less than 6 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 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.,* 12-40 TPS) than it is under more moderate loads (*e.g.,* 6-10 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 (while keeping all other parameters unchanged). Our simulation consisted of 5 consecutive epochs, each running for 125 sec, for a total of 625 seconds. The arrival rate of transactions in these epochs was set to 6, 10, 14, 18, 30 TPS, 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 25 sec, 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 625 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 indistinguishable.

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,

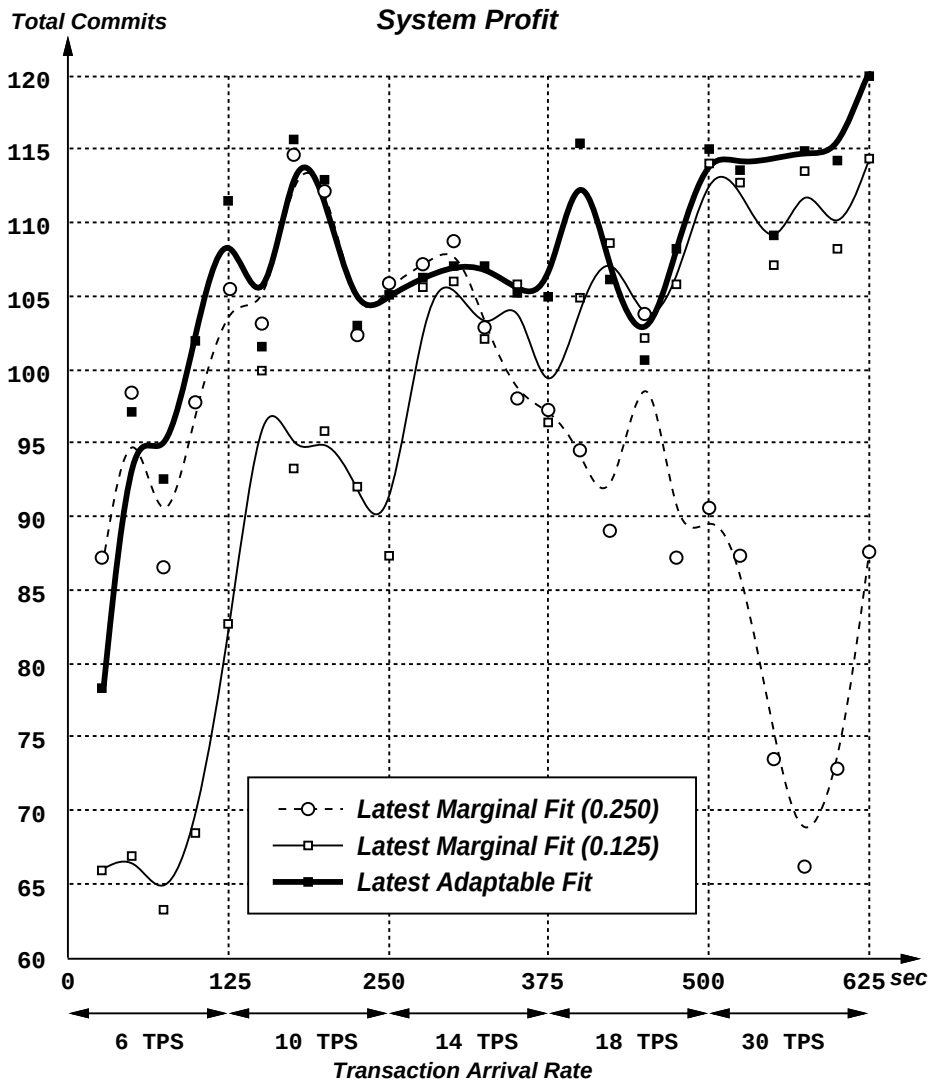

Figure 4: Dynamic Performance of LMF and LAF

average slack factor, average read/write mix, and average recovery block 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 recovery block computation time—are likely to change and

that LAF has to adapt to these changes dynamically.[7] The first step involves the evaluation of the optimum threshold value for each node in a 3-dimensional mesh. Figure 1 shows the different values we considered along each dimension, as well as other parameters used in our simulation.

Parameter	Meaning	Value
`ArrivalRate`	Transaction arrival rate	1 - 16 TPS
`RbCompTime`	Mean Recovery Block Time	25, 50, 125, 250 ms
`RbStdDev`	St.Dev. of Recovery Block Time	0.5 `RbCompTime`
`SlackFactor`	Slack factor	1.5, 2, 3, 4
`TaskSchd`	Task scheduling protocol	`EDF`
`RbSchd`	Rb scheduling protocol	LMF
`Thrsh`	Rb computation threshold	0 - 1

Table 1: Workload Model Parameters

We ran 4 simulations for each setting of `ArrivalRate`[8], `RbCompTime`, and `SlackFactor`—a total of 192 combinations, or 768 simulations. This process was repeated for a number of threshold values in order to compute the *optimal* value per setting.

