Int. J. Advanced Networking and Applications 1456
Volume:04 Issue:01 Pages: 1456-1461 *(2012) ISSN : 0975-0290.*

Admission Control Prototype for Real-Time Databases

Rahul Kumar Mishra[1],Dr. Udai Shanker [2]
[1.]Department of Computer Applications, IFTM Campus, Moradabad, UP, India,
[2].Department of Computer Sc. & Engineering. M. M. Engineering College, Gorakhpur-273 010,UP
Email: rahulmishraiftm@gmail.com

ABSTRACT

We suggest and measure an admission control prototype for RTDBS, in which a transaction is presented to the system as a pair of procedures: a primary task, and a recovery block. The performance necessities of the main 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 applied to determine whether to admit or reject that transaction. Once admitted, a transaction is assured to finish executing before its deadline. A transaction is considered to have finished executing if exactly one of two things occurs: Either its primary task is completed (successful commitment), or its recovery block is finished (safe termination). Committed transactions bring a profit to the system, whereas a terminated transaction brings no profit. The objective of the admission control and scheduling communications protocol (e.g., concurrency control, I/O scheduling, memory management) employed in the system is to maximize system profit. We depict 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.

Date of Submission: May 06, 2012 Date of Acceptance: June 08, 2012

1 INTRODUCTION

The main dispute demanded in scheduling transactions in a Real-Time Databases Management System (RTDBS) is that the resources demanded to accomplish 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 special Worst Case Execution Time (WCET) suits impossible-and the non-deterministic delays associated with the on-the-y acquisition of such resources pose the real challenge of integrating scheduling and concurrency control techniques.
Current real-time concurrency control mechanisms decide the above dispute by decompressing the deadline semantics (thus suggesting best-effort mechanisms for concurrency control in the presence of soft and firm, but not hard deadlines), or by limiting the set of satisfactory 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 suggest and measure, through simulation experiments, a paradigm that preserves the hard deadline meaning without accepting complete a priori knowledge of transaction execution necessities. Our paradigm allows the system to reject a transaction that is submitted for performance, or else admit it and thus assure that one of two outcomes will occur by the transaction's deadline: either the transaction will successfully commit through the performance of a primary task, or the transaction will safely finish through the execution of a recovery block. The system presumes no a priori knowledge of the execution necessities of the primary task, but assumes that the WCET and read/write sets of the recovery block are known. Through the use of reserve admission control policies, we show that it is potential for the system to maximize its profit dynamically.

We begin in section 2 with an overview of our transaction working model and the different elements therein. Next, in section 3 we describe the various

Volume:04 Issue:01 Pages: 1457-1461 *(2012) ISSN : 0975-0290.*

Admission Control Strategies to be used in our simulations. Next, in section 4 we demonstrate and discuss our simulation baseline model and results. In section 5, we brush up previous research work and highlight our contributions. We resolve in section 6 with a summary and a description of succeeding research directions.

2 Organization Model

Each transaction presented to the system consists of two elements: a primary task, and a recovery block. The execution necessities for the primary task are not known a priori, whereas those for the recovery block are experienced a priori. Figure 1 shows the various components in our RTDBS.

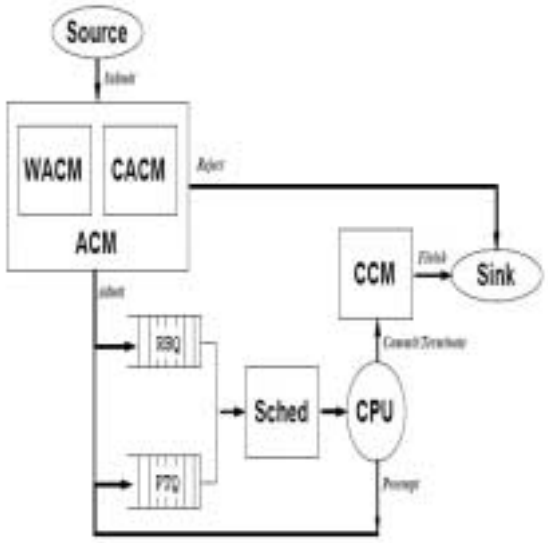

Figure 1: Major System Components

When a transaction is presented to the system, an Admission Control Mechanism (ACM) is applied to decide whether to admit or refuse that transaction. Once admitted, a transaction is ensured to finish executing before its deadline. A transaction is conceived to have finished executing if exactly one of two things occurs: Either its primary task is finished, in which case we say that the transaction has successfully devoted, 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 utilized in the system is to maximize benefit.

