DYNAMIC CPU SCHEDULING WITH IMPRECISE KNOWLEDGE OF COMPUTATION-TIME

Saud A. Aldarmi, Alan Burns

Department of Computer Science
University of York, U.K.
(saud,burns@cs.york.ac.uk)

Abstract: The majority of the studies conducted in scheduling real-time transactions mostly concentrate on concurrency control protocols, while overlooking the CPU as being the primary resource. Consequently, there are various techniques for scheduling the CPU in conventional time-critical systems; meanwhile, there does not seem to be any technique that is adequately designed for scheduling such a resource in Real-Time Database (RTDB) systems. In this paper, we construct an efficient CPU scheduling scheme that minimizes the preemption rate in order to reduce the frequency by which synchronization protocols must be invoked, along with their inherited performance degradation. In addition, we also introduce a new timing model upon which the newly introduced scheduler is incorporated in order to utilize the system's imprecise knowledge of computation time estimates.

Keywords: CPU Scheduling, lowering-preemption, timeliness-functions, and imprecise computation estimates.

1. INTRODUCTION

An *RTDB* system's scheduler is a mechanism for ordering the execution of the outstanding transactions on the system's resources according to some predefined criteria. Thus, an *RTDB* scheduler consists of two separate modules; i.e., a *CPU* scheduler and a Concurrency-Control mechanism. While there are several views on the roll of priority cognizance in resolving contention and conflicts over the database, priority ordering is necessary for resolving contention over the *CPU*.

CPU scheduling algorithms used in current real-time *task* models assume a priori *accurate* knowledge of the tasks' resource requirement; i.e., worst-case computation-time. Such an assumption instigated various scheduling techniques for the task model. Meanwhile, it is nearly impossible to *accurately* determine the *actual* computation-time of a transaction, since the runtime behavior of a transaction is dynamic in the sense that it depends upon the values it finds in the database *at run-time*. Therefore, a worst-case computation-time may be very *pessimistic* in an environment such as an *RTDB*. Thus, *CPU* scheduling schemes that relay heavily on *accurate* computation-time may not perform as well when operating with *imprecise* estimates.

In this paper we construct a *CPU* scheduling scheme that does not rely on accurate knowledge of computation time, while simultaneously minimizing preemption and its undesirable consequences, and is thus, most suitable to operate under an *RTDB* environment.

This paper focuses on scheduling "Soft-Deadline" transactions. That is, the transactions that complete their execution before their deadlines are considered successful and impart a full value, *V*, to the system. Whereas transactions that complete after their deadlines are known to be *tardy* and only impart a portion of their net value that is proportional to their *tardiness*; i.e., lateness. The rest of this paper is based on the following assumptions. All transactions are *aperiodic*, and scheduling is *preemptive*. The system's scheduler learns of the following set of attributes *only* at (*A*), the transaction's arrival time.

- *I* – an *importance* level, reflecting the transaction's *significance* to the overall functionality of the system. Note that $V = I$ at arrival time.
- D^a – *absolute* deadline.
- D^r – *relative* deadline, such that $D^r = D^a - A$. Note that whenever we use *D* without a superscript, we are referring to D^r.

The remainder of this paper is organized as follows: Section (2) presents previous work and the motivation of this study. Section (3) introduces a new timing model upon which our newly introduced scheduling scheme is incorporated. Section (4) introduces a new scheduling scheme called *Aging Timeliness Cumulative Computation (ATCC)*. Section (5) presents our simulation model along with its assumptions and the performance metrics used in this paper. Section (6) presents a comparative study exhibiting the effectiveness of the newly introduced scheme. Section (7) concludes this research.