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 result from the above experiment. The workload for each experiment was constructed by fixing the value along one dimension to emulate a different workload as described in figure 5.

Each experiment consisted of 20 consecutive epochs of 25 sec each for a total running time of 500 sec. At the beginning of each epoch, the values of the parameters were set according to the specifications above. For example, under Workload 3, at the beginning of each epoch, the `SlackFactor` and `RbCompTime` were chosen at random and used for transactions generated during that epoch, while the `ArrivalRate` remained at 16 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 recovery block scheduling techniques, for each workload is shown in figure 6.

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

[8] The `ArrivalRate` varied from 1 TPS up to 8 TPS in increments of 1 with four additional rates of 10, 12, 14, and 16 TPS.

Workload 0 Random: `ArrivalRate`, `RbCompTime`, and `SlackFactor` change throughout the experiment.

Workload 1 Lax Deadlines: `ArrivalRate` and `RbCompTime` change throughout the experiment, while `SlackFactor` is set to 4.

Workload 2 Tight Deadlines: `ArrivalRate` and `RbCompTime` change throughout the experiment, while `SlackFactor` is set to 1.5.

Workload 3 High Arrival Rate: `SlackFactor` and `RbCompTime` change throughout the experiment, while `ArrivalRate` is set to 16 TPS.

Workload 4 Low Arrival Rate: `SlackFactor` and `RbCompTime` change throughout the experiment, while `ArrivalRate` is set to 2 TPS.

Workload 5 Long Recovery Blocks: `ArrivalRate` and `SlackFactor` change throughout the experiment, while `RbCompTime` is set to 250 ms.

Workload 6 Short Recovery Block: `ArrivalRate` and `SlackFactor` change throughout the experiment, while `RbCompTime` is set to 25 ms.

Figure 5: Workload Descriptions

Figure 6 shows that LAF achieves the most profit when all 3 parameters are allowed to change (workload 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 [Jens85, Lock86] have been utilized in both general real-time systems [Biya88, Butt95] as well as in RTDMBS [Abbo88, Huan89, Stan88]. In [Biya88, Butt95], the value of a task is evaluated during the admission control process. The

