Earliest Deadline Scheduling for Real-Time Database Systems

Jayant R. Haritsa   Miron Livny   Michael J. Carey
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

ABSTRACT

Earlier studies have observed that in moderately-loaded real-time database systems, using an Earliest Deadline policy to schedule tasks results in the fewest missed deadlines. When the real-time system is overloaded, however, an Earliest Deadline schedule performs worse than most other policies. This is due to Earliest Deadline giving the highest priority to transactions that are close to missing their deadlines. In this paper, we present a new priority assignment algorithm called Adaptive Earliest Deadline (AED), which features a feedback control mechanism that detects overload conditions and modifies transaction priority assignments accordingly. Using a detailed simulation model, we compare the performance of AED with respect to Earliest Deadline and other fixed priority schemes. We also present and evaluate an extension of the AED algorithm called Hierarchical Earliest Deadline (HED), which is designed to handle applications that assign different values to transactions and where the goal is to maximize the total value of the in-time transactions.

1. INTRODUCTION

A Real-Time Database System (RTDBS) is a transaction processing system that is designed to handle transactions with completion deadlines. Several previous RTDBS studies (e.g. [Abbo88, Abbo89]) have addressed the issue of scheduling transactions with the objective of minimizing the number of late transactions. A common observation of these studies has been that assigning priorities to transactions according to an Earliest Deadline [Liu73] policy minimizes the number of late transactions in systems operating under low or moderate levels of resource and data contention. This is due to Earliest Deadline giving the highest priority to transactions that have the least remaining time in which to complete. These studies have also observed, however, that the performance of Earliest Deadline steeply degrades in an overloaded system. This is because, under heavy loading, transactions gain high priority only when they are close to their deadlines. Gaining high priority at this late stage may not leave sufficient time for transactions to complete before their deadlines. Under heavy loads, then, a fundamental weakness of the Earliest Deadline priority policy is that it assigns the highest priority to transactions that are close to missing their deadlines, thus delaying other transactions that might still be able to meet their deadlines.

This research was partially supported by the National Science Foundation under grant IRI-8657323.

From the above discussion, the following question naturally arises: Can a priority assignment policy be developed based on the Earliest Deadline approach that stabilizes its overload performance without sacrificing its light-load virtues? A well-constructed scheme based on simple real-time principles was presented in [Jens85] for realizing this objective. In order to use this scheme, which was developed in the context of task scheduling in real-time operating systems, a-priori knowledge of task processing requirements is necessary. Unfortunately, knowledge about transaction resource and data requirements is usually unavailable in database applications [Stan88]. Therefore, the [Jens85] solution cannot be used and methods applicable to *transaction scheduling* in real-time database systems have to be developed.

In this paper, we present a new priority assignment algorithm called Adaptive Earliest Deadline (AED) that stabilizes the overload performance of Earliest Deadline in an RTDBS environment. The AED algorithm uses a feedback control mechanism to achieve this objective and does not require knowledge of transaction characteristics. Using a detailed simulation model, we compare the performance of the AED algorithm with that of Earliest Deadline and other fixed priority mappings.

There are real-time database applications that may assign *different values* to transactions, where the value of a transaction reflects the return the application expects to receive if the transaction commits *before* its deadline [Huan89]. For such applications, the goal of the RTDBS is to maximize the value realized by the in-time transactions. Minimizing the number of late transactions is a secondary concern in this environment. A fundamental problem here is how to establish a priority ordering among transactions that are distinguished by both values and deadlines [Biya88, Hari91a]. In particular, it is not clear what *tradeoff* should be established between transaction values and deadlines in generating the priority ordering.

We present here an extension of the AED algorithm, called Hierarchical Earliest Deadline (HED), for integrating the value and deadline characteristics of transactions. The HED algorithm adaptively varies the tradeoff between value and deadline, based on transaction deadline miss levels, to maximize the value realized by the system. We compare the performance of the HED algorithm to that of mappings which establish fixed tradeoffs between value and deadline.

Our simulation model implements a database system architecture that incorporates a *priority mapper* unit. The priority mapper assigns a priority to each transaction on its arrival; these priorities are used by the system schedulers in resolving transaction contention for hardware resources and data objects. This priority architecture is modular since it separates *priority generation* from *priority usage*.

In this paper, we restrict our attention to real-time database systems that execute *firm deadline* applications. For such applications, a transaction that misses its deadline loses its value. This means that transactions are *discarded* when the RTDBS detects that the transaction cannot complete before its deadline. In our model, a transaction is discarded only when it *actually* misses its deadline since the system has no advance knowledge of transaction service requirements. Examples of applications with firm deadlines are given in [Abbo88].

2. Priority Mappings

In order to resolve contention for hardware resources and data, an RTDBS has to establish a priority ordering among the transactions. Apart from the previously discussed Earliest Deadline policy, there are a few other mappings described in the literature that fit our operating constraints. These mappings are described first in this section. Subsequently, our new priority mapping, Adaptive Earliest Deadline, is presented. In the following discussion, A_T, D_T, and P_T are used to denote the arrival time, deadline, and priority of transaction T, respectively. The priority assignments of all the mappings are such that smaller P_T values reflect higher system priority. The details of the mappings are presented below.

2.1. Earliest Deadline (ED)

The Earliest Deadline mapping assigns higher priority to transactions with earlier deadlines, and the transaction priority assignment is $P_T = D_T$.

2.2. Latest Deadline (LD)

The Latest Deadline mapping is the opposite of the Earliest Deadline mapping. It gives higher priority to transactions with later deadlines, and the transaction priority assignment is $P_T = 1/D_T$. We expect that, for many real-time applications, newly-submitted transactions will tend to have later deadlines than transactions already executing in the system. Therefore, it seems plausible that the Latest Deadline mapping would rectify the overload drawback of Earliest Deadline by giving transactions high priority early on in their execution.

2.3. Random Priority (RP)

The Random Priority mapping randomly assigns priorities to transactions without taking any of their characteristics into account. The transaction priority assignment is P_T = Random $(0, \infty)$. The performance obtained with this mapping reflects the impact of the mere existence of *some* fixed ordering among the transactions.

2.4. No Priority (NP)

The No Priority mapping gives all transactions the *same* priority, and the transaction priority assignment is $P_T = 0$. This effectively means that scheduling at each resource is done in order of arrival to the resource (i.e., local FCFS). The performance obtained under this mapping should be interpreted as the performance that would be observed if the real-time database system were replaced by a conventional DBMS (and the feature of discarding late transactions was retained).