2. MOTIVATION AND PREVIOUS WORK

In the task model, *Value-Density* (*VD*) (Jensen *et al.*, 1985, and Lock 1986), i.e., V/C, is a value-based scheduling scheme that is known to outperform many other scheduling algorithms under overload situations. The manner, in which *VD* scales the task's value by its execution time, causes scheduling priority to be derived on the task level. On the other hand, *Dynamic Value Density (DVD)* (Aldarmi and Burns, 1999a), i.e., $V/\overline{C}^{\,2}$ (where $\overline{C}$ represents the remaining computation time), enhances the performance of *VD* by assigning an extremely low priority to newly arriving tasks. Such small priority increases in correspondence with the amount of time that a task executes. Consequently, the scheduling priority is not derived *statically* on the task level; rather, it is derived *dynamically* for the individual execution unit(s). The consequences of such behavior are that preemption is significantly lowered, and execution resumption after preemption is also increased. *DVD* was shown in (Aldarmi and Burns, 1999-a) to better utilize the *CPU* as being the system's primary resource.

(Huang *et al.*, 1989) introduced a technique known as "*Scheduling by Criticalness and Deadline First*" *(CDF)*, that was intended to operate in an *RTDB* environment. The technique is stated in the following function:

$$Priority\ (P) = \frac{Relative\ Deadline}{Criticalness} \equiv \frac{D}{I} \qquad (2.1)$$

Function (2.1) uses the transactions' *significance* to the system and *relative deadline*, both of which are not concerned with the computation time of the corresponding transaction(s). However, function (2.1) relies on two *static* attributes; and thus, the derived priorities are *static* and *fixed* from the instant that they are derived until the transactions' termination, where termination means commit or abort. Therefore, *CDF* is subject to similar weaknesses and limitations as *VD*. Since transactions dynamically undergo several changes as they migrate through the system, monitoring and utilizing the transactions' dynamic attributes can provide the means of a better *CPU* scheduling scheme; hence, better overall performance. In this paper we wish to alter *CDF* and construct a scheduling technique that is capable of *mimicking* the effective behavior of *DVD*, in addition to minimizing the preemption rate in order to suite *RTDB* environments.

3. IMPRECISE-TIMELINESS-FUNCTIONS

Before we introduce our scheduling scheme, we wish in this section to construct the timing-model; i.e., *Imprecise-Timeliness-Function(s) (ITF)*, upon which our scheduler will be incorporated.

If a transaction does not start executing before $t + \overline{C} = D^a$ (for t being the current time), then the transaction is guaranteed to become tardy. Thus, if a *waiting* transaction is subject to becoming tardy due to having an *infeasible* deadline, then it should not receive a relatively high scheduling priority in order not to delay the scheduling of other transactions that have a better chance of meeting their deadlines. *Timeliness-functions* (Aldarmi and Burns, 1998) address such an issue, and therefore, they start reducing the system's internal representation of value at $(D^a - \overline{C})$ in order to project, at the scheduling instant, the final value to be gained at the finishing moment.

Abstractly define Δ to be the diminishing-rate by which a transaction's value is computed when the transaction becomes tardy, and $\psi_{(t)}$ being a transaction's tardiness at time t. For V and $\overline{T}$ respectively being a transaction's *value* and *timeliness*, timeliness-functions introduced in propose shifting V to the left by $\overline{C}$ to produce $\overline{T}$, both of which are illustrated in figure (1), and respectively computed in functions (3.1) and (3.2).

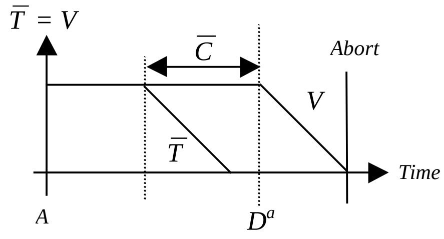

Figure (1)

$$\Psi_{(t)} = max\ (0, t - D^a)$$
$$V_{(t)} = I - [\Delta \times \Psi_{(t)}] \qquad (3.1)$$
$$\overline{T}_{(t)} = I - [\Delta \times \Psi_{(t+\overline{C})}] \qquad (3.2)$$

The proposed technique works well if the transaction's computation-time estimate is accurate. However, if the actual computation-time is much less than the worst-case estimate, then a transaction might be erroneously aborted; i.e., *aborted prematurely*, meanwhile, the transaction could have finished with a significant value. Hence, since transactions in *RTDB* systems do not generally have accurate computation-time estimates, function (3.2) may be more applicable for conventional real-time systems (the task model) than *RTDB* systems (the transaction model). In the next subsection, we address the different types of transactions that might exist in an *RTDB* system.