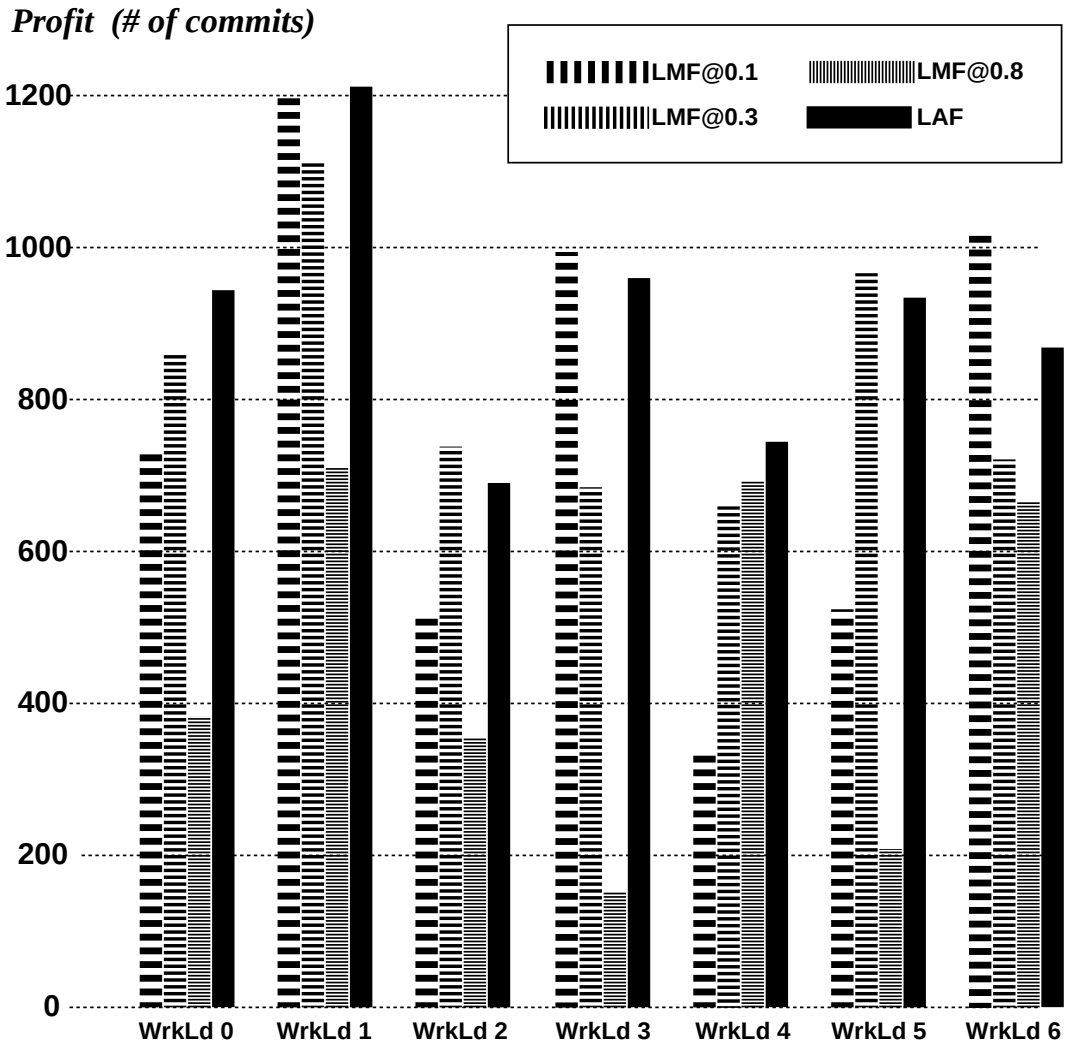

Figure 6: Profits achievable by LMF and LAF in a dynamic environment

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[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*.

This notion of "cost consciousness" is similar to that investigated by Chakravarthy, Hong, and Johnson in [Chak94], where a Cost Conscious Approach with Average Load Factor (CCA-

[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.

ALF) is proposed and evaluated. CCA-ALF is a 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.* [Huan89] use transaction values to schedule system resources (*e.g.,* CPU) and in conflict resolution protocols in a soft real-time environment. Bestavros and Braoudakis [Best95] 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 [Kim95, Kim93], Kim establishes a RTDBS model which includes both hard and soft real-time transactions, maintains temporal and logical consistency of data [Rama93], 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 [Brao94, Best95], 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 [Best94] 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.* [Biya88], Huang *et al.* [Huan89], and Chakravarthy *et al.* [Chak94].

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.

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.

Haritsa *et al.* [Hari91] incorporate a feedback mechanism into an Adaptive Earliest Deadline (AED) scheduling strategy for transactions in a firm deadline environment. Goyal *et al.* [Goya95] 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.

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 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.[10] 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.

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.

[10] In particular, the read/write sets of recovery blocks could be used by an SCC-nS [Best94] algorithm to determine when shadow transactions are to be created.

References

[Abbo88] Robert Abbott and Hector Garcia-Molina. "Scheduling real-time transactions." *ACM, SIGMOD Record*, 17(1):71–81, 1988.

[Best94] Azer Bestavros and Spyridon Braoudakis. "Timeliness via speculation for real-time databases." In *Proceedings of RTSS'94: The 14th IEEE Real-Time System Symposium*, San Juan, Puerto Rico, December 1994.

[Best95] 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.

[Biya88] 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.

[Brao94] Spyridon Braoudakis. *Concurrency Control Protocols for Real-Time Databases.* PhD thesis, Computer Science Department, Boston University, Boston, MA 02215, expected June 1994.

[Butt95] 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.

[Chak94] 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.

[Dert74] M. L. Dertouzos. "Control robotics: The procedural control of physical processes." In *Proceedings IFIP Congress*, pages 807–813, 1974.

[Goya95] 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.

[Hari90] Jayant R. Haritsa, Michael J. Carey, and Miron Linvy. "On being optimistic about real-time constraints." In *Proceedings of the 1990 ACM PODS Symposium*, April 1990.

[Hari91] Jayant R. Haritsa, Miron Linvy, and Michael J. Carey. "Earliest deadline scheduling for real-time database systems." In *Proceedings of the 12th Real-Time Systems Symposium*, December 1991.

[Huan89] 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.

[Jens85] 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.

[Kim93] 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.

[Kim95] Young-Kuk Kim. *Predictability and Consistency in Real-Time Transaction Processing.* PhD thesis, Department of Computer Science, University of Virginia, May 1995.

[Liu73] 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.

[Lock86] C. Locke. *Best Effort Decision Making for Real-Time Scheduling.* PhD thesis, Carnegie-Mellon University, Department of Computer Science, May 1986.

[Mena82] D. Menasce and T. Nakanishi. "Optimistic versus pessimistic concurrency control mechanisms in database management systems." *Information Systems*, 7(1), 1982.

[Rama93] Krithi Ramamritham. "Real-time databases." *International journal of Distributed and Parallel Databases*, 1(2), 1993.

[Stan88] John Stankovic and Wei Zhao. "On real-time transactions." *ACM, SIGMOD Record*, 17(1):4–18, 1988.