2.5. Adaptive Earliest Deadline (AED)

The Adaptive Earliest Deadline priority assignment algorithm modifies the classical Earliest Deadline mapping based on the following observation: Given a set of tasks with deadlines that can all *somehow* be met, an Earliest Deadline priority ordering will *also* meet all (or most of) the deadlines [Jens85]. The implication of this observation is that in order to maximize the number of in-time transactions, an Earliest Deadline schedule should be used among the largest set of transactions that can all be completed by their deadlines. The flaw of the pure ED mapping is that it uses this schedule among *all* transactions in the system, even when the system is overloaded. The AED algorithm tries to address this flaw by using a feedback control process to estimate the number of transactions that are sustainable under an ED schedule.

2.5.1. Group Assignment

In the AED algorithm, transactions executing in the system are collectively divided into two groups, HIT and MISS, as shown in Figure 1. Each transaction, upon arrival, is assigned to

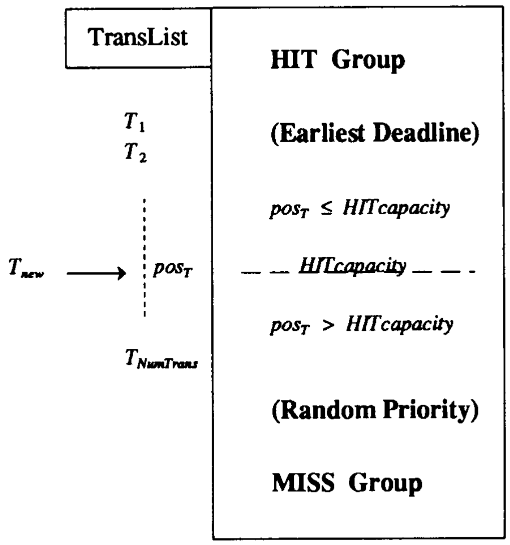

Figure 1: AED Priority Mapping

one of the groups. The assignment is done in the following manner: The newly-arrived transaction is assigned a randomly-chosen unique integer key, I_T. The transaction is then inserted into a *key-ordered* list of the transactions currently in the system, and its position in the list, pos_T, is noted. If pos_T is less than or equal to **HITcapacity**, which is a dynamic control variable of the AED algorithm, the new transaction is assigned to the *HIT* group; otherwise, it is assigned to the *MISS* group.

2.5.2. Priority Assignment

After a new transaction is assigned to a group, it is then assigned a priority using the following formula:

$$P_T = \begin{cases} (0,\ D_T,\ I_T) & \text{if } Group = HIT \\ (1,\ 0,\ I_T) & \text{if } Group = MISS \end{cases}$$

With this priority assignment scheme, all transactions in the *HIT* group have a higher priority than transactions in the *MISS* group. (Since the priority is a vector, priority comparisons are made in lexicographic order.) Within the *HIT* group, the transaction priority ordering is Earliest Deadline. In contrast, the priority ordering in the *MISS* group is Random Priority since the I_T's are selected randomly. The I_T component of the priority serves to break the tie for transactions in the *HIT* group that may have identical deadlines, thus ensuring a *total* priority ordering. Transactions retain their initial priority assignments for the entire duration of their residence in the system.

2.5.3. Discussion

The goal of the AED algorithm is to collect the largest set of transactions that can be completed before their deadlines in the *HIT* group. It trys to achieve this by controlling the size of the *HIT* group, using the *HITcapacity* setting as the control variable. Then, by having an Earliest Deadline priority ordering within the *HIT* group, the algorithm incorporates the observation made in [Jens85] that was discussed earlier. Transactions that cannot be accomodated in the *HIT* group are estimated to miss their deadlines and are therefore assigned to the *MISS* group. The motivation for having a Random Priority mapping in the *MISS* group is explained in Section 5.

We define the "hit ratio" of a transaction group to be the fraction of transactions in the group that meet their deadlines. Using this terminology, we would ideally like to have a (steady-state) hit ratio of 1.0 in the *HIT* group and a hit ratio of 0.0 in the *MISS* group, since this combination of hit ratios ensures that *all* the "doable" transactions, and *only* these transactions, are in the *HIT* group. Achieving this goal would require absolute accuracy in predicting the right *HITcapacity* size; this is impossible as the RTDBS has no knowledge of transaction characteristics. From a practical standpoint, therefore, our aim is to maintain a high hit ratio in the *HIT* group and a low hit ratio in the *MISS* group. The key to achieving this lies in the *HITcapacity* computation, which is discussed next.

2.5.4. HIT Capacity Computation

A feedback process that employs system output measurements is used to set the *HITcapacity* control variable. The measurements used are **HitRatio(HIT)** and **HitRatio(ALL)**. HitRatio(HIT) is the fraction of transactions in the *HIT* group that are making their deadline, while HitRatio(ALL) is the corresponding measurement over *all* transactions in the system. Using these measurements, and denoting the number of transactions currently in the system by *NumTrans*, the *HITcapacity* is set with the following two-step computation:

```
(1) HITcapacity := HitRatio(HIT) * HITcapacity * 1.05;
(2) if HitRatio(ALL) < 0.95 then
       HITcapacity := Min (HITcapacity,
                           HitRatio(ALL) * NumTrans * 1.25 );
```

STEP 1 of the *HITcapacity* computation incorporates the feedback process in the setting of this control variable. By conditioning the new *HITcapacity* setting based on the observed hit ratio in the *HIT* group, the size of the *HIT* group is adaptively changed to achieve a 1.0 hit ratio. Our goal, however, is not just to have a HitRatio(HIT) of 1.0, but to achieve this goal with the *largest* possible transaction population in the *HIT* group. It is for this reason that STEP 1 includes a 5 percent expansion factor. This expansion factor ensures that the *HITcapacity* is steadily increased until the number of transactions in the *HIT* group is large enough to generate a HitRatio(HIT) of 0.95. At this point, the transaction population size in the *HIT* group is close to the required number, and the *HITcapacity* remains stabilized at this setting (since $0.95 * 1.05 \simeq 1.0$).