3.1. *Transactions' Characteristics*

In general, a transaction's computation-time either depends on the nature and functionality of the transaction, and/or on the environment and operating conditions. Several classes of transactions have been identified in previous studies; e.g., (Kim and Son, 1996), each of which has different characteristics and runtime requirements. If a transaction's body were closely inspected, it would either contain a single path with a fixed functionality; e.g., *canned transactions*, which require a specific runtime, and thus, a *precise* computation-time estimate, or it might contain multiple paths. A compiler can certainly approximate the maximum/minimum-required computation-time, which can be used to generate an expected error bound; i.e., a degree of imprecision, within the transaction's worst-case estimate. Consequently, we classify transactions into two categories, those with *precise* estimates, and those with *imprecise* estimates.

Case (1) – Precise Computation-Time

For all transactions with *precise* computation-time, it is reasonable to start diminishing a transaction's timeliness at $(D^a - \overline{C})$, since $\overline{C}$ accurately reflects the remaining computation-time.

Case (2) – Imprecise Computation-Time

For a transaction with an imprecise computation-time estimate, timeliness must not use $\overline{C}$ as stated above, but rather use an estimate of $\overline{C}$ that corresponds to the degree of imprecision. In order to do so, define $\tilde{C}$ to be the total amount of time that a transaction has computed so far. In addition, define α be the degree of imprecision in the transaction's computation-time estimate; i.e., error bound, which implies that the minimum computation-time for the corresponding transaction is "$C \times (1-\alpha)$", which we refer to as C_{min}. That is, for a transaction with a computation-time estimate that is imprecise by *25%*, then $\alpha = 0.25$. For example, for a transaction with $C = 100$, and $\alpha = 0.25$, then the *minimum* computation-time ≡ $C_{min} = 75$. Hence, for a transaction with an accurate estimate $\alpha = 0$. Consequently, a transaction's timeliness can be computed at "$D^a - C_{min} - \tilde{C}$". Hence, a transaction will not be aborted unless its *minimum remaining* required computation-time is infeasible. Thus, timeliness may be computed as given in function (3.3). For further analysis of function (3.3) the reader is referred to (Aldarmi and Burns, 1999b).

$$\overline{T}_{(t)} = I - [\Delta \times \Psi_{(t + C_{min} - \tilde{C})}] \tag{3.3}$$

4. AGING-TIMELINESS-CUMULATIVE-COMPUTATION (ATCC)

Imprecise-Timeliness-Functions *ITF* described in the previous section; i.e., function (3.3), can be utilized to account for many of the transaction's attributes and thus, *ITF* allow the scheduler's decisions to be made *deadline-cognizant*. Therefore, *CDF* (described in section 2) would be a better scheme if it utilized the transaction's time-variant $\overline{T}$ instead of *I*. In addition, since we wish preference to be given in an *increasing* order of the priorities; i.e., opposite to the order of (Huang *et al.*, 1989), we alter function (2.1) above as shown in function (4.1) below.

$$P = \frac{\overline{T}}{D} \tag{4.1}$$

However, the scheduling priority is still derived on the transaction level. Recall that *DVD* achieves its performance significance (superiority) over the traditional *VD* by increasing the scheduling priority by an amount that corresponds to *V* for every execution unit. Therefore, function (4.1) must also increase by (at least) a corresponding amount if it is to mimic the effectiveness of *DVD*. Adding $(\overline{T} \times \tilde{C})$ to the numerator of function (4.1) as shown in function (4.2) allows the scheduler to weight each execution unit according to the corresponding transaction's timeliness. Therefore, the scheduler not only becomes aware of various valued execution units, but also is able to credit each transaction for all the units it executed with respect to the transaction's own timeliness.

