EARLIEST DEADLINE SCHEDULING FOR REAL-TIME DATABASE SYSTEMS

by

Jayant R. Haritsa
Miron Livny
Michael J. Carey

Computer Sciences Technical Report #1025

May 1991

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

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

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 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 much worse than most other policies. This is due to Earliest Deadline giving the highest priority to tasks that are close to missing their deadlines. Previously suggested schemes for stabilizing the overload performance of Earliest Deadline require a-priori knowledge of task processing requirements. In the context of real-time database systems, however, knowledge of transaction resource requirements or data access semantics is in most cases not available, and alternate schemes are therefore required.

In this paper, we present a new priority assignment policy called Adaptive Earliest Deadline (AED). The AED policy delivers stable overload performance, while retaining the light-load virtues of Earliest Deadline. It features a feedback control mechanism that detects overload conditions and modifies transaction priority assignments accordingly. Since AED assumes no knowledge of transaction requirements, it is suitable for use in real-time database systems. Using a detailed simulation model, we compare the performance of the AED policy with respect to Earliest Deadline and other fixed priority schemes. We also present and evaluate in this paper an extension of the AED policy called Hierarchical Earliest Deadline (HED). The HED priority policy is designed to handle applications where transactions may be assigned different values and the goal of the real-time database system is to maximize the total value of the in-time transactions.

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

1. INTRODUCTION

A Real-Time Database System (RTDBS) is a transaction processing system that attempts to satisfy the timing constraints associated with each incoming transaction. Typically, a time constraint is expressed in the form of a *deadline*, that is, the application submitting the transaction would like it to be completed before a certain time in the future. Accordingly, a higher quality of service is associated with processing transactions before their deadlines as compared to completing them late. In contrast to a conventional DBMS, where the goal usually is to minimize response times, the emphasis here is on meeting transaction deadlines. An RTDBS thus has the task of enforcing data integrity constraints and satisfying transaction time constraints [Stan88, Buch89].

Several previous studies of real-time database systems (e.g. [Abbo88, Abbo89]) have addressed the issue of scheduling transactions with the objective of minimizing the percentage 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 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 a transaction to complete before its deadline. Under heavy loads, then, a fundamental weakness of the Earliest Deadline policy is that it assigns the highest priority to transactions that are close to missing their deadlines, thus delaying other transactions that can still meet their deadlines.

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 "high-miss" performance without sacrificing its "low-miss" virtues? A well-constructed scheme 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 policy called Adaptive Earliest Deadline (AED) that stabilizes the overload performance of Earliest Deadline in an RTDBS environment. The AED policy 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 policy with that of Earliest Deadline and other fixed priority mappings under a variety of workloads.

There are real-time database applications where transactions may be assigned *different values*, 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 such an environment. A fundamental problem in this scenario 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 have developed a new priority assignment policy called Hierarchical Earliest Deadline (HED) for integrating the value and deadline characteristics of transactions. The HED policy is an extension of the AED mapping. It adaptively varies the tradeoff between value and deadline, based on transaction deadline miss levels, so as to maximize the value realized by the system. In this paper, we compare the performance of the HED policy 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 generates a priority for a transaction on its arrival, and this priority is subsequently used throughout the system to resolve contention for both hardware resources and data objects. The design is modular since it allows *priority generation* to be separated from *priority usage*. We take this approach since the use of multiple priority mappings within a system makes it difficult to identify the source of performance behavior.

In this paper, we restrict our attention to real-time database systems that execute *firm deadline* applications. For such applications, a transaction whose deadline is missed loses its value. This means that transactions are *discarded* when the system 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 knowledge of transaction service requirements. Real-world examples of applications with firm deadlines are given in [Abbo88].

The remainder of this paper is organized in the following fashion: Section 2 provides a historical overview of the Earliest Deadline policy. The AED priority policy is presented in Section 3 along with several fixed priority mappings. Section 4 describes the concurrency control algorithm used in the study. Then, in Section 5, we describe our RTDBS model and its parameters, while Section 6 highlights the results of the AED simulation experiments. We then go on to present the HED priority policy in Section 7, and evaluate its performance in Section 8. Finally, Section 9 summarizes the main conclusions of the study and outlines future avenues to explore.