STEP 2 of the *HITcapacity* computation is necessary to take care of the following special scenario: If the system experiences a long period where HitRatio(ALL) is close to 1.0 due to the system being lightly loaded, it follows that HitRatio(HIT) will be virtually 1.0 over this extended period. In this situation, the *HITcapacity* can become very large due to the 5 percent expansion factor, that is, there is a "runaway" effect. If the transaction arrival rate now increases such that the system becomes overloaded (signaled by HitRatio(ALL) falling below 0.95), incrementally bringing the *HITcapacity* down from its artificially high value to the right level could take a considerable amount of time (with the feedback process of STEP 1). This means that the system may enter the unstable high-miss region of Earliest Deadline as every new transaction will be assigned to the *HIT* group due to the high *HITcapacity* setting. To prevent this from occurring, an upper bound on the *HITcapacity* value is used in STEP 2 to deal with the *transition* from a lightly-loaded condition to an overloaded condition. The upper bound is set to be 25 percent greater than an estimate of the "right" *HitCapacity* value, which is derived by computing the number of transactions that are *currently* making their deadlines. (The choice of 25 percent is based on our expectation that the estimate is fairly close to the "right" value.) After the *HITcapacity* is quickly brought down in this fashion to near the appropriate setting, the HitRatio(HIT) value then takes over as the "fine tuning" mechanism in determining the *HITcapacity* setting.

2.5.5. Feedback Process

The feedback process for setting the *HITcapacity* control variable has two parameters, *HITbatch* and *ALLbatch*. These parameters determine the sizes of transaction batches that are used in computing the output hit ratios. The feedback process operates in the following manner: Assume that the *priority mapper* has just set the *HITcapacity* value. The next *HITbatch* transactions that are assigned to the *HIT* section of the bucket are marked with a special label. At the RTDBS output, the completion status (in-time or late) of these specially-marked transactions is monitored. When the last of these *HITbatch* transactions exits the system, HitRatio(HIT) is measured as the fraction of these transactions that completed before their deadline. The HitRatio(ALL) is continuously measured at the output as the hit ratio of the last *ALLbatch* transactions that exited from the system. After each measurement of HitRatio(HIT), the HitRatio(HIT) value is fed back to the *priority mapper* along with the current HitRatio(ALL) value. The *priority mapper* then reevaluates the *HITcapacity* setting, after which the whole process is repeated.

3. Concurrency Control

Several different mechanisms are available for implementing concurrency control, including locking, timestamps, and optimistic concurrency control [Bern87]. While we have shown in previous studies [Hari90a, Hari90b] that optimistic algorithms can outperform locking algorithms in a firm deadline RTDBS, we will consider only the 2PL-HP prioritized locking algorithm [Abbo88] here. The reason is that the interaction of optimistic algorithms and priority policies is somewhat complicated and space limitations prevent us from presenting these complexities here. For the workloads considered in this study, however, we have observed optimistic algorithms to perform better than 2PL-HP (see [Hari91b]).

In 2PL-HP, classical two-phase locking [Eswa76], where transactions hold locks until commit time, is augmented with a *High Priority* [Abbo88] conflict resolution scheme. This scheme ensures that high priority transactions are not delayed by low priority transactions by resolving all data conflicts in favor of the transaction with the higher priority. When a transaction requests a lock on an object held by other transactions in a conflicting lock mode, if the requester's priority is higher than that of all the lock holders, the holders are restarted and the requester is granted the lock; otherwise, the requester waits for the lock holders to release the object.

4. RTDBS Performance Model

A detailed model of a real-time database system was used to study the performance of the various priority mappings. We briefly describe the model in this section (see [Hari91b] for details). In this model, the database system consists of a shared-memory multiprocessor operating on disk resident data (for simplicity, we assume that all data is accessed from disk and buffer pool considerations are therefore ignored). The database itself is modeled as a collection of pages. Transactions arrive in a Poisson stream and each transaction has an associated deadline. A transaction consists of a sequence of read and write page accesses. A read access involves a concurrency control request to get access permission, followed by a disk I/O to read the page, followed by a period of CPU usage for processing the page. Write requests are handled similarly except for their disk I/O – their disk activity is deferred until the transaction has committed. A transaction that is restarted due to data conflict follows the same data access pattern as the original transaction. If a transaction has not completed by its deadline, it is immediately aborted and discarded.

Table 1 summarizes the key parameters of the workload model. The *ArrivalRate* parameter specifies the mean rate of transaction arrivals. The number of pages accessed by a transaction varies uniformly between 0.5 and 1.5 times *TransSize*. Page requests are generated from a uniform distribution (without replacement) spanning the entire database. *WriteProb* gives the probability of a page that is read being also updated.

In our experiments, we used the following formula for deadline assignment:

$$D_T = A_T + SF_T * R_{max}$$

where D_T and A_T are the deadline and arrival time of transaction T, respectively, and R_{max} is the expected execution time of the largest possible transaction (a transaction accessing 1.5 * *TransSize* pages). SF_T is a *slack factor* that varies uniformly over the range set by the workload parameters *LSF* and *HSF*, and it determines the tightness/slackness of deadlines.

The physical resources in our model consist of multiple CPUs and multiple disks. There is a common queue for the CPUs and the service discipline is priority Preemptive-Resume. Each of the disks has its own queue and the service discipline is priority Head-Of-Line. Table 2 summarizes the key parameters of the system model. The *DatabaseSize* parameter gives the number of pages in the database, and the data pages are modeled as being uniformly distributed across all of the disks. The

Parameter	Meaning
ArrivalRate	Transaction arrival rate
TransSize	Avg. transaction size
WriteProb	Write probability/accessed page
LSF	Low Slack Factor
HSF	High Slack Factor

Table 1: Workload Parameters

Parameter	Meaning
DatabaseSize	Number of pages in database
NumCPUs	Number of processors
NumDisks	Number of disks
PageCpu	CPU time for processing a data page
PageDisk	Disk service time for a page

Table 2: System Parameters

NumCPUs and *NumDisks* parameters specify the hardware resource composition, while the *PageCPU* and *PageDisk* parameters capture CPU and disk processing times per data page.

5. Experiments and Results

In this section, we present simulation performance results for our experiments comparing the various priority mappings in a real-time database system environment. The performance metric used in this set of experiments is *Miss Percent*, which is the percentage of input transactions that the system is unable to complete before their deadline. Miss Percent values in the range of 0 to 20 percent are taken to represent system performance under "normal" loadings, while Miss Percent values in the range of 20 to 100 percent represent system performance under "heavy" loading. All the experiments evaluate the Miss Percent as a function of the transaction arrival rate. (The Miss Percent graphs show mean values with relative half-widths about the mean of less than 5% at the 90% confidence interval.)