$$P = \frac{\overline{T} + \overline{T} \times \tilde{C}}{D} = \frac{\overline{T} \times [1 + \tilde{C}]}{D} \tag{4.2}$$

Furthermore, transactions in an *RTDB* environment have to block for synchronization, I/O, commit protocols, buffer management, etc. Therefore, we should account for such delays in the scheduling decision. Thus, define ω to be the total amount of time that a transaction has been waiting in the system since its arrival, excluding any $\tilde{C}$. That is, $\omega_{(t)} = t - A - \tilde{C}_{(t)}$, which is a *dynamic* attribute corresponding to *aging*. In order to account for a database inherited delays, we need to add such an attribute to function (4.2). In addition, we must also treat it in the same manner that we treated $\tilde{C}$; i.e., multiply it by $\overline{T}$, in order to distinguish between a waiting highly significant transaction vs. a waiting less-significant transaction. However, if the priority of a waiting transaction increases in a manner similar to that of an executing transaction, then the waiting transaction might interfere with an executing transaction, and *thrashing* would be expected due to the corresponding high preemption rate. Therefore, we need to scale down ω first, then multiply it by $\overline{T}$.

Note that if a transaction is short then it is not going to require a substantial amount of system's resources and it should be given scheduling preference. We argue that in order to increase the system's throughput, then we must speedup the execution of all short transactions. Based on such an observation we propose scaling ω down by *C*. However, since *C* might involve a certain degree of imprecision, then such imprecision must be utilized when considering *C*. Thus, define $\Omega_{(t)}$ to be a weighted aging factor at time *t* such that:

$$\Omega_{(t)} = \frac{\omega_{(t)}}{C_{min}}$$

With Ω defined as such, function (4.2) is mapped into function (4.3) below, which may be called *Aging Timeliness Cumulative Computation* (*ATCC*), owing its name to all the parameters involved in it. Note that function (4.3) utilizes the underlying timing-model by using $\overline{T}$ in weighting $\tilde{C}$ and Ω.

$$ATCC = \frac{\overline{T} \times [1 + \tilde{C} + \Omega]}{D} \qquad (4.3)$$

Function (4.3) clearly shows the feasibility of utilizing information about runtime behavior; e.g., *computation-time*, in priority assignment schemes for the transaction model. The function involves only simple arithmetic, and utilizes the transactions' timeliness (thus value with respect to deadline), in addition to the amount of computation-time that has already been consumed and remains to be acquired. Furthermore, the function also accounts for the amount of time that transactions wait in the system (whether for I/O or concurrency control). Many details have been omitted due to space limitation, all of which can be found in (Aldarmi and Burns, 1999b).

5. SIMULATION MODEL

The simulator we use in this paper is based on *CSIM*; C-based process oriented language (Schwetman, 1994), and has the following parameters and assumptions.

- The levels of importance are randomly assigned to transactions from a *uniform* distribution from (1.0, 5.0), which may be viewed as {low, mid-low, mid, mid-high, high}.
- The worst-case computation-time, *C*, is randomly assigned to transactions from a *uniform* distribution from (100.0, 200.0). However, the simulated load has the following characteristics: 1/3 of the transactions have *precise* computation-time; and thus, α = 0. The 2nd 1/3 of the transactions have *imprecise* computation-time such that $0.5C \leq \hat{C} \leq C$, for $\hat{C}$ being the actual computation time; thus, α = 0.5. The 3rd 1/3 of the transactions have *imprecise* computation-time such that $0.1C \leq \hat{C} \leq C$; thus, α = 0.9.
- The time it takes to process each data granule is 10 time units, and the number of data granules requested by any transaction corresponds to the transaction's actual length; e.g., a transaction that is 100 units long requires 10 locks. However, since we are assuming memory resident database, there is no I/O blocking delays; rather, there is only concurrency control blocking delay.
- All required data granules are requested from the database manager, which employs *Priority-Abort (PA)* (Abbott and Garcia-Molina, 1988) for conflict resolution.
- When a transaction is submitted to the system, it is assigned its *deadline* (D^a), such that:

  $$D^a = A + C + uniform\ (2.0,5.0) \times C$$

  Note that the *C* corresponds to the worst-case computation time estimate and not the actual computation time.
- All transactions have *soft-deadline*, and when a transaction's (*value*) $V \leq I/100$, the transaction is assumed to have lost its validity and therefore is aborted. For $\hat{I}$ and $\hat{C}$, respectively, being the maximum importance level and the maximum computation-time required by any transaction within the entire system, the *diminishing speed* (Δ) of a transaction's value = $\hat{I} / \hat{C}$, and thus, the values of all transactions diminish at the same speed.
- When a transaction is preempted an artificial delay = $\hat{C} / 100$ is introduced in order to simulate the overhead of context switching.
- The simulated load is 80% to 200% controlled by an *exponential* distribution for the transactions' arrival, which in turn is controlled by the *average actual* computation-time ($\hat{C}_a$) of all transactions divided by the desired load (σ). Thus, the submission rate $\equiv \lambda = \hat{C}_a / \sigma$. Note that an exponential distribution allows for *bursty* arrivals, which may result in some transactions missing their deadlines even under normal load.
- The simulation corresponds to the average behavior of 5,000 transactions at each simulated load.

The performance of the simulated techniques is measured according to the following metrics; the interested reader may refer to (Aldarmi and Burns, 1999-b) for other performance metrics.

- *CPU Wastage % = (total time spent on aborted tasks + total time spent on preemption)* $\times$ *100* $\div$ *total time spent on all transactions.*
- *Value-Sum % = total value collected* $\times$ *100* $\div$ *total value of all transactions submitted to the system.*
- *Preemption = total number of preemption* $\times$ *100* $\div$ *total number of transactions submitted to the system.*
- *Restart % = total number of restarted transactions* $\times$ *100* $\div$ *total number of transactions submitted to the system.*
- *Wasted Locks % = total number of lost locks* $\times$ *100* $\div$ *total number of locks acquired by all transactions.*

6. COMPARATIVE STUDY

In this section, we contrast the performance of function (4.3) against several *CPU* scheduling schemes; namely:

- *Earliest Deadline First (EDF)* (Liu and Laylan, 1973) – the transaction with the closest deadline receives the highest priority,
- *Scheduling by Criticalness and Deadline First (CDF)*; i.e., function (2.1),
- *Value-Density (VD),*
- *Dynamic-Value-Density (DVD).* Note that we use $(C - \tilde{C})$ in computing the remaining computation time; i.e., $\bar{C}$. However, since the actual computation time may differ from the worst-case estimate in *RTDB* environments as well as our simulator, *VD* and *DVD* might make the wrong decision(s). This scenario exhibits the limitation of *VD* as well as *DVD.*

Figures (2 to 6) show the overall performance of the scheduler. In general, the figures seem to classify the five scheduling schemes being simulated into three groups: *EDF* being in its own group, while *CDF* and *VD* seem to be comparable as in one group, and finally, *DVD* and *ATCC* being somewhat comparable as in a separate group.

Figure (2) shows the total amount of *CPU* time that is wasted on preemption and aborted, partially processed, transactions. The figure shows that *EDF* wastes as much as 38% under a load of 200%. Meanwhile, *VD* and *CDF* waste as much as 10-15% under a load of 80-200%, respectively. On the other hand, while *DVD* waste up to 3% of the *CPU* capacity, *ATCC* wastes as little as 0.19% (practically 0%). The efficiency of *ATCC* in utilizing the *CPU* over the other schemes is also reflected in the rest of the system's performance as depicted in figures 3 to 6.

Figure (3) shows the total amount of value that the scheduler is able to collect from the completed set of transactions. The figure clearly shows that *ATCC* not only is capable of sustaining higher performance under overload situations, but also outperforms the remaining simulated schemes under normal operating conditions as well! Thus, an efficient *CPU* scheduling can be achieved without accurate knowledge of computation time estimates.