2. RELATED WORK

A great deal of attention has been devoted by the real-time research community to the problem of dynamic (on-line) scheduling of tasks with deadlines. Liu and Layland, in their classic paper [Liu73], showed that the Earliest Deadline priority policy is optimal for *periodic* task sets. It is optimal in the sense that if a set of tasks can be successfully scheduled by some priority policy, then this task set is guaranteed to be successfully scheduled by the Earliest Deadline policy also. In [Dert74], the optimality of Earliest Deadline was extended to *arbitrary* task sets executing on a single processor. For multi-processor systems, however, the policy ceases to be optimal except under restricted circumstances [Mok78]. In fact, a stronger result presented in [Mok78] is that there are *no* optimal on-line scheduling policies for arbitrary task sets executing on multiprocessor systems.

We now turn our attention to real-time systems operating under firm deadlines, where optimality is defined in terms of minimizing the number of missed task deadlines. For this framework, Panwar and Towsley proved in [Pan88] that the Earliest Deadline policy is optimal for G/M/c/G queueing systems. (The notation G/M/c/G specifies that the task arrival process can have an arbitrary distribution, the task service times are exponentially distributed (memoryless), the number of servers is arbitrary, and task deadlines can have an arbitrary distribution.) The underlying assumptions under which the optimality result holds is that task execution times are independent of their relative deadlines and inter-arrival times. In practice, however, these distribution and independence constraints may

not be satisfied by real-time tasks. For example, task execution times are usually bounded by their relative deadlines and task deadlines are often correlated with their execution times.

Several simulation studies have evaluated the performance of dynamic scheduling policies in firm deadline real-time systems [Jens85, Abbo88, Huan89, Hari91a]. A common observation of these studies has been, as mentioned in the introduction, that the Earliest Deadline scheduler provides unstable performance at high loads. A method for stabilizing the overload performance of Earliest Deadline was presented in [Jens85]. This algorithm requires a-priori knowledge of task processing requirements and operates in the following manner: The scheduler starts with a deadline-ordered sequence of the available tasks, which is then sequentially checked for its probability of missing the next transaction's deadline. At any point in the sequence at which this overload probability passes a preset threshold, the task prior to the overload with the longest processing time is removed from the sequence. The removal process is repeated until the overload probability is reduced to acceptable levels. A simulation-based performance study of the algorithm showed that, for appropriate settings of the overload probability threshold, it performed as well as Earliest Deadline at low miss levels and considerably better at high miss levels.[1]

While the above method is a promising approach for task sets that are fully-characterized, and has been incorporated in the Archons project [Lock86], it does not easily carry over to *transaction scheduling* in real-time database systems. This is because database transactions are typically complex and detailed knowledge of their hardware resource requirements or data access semantics is not available in most cases [Abbo88]. Also, even if this information were available, determining the overload probability may become computationally expensive or even impossible since database systems interact with the operating system and I/O subsystem in unpredictable ways. Transactions require concurrency control, commit atomicity, recovery safeguards, buffer management, and access to disks, thus making it difficult to accurately predict transaction processing times [Stan88].

In this paper, we consider the problem of developing a dynamic priority assignment policy for real-time transactions operating under firm deadlines. We assume that the priority assignment mechanism possesses no knowledge of transaction characteristics other than their deadlines, and that the system resources provide service based on transaction priorities. Our goal is to develop a computationally simple priority assignment policy that adapts to system loading levels and thereby minimizes the number of missed deadlines.

3. PRIORITY MAPPINGS

In order to resolve contention for hardware resources and data, an RTDBS has to establish a priority ordering among the transactions. The priority ordering should reflect the goal of minimizing the number of missed deadlines. Since we assume that the database system has no knowledge of transaction characteristics other than their deadlines, the choice of priority mappings is limited. Apart from the previously discussed Earliest Deadline policy, there are a few other mappings mentioned in the literature that fit our operating constraints. These mappings are described first in this section. Subsequently, the new priority mapping, Adaptive Earliest Deadline, is presented. In the following discussion, A_T, D_T, and P_T will be used to denote the arrival time, deadline, and priority of transaction T,

[1] The stabilized algorithm described in [Jens85] is actually more general, and can handle tasks with differing values. The description here has been modified to suit the context of the discussion which assumes that all tasks have the same value.

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.