While describing the AED algorithm in Section 2.5, we mentioned two parameters, *HITbatch* and *ALLbatch*, that are used to determine the sample size in computing transaction hit ratios. These parameters were both set to 20 in the experiments described here (we comment on this choice in Section 8).

5.1. Resource Contention (RC)

Our first experiment investigated the performance of the priority mappings when resource contention is the sole performance limiting factor. The settings of the workload parameters and system parameters for this experiment are listed in Table 3. The *WriteProb* parameter, which gives the probability that an accessed page is updated, is set to 0.0 to ensure that there is no data contention. Therefore, no concurrency control is required in this experiment since all transactions belong to the *query* (read-only) class.

For this experiment, Figures 2a and 2b show the Miss Percent results under normal load and heavy load conditions, respectively. From this set of graphs, we observe that at normal loads, the ED (Earliest Deadline) mapping misses the fewest deadlines among the fixed priority mappings. As the system load is increased, however, the performance of ED steeply degrades, and its performance actually is close to that of NP (No Priority) at heavy loads. This is because at heavy loads, where the resources become saturated, transactions under ED and NP make progress at similar *average* rates. This is explained as follows: Under NP, every transaction makes slow but steady progress from the moment of arrival in the system since all transactions have the same priority. Under ED, however, a new transaction usually has a low priority since its deadline tends to be later than those of the transactions already in the system. Therefore, transactions tend to start off at low priority and gain high priority only as their deadline draws close. This results in transactions making little progress initially, but making fast progress as their deadline approaches. The *net* progress made under ED, however, is about the same as that under NP. This was experimentally confirmed by measuring the average progress that had been made (i.e. number of steps executed) by transactions that missed their deadline; indeed, we found that once the resources are saturated, the average progress made by transactions is virtually the same for NP and ED.

Workload Parameter	Value	System Parameter	Value
TransSize	16 pages	*DatabaseSize*	1000 pages
WriteProb	0.0	*NumCPUs*	8
LSF	1.33	*NumDisks*	16
HSF	4.0	*PageCpu*	10ms
		PageDisk	20ms

Table 3: RC Parameter Settings

Turning our attention to the RP (Random Priority) mapping, we observe that it behaves poorly at normal loads since it does not take transaction time constraints into account. At heavy loads, however, it surprisingly performs significantly better than ED. The reason for this behavior is the following: Under ED, as discussed above, transactions gain priority slowly. At heavy loads, this gradual process of gaining priority causes most transactions to miss their deadlines. The RP mapping, on the other hand, due to its static random assignment of priorities, allows some transactions to have a high priority right from the time they arrive in the system. Such transactions tend to make their deadlines, and therefore there is always a certain fraction of the transactions in the system that are virtually guaranteed to make their deadlines.

Focusing next on the LD (Latest Deadline) mapping, we observe that it performs worse than all the other algorithms at normal loads. The reason is that this mapping gives the highest priority to transactions that have loose time constraints, thus tending to miss the deadlines of transactions that have tight time constraints. At heavy loads, it performs better than ED, however, since transactions with loose time constraints continue to make their deadlines as they retain high priority for a longer period of time.

Moving on to the adaptive AED mapping, we note that at normal loads it behaves identically to Earliest Deadline. As the overload region is entered, it changes its behavior to be qualitatively similar to that of RP, and in fact, performs even better than RP. Therefore, in an overall sense, it delivers the best performance. In Figure 2c, the hit ratios in the *HIT* and *MISS* groups are shown. It is clear from this figure that a hit ratio of more than 0.90 in the *HIT* group and less than 0.10 in the *MISS* group is achieved through the entire loading range. This indicates that the feedback mechanism used to divide transactions into *HIT* and *MISS* groups is effective and achieves the goal of having a high hit ratio in the *HIT* group and a low hit ratio in the *MISS* group. In Figure 2d, the average number of transactions in the *HIT* group and the average number of transactions in the whole system are plotted. From this figure, we can conclude that for the given workload, the RTDBS can successfully schedule about 60 concurrently executing transactions under an

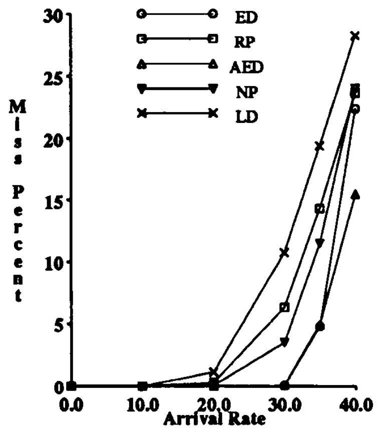

Figure 3a: RC (Normal Load)

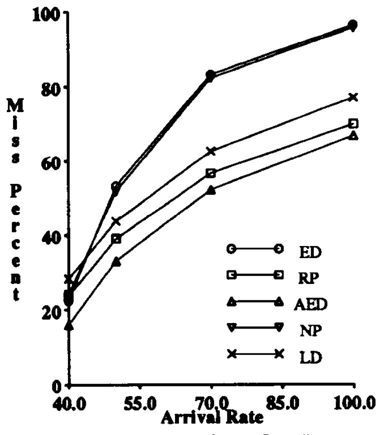

Figure 3b: RC (Heavy Load)

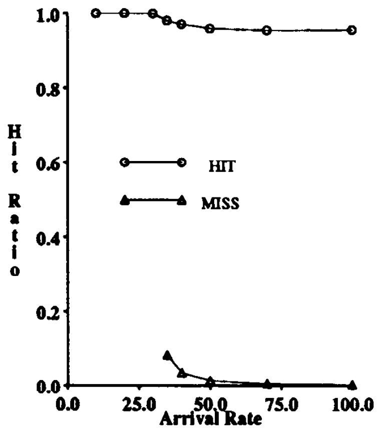

Figure 3c: AED (Group Hit Ratio)

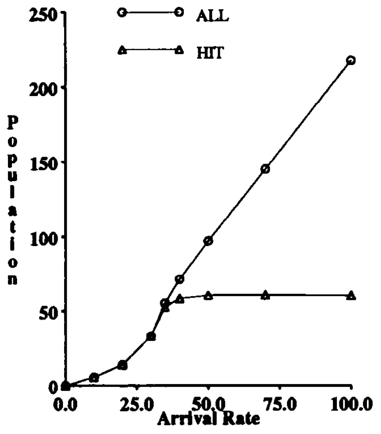

Figure 3d: AED (Population)