Figures (4 and 5), respectively, show the preemption rate and the percentage of transactions that are restarted due to conflict resolution. Notice the strong correlation between the two figures! If a transaction is preempted while it holds some resources, then the problem of *priority-inversion* (Abbott and Garcia-Molina, 1988) becomes an issue, which causes aborting and possibly restarting several conflicting transactions. *Aborting* and *redoing* overhead could accumulate as data-contention rises, which could increase further as preemption rises. Regardless of the concurrency control mechanism being employed (pessimistic or optimistic), a higher preemption rate could significantly contribute to higher amounts of conflict within the system, which translates into performance degradation. Thus, lowering the preemption rate is very important in an *RTDB* system due to the fact that even if data-contention is high between the current set of transactions within the scheduler's queue, due to the nature of the transactions, it is naturally reduced if preemption is reduced. Tailoring the underlying *CPU* scheduling scheme in a manner that minimizes the preemption rate can significantly reduce concurrency control degradation. *ATCC* is a preemptive scheme that attempts to derive and assign the scheduling priorities in a manner that actually construct a sequence of transactions that minimizes preemption while achieving a relatively high level(s) of performance as shown in figures 2 to 6.

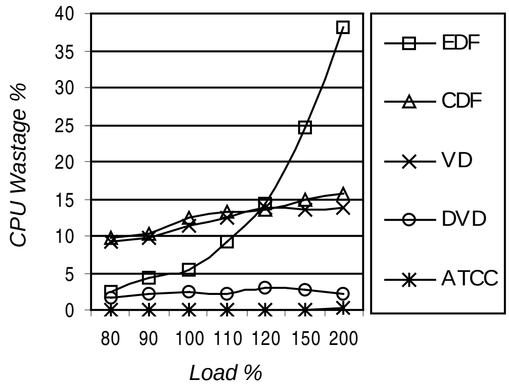

Figure (2)

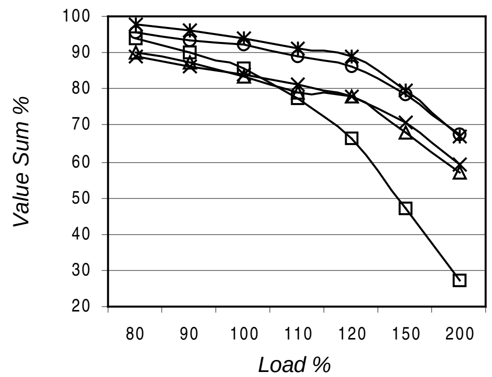

Figure (3)

Finally, figure (6) shows the percentage of locks that is lost (wasted) due to both, restarts as well as aborting partially processed transactions. In contrast to the wasted *CPU* time, figure (6) can be viewed as the wasted database-manger's time. Thus, figures (5 and 6) show that *ATCC* assigns the scheduling priorities in a manner that not only minimizes restarts, but also attempts to prevent wasting any acquired locks and thereby not wasting any of the database-manager's time. Thus, many existing studies conducted in scheduling real-time transactions attempt to minimize restarts and wasted locks by devising new concurrency control protocols. However, figures 5 and 6 show that the answer to reducing restarts and wasted-locks does not highly depend on concurrency control protocols, but mainly on the *CPU* scheduler and the manner in which it derives and assigns the scheduling priorities!