3.1. Earliest Deadline (ED)

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

3.2. 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 can be interpreted as the performance that would be observed if the real-time database system were to be replaced by a conventional DBMS and the feature of discarding late transactions was retained.

3.3. Random Priority (RP)

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

3.4. 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 $P_T = 1 / D_T$. A new transaction usually tends to have a later deadline 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.

3.5. Adaptive Earliest Deadline (AED)

The Adaptive Earliest Deadline policy takes the approach of modifying the basic Earliest Deadline mapping by using the following observation: Given a set of tasks with deadlines that can all somehow be met, an Earliest Deadline priority ordering meets all (or most of) the deadlines [Jens85]. The implication of this observation is that in order to maximize the number of in-time transactions, we would like to have an Earliest Deadline schedule 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. Instead, if the system could "magically" determine at arrival time the eventual completion status (in-time or late) of a transaction, it should use an Earliest Deadline schedule among the in-time transactions and discard the late transactions. In the absence of such perfect foresight, alternate methods are required to estimate the completion status of a transaction. The AED policy takes the approach of using a feedback control process as the estimation method.

In the AED algorithm, transactions executing in the system are collectively divided into two groups, *HIT* and *MISS*, as shown in Figure 1. The assignment of a transaction to either of these groups is made when the transaction arrives at the system. When a new transaction arrives, it is assigned a unique[2] key, I_T, with the key being a randomly chosen integer. 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 *HITcapacity*, which is a dynamic control variable of the algorithm, the new transaction is assigned to the *HIT* group; otherwise, it is assigned to the *MISS* group.

The following formula is then used to assign a priority to the transaction:

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

This priority assignment scheme[3] results in all transactions in the *HIT* group having a higher priority than those in the *MISS* group. The priority ordering within transactions of the *HIT* group is Earliest Deadline. For transactions in this group that have identical deadlines, the I_T component of the priority serves to break the tie. The priority ordering within the *MISS* group is Random Priority, since the I_T's are selected randomly.

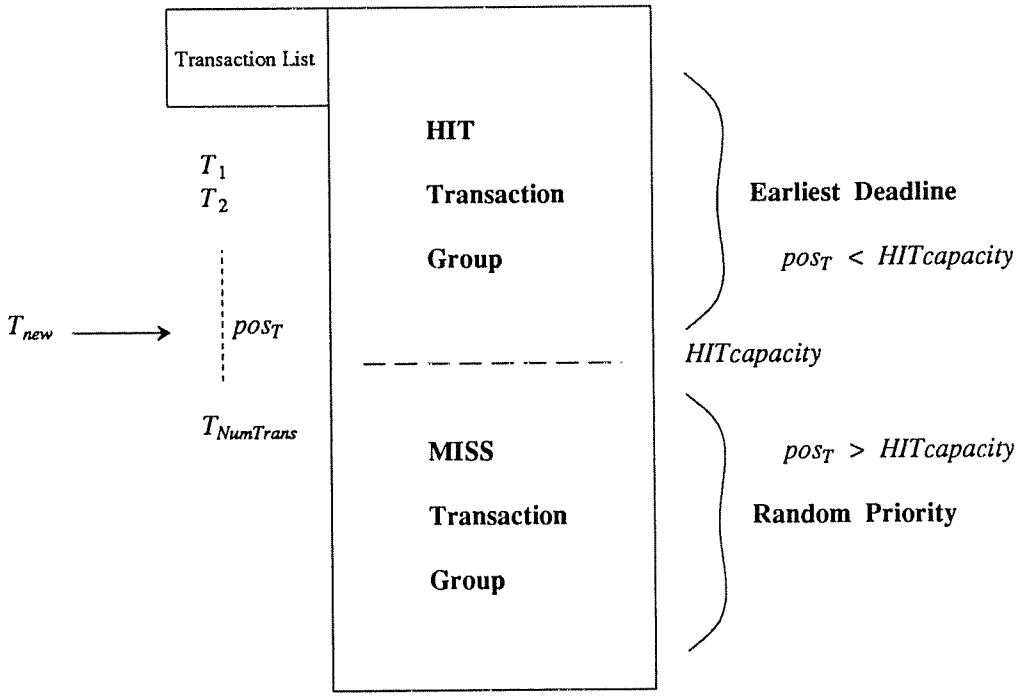

Figure 1: AED Priority Mapping

[2] Transaction keys are sampled uniformly over the set of integers. In the unlikely event that a new key matches that of an existing transaction, the key is re-sampled until a unique key is obtained.