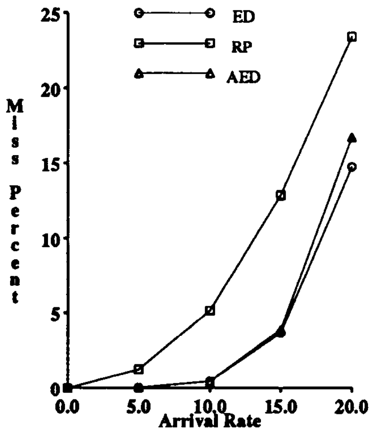

Figure 4a: RC + DC (Normal Load)

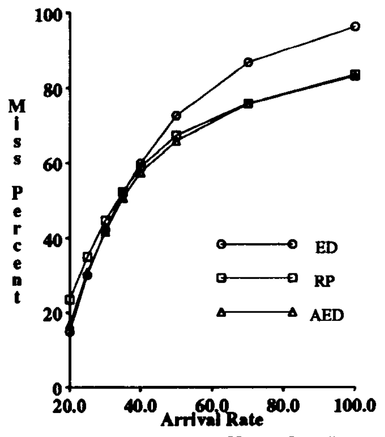

Figure 4b: RC + DC (Heavy Load)

Earliest Deadline schedule. For system loadings above this level, a pure Earliest Deadline schedule causes most transactions to miss their deadline since they receive high priority only when they are close to missing their deadline. The AED mapping, however, by its division of transactions into different priority groups, creates a "core set" of transactions in the *HIT* group that are virtually certain to make their deadlines independent of system loading conditions. Viewed from a different perspective, we have revisited the classic multi-programming thrashing problem where increasing the number of transactions in a system can lead to *decreased* throughput. In our framework, adding transactions to a set of transactions that can just be completed with an Earliest Deadline schedule causes more missed deadlines.

As promised in the description of the AED algorithm in Section 2.5, we now provide the rationale for using a Random Priority mapping in the *MISS* group. The reason is the following: Transactions assigned to the *MISS* group essentially "see" a heavily-loaded system due to having lower priority than transactions of the *HIT* group. Since our experiments show Random Priority to have the best performance among the non-adaptive algorithms at heavy loads, we have chosen this priority ordering for the *MISS* group. The reason that AED does better than the pure RP mapping at heavy loads is that the transaction population in the *HIT* group is sufficiently large that using ED, instead of RP, among this set has an appreciable performance effect. As the loading level is increased even further, however, the performance of AED would asymptotically reach that of RP since the number of transactions in the *HIT* group would be small compared to the total number of trasnactions in the system.

Summarizing the results of the above set of experiments, we can draw the following conclusions for the *query* workloads examined in this section: First, the AED mapping provides the best overall performance. Its feedback mechanism is effective in detecting overload conditions and limiting the size of the *HIT* group to a level that can be handled by an Earliest Deadline schedule. Second, at normal loads, the Earliest Deadline priority ordering meets most transaction deadlines and is therefore the right priority mapping in this region. At heavy loads, however, the Random Priority algorithm delivers the best performance among the non-adaptive algorithms due to guaranteeing the completion of high-priority transactions.

As in this experiment, we observed No Priority and Latest Deadline to perform poorly for the other workloads that we considered. Therefore, we will present further results only for the Earliest Deadline, Random Priority and Adaptive Earliest Deadline priority mappings.

5.2. Resource and Data Contention (RC + DC)

Our next experiment explored the situation where both resource contention *and* data contention contribute towards system performance degradation. This was done by changing the write probability from 0.0 to 0.25, which implies that one-fourth of the data items that are read will also be updated. The 2PL-HP algorithm (see Section 4) is used as the concurrency control mechanism since the workload now includes transactions that belong to the *updater* class.

For this experiment, Figures 3a and 3b show the Miss Percent results for the various priority mappings under normal load and heavy load conditions, respectively. From these figures it is evident that Earliest Deadline performs the best at low loads, while Random Priority is superior at heavy loads. The AED mapping behaves almost as well as ED at low loads and behaves like RP in the overload region, thus providing the best overall performance. In this experiment, the increased contention levels cause the population in the *HIT* group to be quite small compared to the overall system population at heavy loads. Therefore, using ED instead of RP in this group does not have an appreciable performance effect. We therefore see that the performance of AED approaches that of RP at a lower load than in the pure resource contention experiment (see Figure 2b).

From the above set of experiments, we observe that the AED algorithm is capable of performing well under both resource contention and data contention. We conducted further experiments to examine the effects of changes in deadline assignments, transaction write probabilities, hardware resource quantities, etc. The results of these experiments reinforced the general conclusions given above. We also conducted a few preliminary experiments to determine how well the AED algorithm could adapt to bursty transaction arrival patterns. In these experiments, the AED algorithm proved to be robust in its performance [Hari91b].

We have seen that the AED algorithm exhibits ED-like behavior in the light-load region and RP-like behavior in the overload region. From these results, it might appear that a much simpler approach than AED would be to switch from ED to RP (for all transactions) when the transaction miss percentage exceeds a threshold. The threshold, of course, would be the miss level at which RP starts performing better than ED. The problem with this approach is that we do not a-priori *know* this changeover threshold. Also, the threshold is a function of workload characteristics and may vary dynamically with changes in the input workload. For example, in the pure resource contention experiment, the ED to RP changeover miss percent threshold is about 25 percent (see Figure 2a); in the resource plus data contention experiment, however, the threshold is about 50 percent (see Figure 3b). Therefore, while the AED algorithm is complicated, the complexity appears necessary to make the priority assignment adapt to changing workload and system conditions.