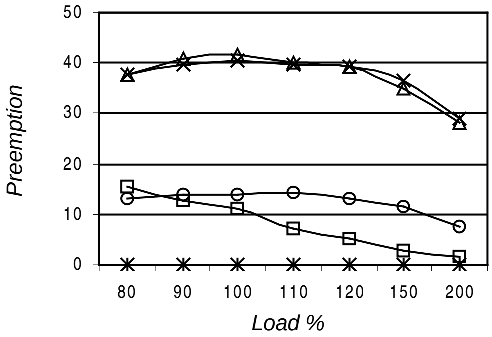

Figure (4)

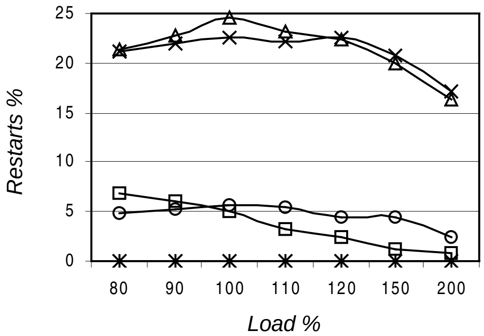

Figure (5)

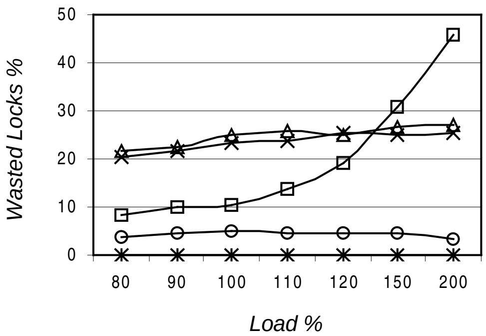

Figure (6)

7. CONCLUSION

In this paper, we constructed a new *CPU* scheduler (*ATCC*) along with the necessary underlying timing model (*ITF*), both of which render an *RTDB* system's scheduler the ability to accommodate *precise* as well as *imprecise* computation-time estimates. The introduced schemes were shown to substantially reduce the amount of wasted system's resources; and thus, redirect such resources towards sustaining a higher level of performance. Hence, providing an *RTDB* system with a time cognitive behavior complying with the requirements and specifications of time-critical systems. This paper clearly shows that transaction *CPU* scheduling in an *RTDB* system is not hindered by the absence of accurate knowledge of the execution time. In addition, it showed that it is not the concurrency control module, but rather, it is the *CPU* scheduler that is mainly responsible for wasting the system's resources. Hence, with the techniques introduced in this paper, concurrency control is no longer a bottleneck in the performance of *RTDB* systems.

8. REFERENCES

R. Abbott, and H. Garcia-Molina (1988). Scheduling Real-time Transactions: A Performance Evaluation. In: *Proceedings of the 14th International Conference on Very Large DataBases*, Los Angeles - California.

S. Aldarmi and A. Burns (1998). Time-Cognizant Value Functions for Scheduling Real-Time Systems. *Technical Report YCS-306*, Department of Computer Science. University of York.

S. Aldarmi and A. Burns (1999a). Dynamic Value-Density for Scheduling Real-Time Systems. *Technical Report YCS-310*, Department of Computer Science – University of York, U.K. (To appear in *the 11th Euromicro Conference on Real-Time Systems* – York, U.K., 1999).

S. Aldarmi and A. Burns (1999b). Dynamic CPU Scheduling with Imprecise Knowledge of Computation-Time. *Technical Report YCS-314*, Department of Computer Science – University of York, U.K.

J. Huang, J. Stankovic, D. Towsley, and K. Ramamritham (1989). Experimental Evaluation of Real-time Transaction Processing. In: *Proceedings of the 10th Real-time Systems Symposium*, pp. 144-153.

E. D. Jensen, C. D. Locke, H. Tokuda (1985). A Time-Driven Scheduling Model for Real-time Operating Systems. In: *Proceedings of IEEE Real-time Systems Symposium*.

Y. Kim and S. H. Son (1996). Supporting Predictability in Real-Time Database Systems. *IEEE Real-Time Technology and Application Symposium (RTAS' 96)*, Boston, MA.

C. L. Liu, and J. W. Layland (1973). Scheduling Algorithms for Multiprogramming in Hard-Real Time Environments. *Journal of the ACM*, Vol. 20, No. 1.

C. D. Locke (1986). Best-effort Decision Making for Real-time Scheduling. Ph.D. thesis, Computer Science Department, Carnegie Mellon University.

H. Schwetman (1994). CSIM Reference Manual.