When presented to the system, each transaction is linked with a deadline and a profit (to be gained only if the transaction is devoted by its deadline.) In this paper we conceive only hard deadlines and thus assume that no transaction will finish (i.e. successfully commit or safely terminate) past its deadline.[2] Also, we presume that all proceedings bring in equal profit when devoted on time.

The ACM constitutes of two major elements: a Concurrency Admission Control Manager (CACM) and a Workload Admission Control Manager (WACM). The CACM is responsible ensuring that admitted transactions do not overburden the system by demanding a level of concurrency that is not sustainable. The WACM is responsible for promising that accepted deals do not overload the system by demanding computing (e.g., CPU time) that are not sustainable.

In this paper we presume that an Optimistic Concurrency Control Algorithm with forward establishment (such as OCC-BC [Mena82] or SCC-nS [Best94]) is used to assure serializability. OCC techniques are better fitted for systems with governable utilization [Hari90], which is the case in a system with admission control like ours.[3]

We adopt a 2-level priority strategy to schedule system resources (e.g., CPU). In particular, all recovery blocks are accepted 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 pre-empted.

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 trans-action submitted for execution is based upon a feedback mechanism that assumes into consideration the current demand on the resources in the system. This conclusion is prompted by the overall goal for maximizing profit by maximizing the number of productive dedications (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 significant function of the WACM is the programming of recovery blocks. A transaction is refused if its recovery block cannot be scheduled, even if the current demand on the resources in the system is low.

Volume:04 Issue:01 Pages: 1458-1461 *(2012) ISSN : 0975-0290.*

2.2 Concurrency Admission Assure Manager

In order to assure that recovery blocks can execute unhampered (and thus complete within their WCETs) the CACM must ensure 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 accepted 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 product retrieval blocks may be accomplishing concurrently. In such a system, the CACM assures that only those recovery blocks that do not conflict with each other are allowed to overlap when executed.

2.3 Processor Programming Algorithm

There are two queues handled by the processor scheduler: the Primary Task Queue (PTQ) and the Recovery Block Queue (RBQ). Each accepted transaction adds one entry in each of these queues. A primary task is prepare to execute as soon as it is enqueued in the PTQ, whereas a recovery block must wait for its start time, defined by the ACM. As indicated before, recovery blocks accomplish 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 prepare to execute.

Since all tasks in the PTQ are ready to execute, a scheduling algorithm must be used to apportion the CPU time between these tasks. We use the Earliest Deadline First algorithm (EDF) [Liu73], which is optimal for a uniprocessor system with independent, pre-empt tasks having arbitrary deadlines [Dert74].

2.4 Concurrency Assure Manager

As each transaction completes its execution, either by the dedication of its primary task or by the safe outcome 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 communications protocol in effect. In the case of OCC-BC, self-contradictory transactions are restarted whereas with SCC-nS, we roll-back the deal to a point preceding the conflicting action. All transactions, whether filled out or refused, are removed from the system and sent to the sink which gives statistical information used to evaluate the system operation.

3 Optimizing Profit through ACM

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

As suggested before, the main epitome 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 proficiency, 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 ~t the recovery block, then the transaction is rejected, otherwise it is admitted.

Latest-Fit (LF) Using this proficiency, 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 proficiency is identical to Latest-Fit, except that the scheduling of a recovery block-and, if necessary, the ensuing rescheduling of other recovery blocks[4]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 proficiency 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 go forward 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

Volume:04 Issue:01 Pages: 1459-1461 *(2012) ISSN : 0975-0290.*

dynamically using a table lookup procedure. The table is computed o~-line (using simulations) to determine the optimum quiescent value for the threshold under a host of other parameters.

4 Performance Evaluations

We have carried out the above ACM policies for a uniprocessor system using OCC-BC. In this section we show the value of admission control by equating the performance achievable through FF, LF, LMF, and LAF. Since we presume 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 main 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 points were related 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 attracted 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.Figure 2 demonstrates the absolute number of successfully devoted deals, 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 anticipated due to LF's tighter packing of recovery blocks via scheduling, 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 stave off thrashing, thus continuing the system's profit in check with its capacity.