[3] Since the transaction priority is a vector, priority comparisons are made in lexicographic order.

The goal of the AED policy is to collect in the *HIT* group the largest set of transactions that can be completed before their deadline. 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 described above. Ideally, we would like to have a hit ratio of 1.0 in the *HIT* group and 0.0 in the *MISS* group since this means that *all* the "doable" transactions are in the *HIT* group. Achieving this goal would require complete accuracy in predicting the completion status of a transaction; this is impossible given that the system 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 small hit ratio in the *MISS* group. The key to achieving this lies in the *HITcapacity* computation, which is discussed below. (The motivation for having a Random Priority mapping in the *MISS* group is explained in Section 6.)

3.5.1. HIT Capacity Computation

A feedback process that employs system output measurements is used to set the *HITcapacity* control variable. The measurements that are 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:

(STEP 1) *HITcapacity := HitRatio(HIT) * HITcapacity * 1.05;*

(STEP 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 with the largest 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, we have a close to optimal population size in the *HIT* group, and the HITcapacity remains stabilized at this setting (since $0.95 * 1.05 \approx 1.0$).

STEP 2 of the *HITcapacity* computation is necessary to take care of the following special scenario: If the system experiences a long period where the HitRatio(ALL) is close to 1.0 due to the system being lightly loaded, it follows that the HitRatio(HIT) will also be virtually 1.0. In this situation, the *HITcapacity* can become very large due to the 1.05 expansion factor. (For this expansion factor and for a HitRatio(HIT) of 1.0, the *HITcapacity* doubles every fifteen cycles of the feedback loop). If the transaction arrival rate now increases such that the system becomes overloaded (signaled by the HitRatio(ALL) falling below 0.95), bringing the *HITcapacity* down from its artificially high value to the right level could take a considerable amount of time. This means that the system may enter the unstable high-miss region of Earliest Deadline as every transaction is assigned to the *HIT* group because of 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.[4] The estimate is derived by computing the number of transactions that are *currently* making their deadlines. 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.

The feedback process for setting *HITcapacity* 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, where *HITbatch* is an algorithm parameter. At the RTDBS output, the completion status (in-time or late) of these specially-marked transactions is monitored. When the last of the *HITbatch* transactions exits the system, HitRatio(HIT) is measured as the fraction of these transactions that completed before their deadline. HitRatio(ALL), on the other hand, is continuously measured at the output as the hit ratio of the last *ALLbatch* transactions that exited from the system, where *ALLbatch* is an algorithm parameter. 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. At system initialization time, both the hit ratios are set to 1.0, while the *HITcapacity* is set equal to the database administrator's estimate of the number of concurrent transactions that the system can handle without missing deadlines. (Note that this estimate does not have to be accurate, and even if it were grossly wrong, would not impact system performance in the long run. The error in the estimate only affects how long it takes the *HITcapacity* to reach its steady state value at system startup time).

4. CONCURRENCY CONTROL ALGORITHMS

The resource scheduling policies used in most studies of real-time database systems (e.g. [Abbo88, Abbo89]) are pre-emptive resume based on priorities at the CPUs, and non-preemptive priority scheduling at the disks. These policies utilize the priority ordering established by the mappings described in the previous section in a straightforward manner. For implementing concurrency control, however, several different mechanisms are available, including locking (e.g. [Gray79]), timestamps (e.g. [Reed78]), and optimistic concurrency control (e.g. [Kung81]). While we have shown in previous studies [Hari90a, Hari90b, Hari91a] that optimistic algorithms outperform locking algorithms in a firm deadline RTDBS, we will consider only the 2PL-HP prioritized locking algorithm [Abbo88] in this paper. The reason is that the interaction of optimistic algorithms and priority policies is more complicated and space limitations prevent us from presenting these complexities here. The basic results about the superior performance of optimistic algorithms, however, are also true for the experiments considered in this study.