6. Extending AED for Transaction Values

In this section, we consider the case where transactions have different values assigned to them. The goal here is to maximize the sum of the values of those transactions that commit by their deadline, and minimizing the number of missed deadlines becomes a secondary concern. A fundamental problem when transactions are characterized by both value and deadline is how to construct a priority ordering, since this requires a tradeoff to be established between these two orthogonal characteristics. In [Hari91a], several priority mappings that establish different, but fixed, tradeoffs between value and deadline were investigated. It was found that one of two mappings – either Earliest Deadline (ED) or Highest Value (HV), which implement extreme tradeoffs – almost always provided the best performance. The Earliest Deadline mapping is the same as that discussed so far, and transaction values are not taken into account. In the Highest Value mapping, transactions with higher value are given higher priority, and transaction deadlines are ignored. For transaction workloads with a limited, uniform spread in their values, Earliest Deadline provided the best performance at light loads. Under heavy loads, however, the Highest Value mapping generated the most value. For workloads that had a large spread or pronounced skew in transaction values, the Highest Value mapping was found to deliver the best performance throughout virtually the entire loading range.

In this section, we present a value-based extension of the AED algorithm called **Hierarchical Earliest Deadline (HED)**, which adaptively varies the tradeoff between value and deadline to maximize the value realized by the system. Informally, the HED algorithm groups transactions, based on their values, into a hierarchy of prioritized buckets. It then uses an AED-like algorithm within each bucket to determine the relative priority of transactions belonging to the bucket. The details of the HED algorithm are described below, after which the rationale behind the construction of the algorithm is discussed.

6.1. Bucket Assignment

The HED algorithm functions in the following manner: The *priority mapper* unit maintains a *value-based* dynamic list of buckets, as shown in Figure 5. Every transaction, upon arrival, is assigned based on its value to a particular bucket in this list.

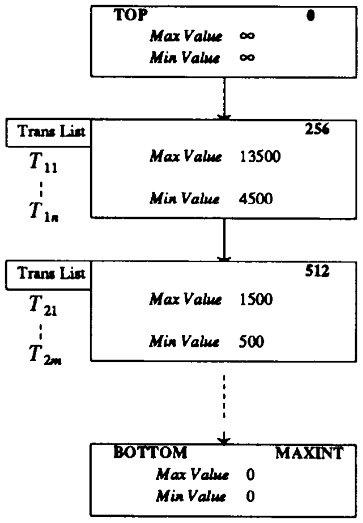

Figure 5: HED Bucket Hierarchy

Each bucket in the list has an associated *MinValue* and *MaxValue* attribute – these attributes bound the values that transactions assigned to the bucket may have. Each bucket also has an identifier, and bucket identifiers in the list are in monotonically increasing order. There are two special buckets, TOP and BOTTOM, that are always at the head and tail of the list, respectively. The *MinValue* and *MaxValue* attributes of TOP are set to ∞, while the *MinValue* and *MaxValue* attributes of BOTTOM are set to zero. Since we assume that all transaction values are finite and positive, no transactions are ever assigned to these buckets, and their function is merely to serve as permanent list boundaries. The identifiers of the TOP and BOTTOM buckets are preset to 0 and MAXINT, respectively.

When a new transaction, T_{new}, arrives in the system, it is assigned to the bucket closest to TOP that satisfies the constraint $MinValue \le Value_{new} \le MaxValue$. If no such bucket exists, a new bucket is inserted in the list between the bucket closest to TOP that satisfies $MinValue < Value_{new}$ and its predecessor, and the transaction is assigned to this bucket. A newly created bucket is assigned its identifier by halving the sum of the identifiers of its predecessor and successor buckets. For example, a bucket inserted between buckets with identifiers 256 and 512 will have 384 as its identifier. When a transaction leaves the system, it is removed from its assigned bucket. A bucket that becomes empty is deleted from the bucket list.

The *MinValue* and *MaxValue* attributes of a bucket are set as follows: Each bucket maintains an *AvgValue* attribute that monitors the average value of the set of transactions that are *currently* assigned to the bucket. The *MinValue* and *MaxValue* attributes of the bucket are then computed as (*AvgValue*/*SpreadFactor*) and (*AvgValue***SpreadFactor*), respectively, where *SpreadFactor* is a parameter of the HED algorithm. The *SpreadFactor* parameter controls the maximum spread of values allowed within a bucket. Whenever a transaction enters or leaves the system, the associated bucket updates its *AvgValue*, *MinValue* and *MaxValue* attributes.

6.2. Group Assignment

In similar fashion to the AED algorithm, transactions in each bucket are divided into *HIT* and *MISS* groups, with the *HIT* group size controlled by a *HITcapacity* variable. After a new transaction has been assigned to a bucket, its group assignment within the bucket is as follows: The transaction is first given a randomly chosen unique integer key, I_T. It is then inserted into a *value-ordered* list of transactions belonging to the bucket, with transactions that have identical values being ordered by their I_T keys. The position of the new transaction in the list, pos_T, is noted. If pos_T is less than or equal to the *HITcapacity* of the bucket, the new transaction is assigned to the *HIT* group in the bucket; otherwise, it is assigned to the *MISS* group. The *HITcapacity* computation in each bucket is implemented with a separate feedback process; each feedback process is identical to that described for the AED algorithm in Section 2.5.

6.3. Priority Assignment

After its bucket and group assignment, a new transaction is assigned its priority using the following formula:

$$P_T = \begin{cases} (B_T,\ 0,\ D_T,\ I_T) & \text{if } Group = HIT \\ (B_T,\ 1,\ 1/V_T,\ I_T) & \text{if } Group = MISS \end{cases}$$

where B_T is the identifier of the transaction's bucket.

The above priority assignment results in transactions of bucket i having higher priority than all transactions of bucket j for $j > i$, and lower priority than all transactions of bucket g for $g < i$. Within each bucket, transactions in the *HIT* group have a higher priority than transactions in the *MISS* group. The transaction priority ordering in the *HIT* group is Earliest Deadline, while the priority ordering in the *MISS* group is Highest Value. The I_T priority component serves to break the tie for transactions in the *HIT* or *MISS* group that have identical deadlines or values, respectively. This ensures a *total* priority ordering of all transactions in the system.

As mentioned earlier, the priority assignment process within each bucket is similar to that of the AED algorithm. There are, however, two important differences: First, the transaction list within a bucket is ordered based on transaction values, instead of transaction keys. Second, the priority ordering within the *MISS* group is Highest Value instead of Random Priority.