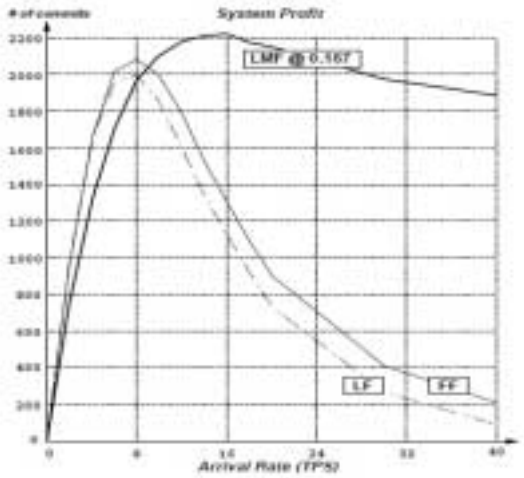

Figure 2: Performance of FF, LF, and LMF

The value of the threshold to be used in LMF is key to its operation. As we explained before, the optimal value for this threshold depends upon 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 deal 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 presented transactions that was successfully committed by LMF for these threshold values and arrival rates.

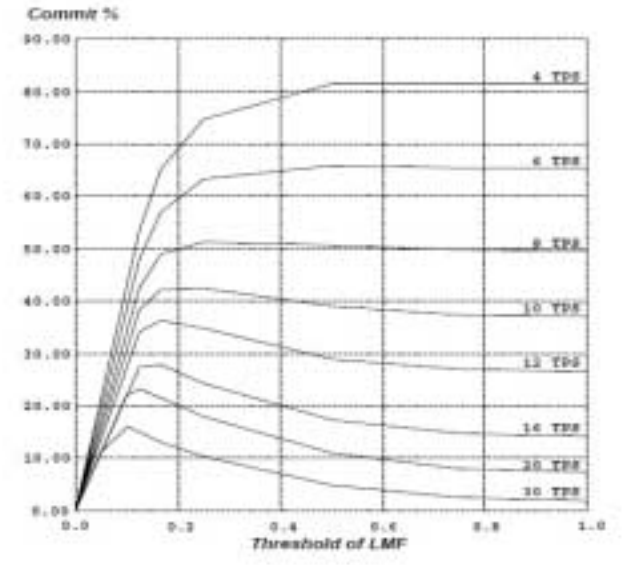

Figure 3: Effect of threshold setting on LMF performance

Volume:04 Issue:01 Pages: 1460-1461 *(2012) ISSN : 0975-0290.*

Figure 3 demonstrates 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 deals should be accepted, making the performance of LMF identical to that of LF. For moderately-loaded and heavily-loaded systems, Figure 3 suggests that an optimum threshold exists for each arrival rate. Setting the threshold to that optimal value yields the highest percentage of productive commitments, and thus gives the highest potential 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 measure the effect of dynamically altering the threshold in LAF, we ran a computer simulation of the system, in which we varied the arrival rate (while keeping all other parameters unchanged). Our simulation consisted of 5 straight 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 demonstrates the performance of LAF versus 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 productive 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 qualify the public presentation 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 anticipated, 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 reasonably loaded, the performance of all three techniques is indistinguishable.

In the above experiment, only the arrival rate of transactions exchanges from one epoch to the other, and as a result, LAF was allowed for 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 well 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 recovery block length, among others), then expending o~-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 reserve dimensions for this optimization process is an interesting research problem.

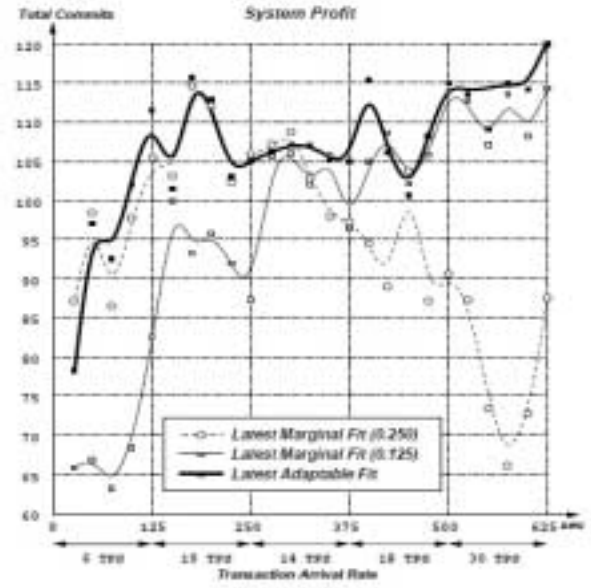

Figure 4: Dynamic Performance of LMF and LAF

5 Conclusions 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.

Volume:04 Issue:01 Pages: 1461-1461 *(2012) ISSN : 0975-0290.*

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.

References

[Abbo88] Robert Abbott and Hector Garcia-Molina. “Scheduling real-time transactions." A CM, 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.