6.4. Discussion

The core principle of the AED mapping is to use an Earliest Deadline schedule among the largest possible set of transactions that can be completed by their deadline, i.e. the *HIT* group. The HED mapping extends this principle in two ways: First, within a bucket, it ensures that higher-valued transactions are given precedence in populating the *HIT* group, as this should increase the realized value. Second, by creating a value-based hierarchy of buckets, the HED algorithm ensures that transactions with substantially different values are not assigned to the same bucket. The reason for doing this is the following: The AED algorithm only *approximates* a hit ratio of 1.0 in the *HIT* group. Therefore, there is always the risk of losing an extremely high-valued transaction since transactions within the *HIT* group are prioritized by deadline and not by value. Missing the deadlines of such "golden" transactions can seriously affect the realized value; our solution is to establish a value-based bucket hierarchy, thus ensuring the completion of high-valued transactions.

7. Experiments and Results

In this section, we present performance results for our experiments comparing the Earliest Deadline, Highest Value and Hierarchical Earliest Deadline priority mappings when transactions have different values. The performance metric used now is *Loss Percent*, which is the ratio of the sum of the values of late transactions to the total input value, i.e., it is the percentage of the offered value that is *not* realized by the system. Just as for the earlier Miss Percent figures, Loss Percent values in the range of 0 to 20 percent and 20 to 100 percent are taken to represent system performance under "normal" loadings and "heavy" loadings, respectively.

We experimented with two value assignment distributions: Uniform and Skewed. In the Uniform distribution, transactions were randomly assigned values from a uniform distribution ranging between 50 and 150. In the Skewed distribution, 10 percent of the transactions constituted 90 percent of the offered value. Transactions belonging to this group had values ranging uniformly between 450 and 1350, while the remaining 90 percent had values ranging between 6 and 16. The average value of a transaction for both distributions is thus the same, namely 100.

While we evaluated the performance of the mappings for a variety of workloads, due to space constraints we will discuss only the results obtained for the case where system performance is limited by both resource contention and data contention. The settings of the workload and system parameters are the same as those for the experiment of Section 5.2. The *SpreadFactor* parameter of the HED algorithm is set to 3 in the experiments described here (we comment on this choice in Section 8).

7.1. Uniform Value Distribution

Our first experiment investigated the performance of the priority mappings for the Uniform transaction value workload. For this experiment, Figures 6a and 6b show the Loss Percent results under normal load and heavy load conditions, respectively. From this set of graphs, it is clear that at normal loads, the Earliest Deadline (ED) mapping delivers the most value. This might be considered surprising since ED is a value-indifferent mapping, while the Highest Value (HV) mapping is value-cognizant. The reason for ED's good performance is that it misses the deadlines of very few (if any) transactions and therefore delivers the most value. In contrast, the Highest Value mapping focuses its effort on completing the high-value transactions. In the process, it prevents some lower value transactions from making their deadlines, even though most of these deadlines could have been met (as demonstrated by ED), thereby losing more of the offered value. As the system load is increased, however, the performance of ED steeply degrades, while the performance of HV becomes considerably superior. This is because following the Highest Value principle is a better idea at high loads since the system has sufficient resources to handle only a fraction of the transactions in the system. In such a situation, the transactions that should be run are those that can deliver high value.

Moving on to the HED mapping, we note that at normal loads it behaves almost identically to Earliest Deadline. Then, as the overload region is entered, it changes its behavior to be similar to that of Highest Value. Therefore, in an overall sense, the HED mapping delivers the best performance. It should be noted that for this uniform workload, all transactions are assigned to the same bucket since transaction values are all within a factor of 3 (the SpreadFactor setting) of each other.

Summarizing the above results, we can draw the following conclusions for the uniform value workload: First, the HED

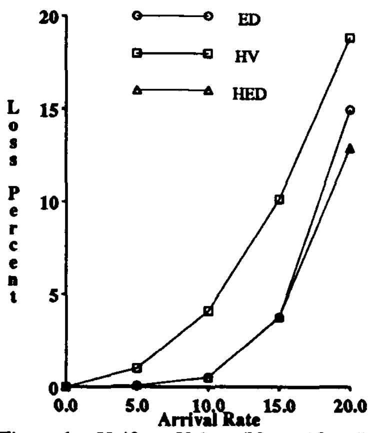

Figure 6a: Uniform Values (Normal Load)

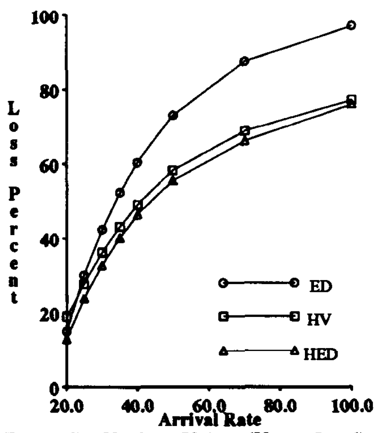

Figure 6b: Uniform Values (Heavy Load)

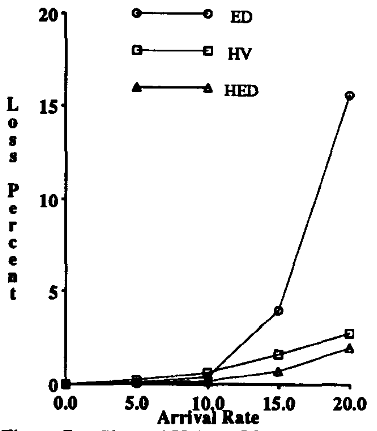

Figure 7a: Skewed Values (Normal Load)

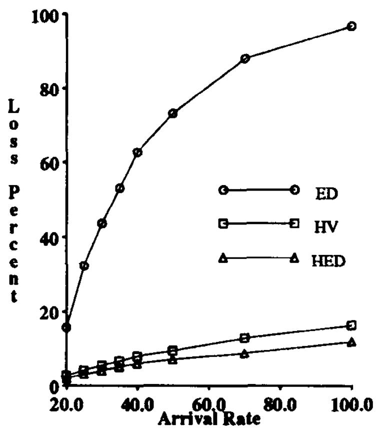

Figure 7b: Skewed Values (Heavy Load)

mapping provides the best overall performance. Its feedback mechanism is effective in detecting overload conditions and limiting the size of the *HIT* group to a manageable number. It also realizes a high value by populating the *HIT* group with higher-valued transactions. Second, under normal loads, the Earliest Deadline priority ordering meets most transaction deadlines and is therefore the right schedule in this region. Under high loads, however, the Highest Value mapping delivers good performance as it guarantees the completion of high-value transactions.

7.2. Skewed Value Distribution

The next experiment examined the effect of having a skew in the transaction value distribution. For this experiment, the Skewed value distribution was used to assign values to transactions. The Loss Percent results for this experiment are shown in Figures 7a and 7b. From these figures we note that the performance of the Earliest Deadline (ED) mapping remains the same as for the Uniform value distribution (compare with Figures 6a and 6b). This is because the ED mapping is value-indifferent. The figures also show that the performance of the Highest Value (HV) mapping improves greatly as compared to the Uniform case. Note that even at low load, the HV mapping performs almost as well as the ED mapping. The HV mapping, by making certain that all of the (few) high-value transactions make their deadline, ensures that it always realizes at least 90 percent of the offered value. In addition, at low loads, the value of the missed transactions constitutes a very small fraction of the total value, and the performance impact of having a higher number of missed deadlines than ED is therefore negligible.

If we now consider the HED mapping, we observe that it performs better than both ED and HV over the entire loading range. The reason for its good performance is twofold: First, the bucket hierarchy construction ensures that the few high-valued transactions are assigned to a separate higher priority bucket. This guarantees that these transactions are completed and therefore their value is realized. Second, using the AED policy within each bucket results in more deadlines being made and a corresponding increase in the realized value.

8. Conclusions

In this paper, we have addressed the issue of stabilizing the overload performance of Earliest Deadline in real-time database systems for applications with firm deadlines. Our operating constraint is that a-priori knowledge of transaction resource requirements or data access patterns is not available. We introduced the Adaptive Earliest Deadline (AED) priority assignment algorithm and, using a detailed simulation model of a real-time database system, studied its performance relative to Earliest Deadline and other fixed priority mappings. Our experiments showed that for the workloads considered in this study, which examined both resource contention in isolation and in association with data contention, the AED algorithm delivered the best overall performance. At light loads, it behaved exactly like Earliest Deadline; at high loads its behavior was similar to that of Random Priority, which was the best performer among the fixed priority mappings. The feedback control mechanism of AED was found to be accurate in estimating the number of transactions that could be sustained under an ED schedule. AED's policy of restricting the use of the Earliest Deadline approach to the *HIT* group delivered stabilized performance at high loads. The AED algorithm has also been observed to be robust to limited fluctuations in the transaction arrival pattern.

In some real-time applications, different transactions may be assigned different values. Assigning priorities to transactions when they are characterized by both values and deadlines is a challenging problem. We introduced the Hierarchical Earliest Deadline (HED) priority assignment algorithm here to address this issue. The HED algorithm groups transactions, based on their value, into a hierarchy of prioritized buckets; it then uses the AED algorithm within each bucket. Using our RTDBS simulation model, we evaluated the performance of HED with respect to mappings that establish fixed tradeoffs between values and deadlines. Our experiments showed that, both for workloads with limited spread in transaction values and for workloads with pronounced skew in transaction values, the HED algorithm provided the best overall performance. At light loads, its behavior was identical to that of Earliest Deadline, while at heavy loads its performance was better than that of Highest Value. Use of the AED algorithm within the transactions of a bucket decreased the number of missed deadlines. Also, by giving preference to more valuable transactions in populating the *HIT* group of each bucket, the HED algorithm increased the realized value. For workloads with pronounced skew in transaction values, the hierarchical nature of the HED algorithm was effective in ensuring that "golden" (high-valued) transactions were completed and their value realized.

While the AED and HED algorithms appear promising in their approach and performance, they have some limitations in their current form. In particular, they have several algorithmic parameters (HITbatch, SpreadFactor, etc.) that need to be set by the database administrator. The settings in the experiments discussed here were arrived at after experimentation with several different choices. However, these settings may not prove suitable for other workloads and environments. Therefore, a mechanism that adaptively generates the right settings is required. Another limitation of the algorithms is that they assume a transaction workload that is homogeneous in its characteristics, which is not always the case in practice. We hope to address these limitations in our future research.

REFERENCES

[Abbo88] Abbott, R., and Garcia-Molina, H., "Scheduling Real-Time Transactions: A Performance Evaluation," *Proc. of the 14th Int. Conf. on Very Large Database Systems*, Aug. 1988.

[Abbo89] Abbott, R., and Garcia-Molina, H., "Scheduling Real-Time Transactions with Disk Resident Data," *Proc. of the 15th Int. Conf. on Very Large Database Systems*, Aug. 1989.

[Bern87] Bernstein, P., Hadzilacos, V., and Goodman, N., "Concurrencyn Control and Recovery in Database Systems," Addison-Wesley, 1987.

[Biya88] Biyabani, S., Stankovic, J., and Ramamritham, K., "The Integration of Deadline and Criticalness in Hard Real-Time Scheduling," *Proc. of the 9th IEEE Real-Time Systems Symposium*, Dec. 1988.

[Buch89] Buchmann, A., et al, "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control," *Proc. of the 5th Int. Conf. on Data Engineering*, Feb. 1989.

[Eswa76] Eswaran, K., et al, "The Notions of Consistency and Predicate Locks in a Database System," *Comm. of the ACM*, Nov. 1976.

[Hari90a] Haritsa, J., Carey, M., Livny, M., "On Being Optimistic about Real-Time Constraints," *Proc. of the 1990 ACM PODS Symposium*, April 1990.

[Hari90b] Haritsa, J., Carey, M., Livny, M., "Dynamic Real-time Optimistic Concurrency Control," *Proc. of 11th IEEE Real-Time Systems Symposium*, Dec. 1990.

[Hari91a] Haritsa, J., Carey, M., Livny, M., "Value-Based Scheduling in Real-Time Database Systems," *Tech. Report 1024*, Univ. of Wisconsin, Madison, May 1991.

[Hari91b] Haritsa, J., "Transaction Scheduling in Firm Real-Time Database Systems," *PhD. Thesis*, Computer Sciences Department, Univ. of Wisconsin, Madison, August 1991.

[Huan89] Huang, J., et al, "Experimental Evaluation of Real-Time Transaction Processing," *Proc. of 10th IEEE Real-Time Systems Symposium*, Dec. 1989.

[Jens85] Jensen, E., Locke, C., and Tokuda, H., "A Time-Driven Scheduling Model for Real-Time Operating Systems," *Proc. of 6th IEEE Real-Time Systems Symposium*, Dec. 1985.

[Liu73] Liu, C. and Layland, J., "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, Jan. 1973.

[Stan88] Stankovic, J. and Zhao, W., "On Real-Time Transactions," *ACM SIGMOD Record*, March 1988.