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 of 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. The High Priority

[4] The choice of 25 percent is based on our expectation that the *HitCapacity* estimate is fairly close to the "right" value.

scheme also serves as a deadlock prevention mechanism.[5]

5. REAL-TIME DBMS PERFORMANCE MODEL

A detailed model of a real-time database system was used to study the performance of the various priority mappings. The model is similar to that of our earlier studies [Hari90a, Hari90b, Hari91a]. In this model, the database system consists of a shared-memory multiprocessor operating on disk resident data.[6] 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.[7] A transaction that is restarted due to data conflict follows the same access pattern as the original transaction. If a transaction is not completed by its deadline, it is immediately aborted and discarded. The basic structure of the model is shown in Figure 2.

The model has five components: a *source* that generates transactions; a *transaction manager* that models the execution of transactions; a *concurrency control (CC) manager* that implements the details of the concurrency control algorithms; a *resource manager* that models the CPU and I/O resources; and a *sink* that gathers statistics on completed transactions. The *priority mapper* unit is embedded in the transaction manager. The following two subsections describe the workload generation process and the hardware resource configuration.

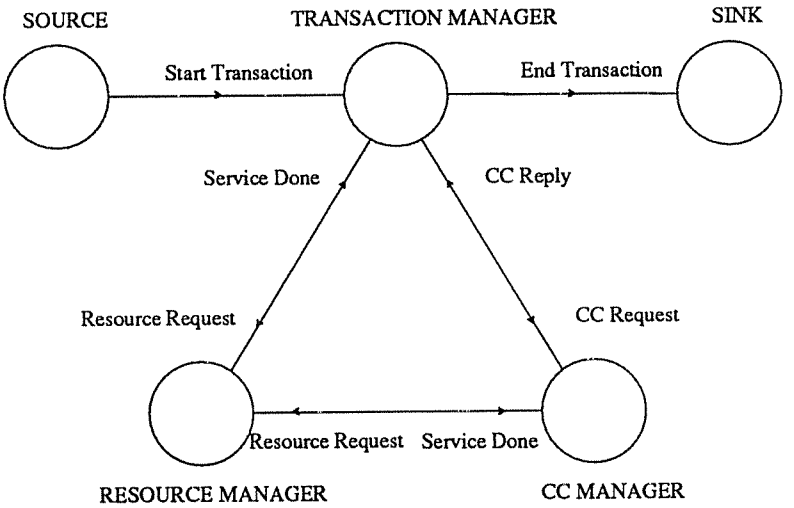

Figure 2: RTDBS Model Structure

[5] This is true only for priority assignment schemes that assign unique priority numbers to transactions and do not change a transaction's priority during the course of its execution.

[6] It is assumed, for simplicity, that all data is accessed from disk and buffer pool considerations are therefore ignored.

[7] We assume sufficient buffer space to allow the retention of updates until commit time, and we also assume the use of a log-based recovery scheme where only log pages are forced to disk prior to commit.

5.1. Workload Model

The workload model characterizes transactions in terms of the pages that they access and the number of pages that they update. Table 1 summarizes the key parameters of the workload model. The *ArrivalRate* parameter specifies the mean rate of transaction arrivals. The *DatabaseSize* parameter gives the number of pages in the database. The number of pages accessed by a transaction varies uniformly between half and one-and-a-half times the value of *PageCount*. 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. If we use the term *resource time* to denote the total service time at the resources that a transaction requires for its data processing, then $R_{\max}$ is the expected resource time of the largest transaction in our workload (i.e. a transaction accessing 1.5 * *PageCount* 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. A transaction is detected as being late only when it actually misses its deadline, as the system cannot estimate the remaining service requirements of the transaction.

5.2. Resource Model

The physical resources in our model consist of multiple CPUs and multiple disks. There is a single queue for the CPUs and the service discipline is preemptive-resume, with the preemption being based on transaction priorities. Each of the disks has its own queue and is scheduled with a non-preemptive priority scheduling policy. Table 2 summarizes the key parameters of the resource model. The *NumCPUs* and *NumDisks* parameters specify the hardware resource composition, while the *PageCPU* and *PageDisk* parameters capture CPU and disk processing times per data page. The data itself is modeled as being uniformly distributed across all of the disks.

Parameter	Meaning
ArrivalRate	Transaction arrival rate
DatabaseSize	Number of pages in database
PageCount	Avg. number of pages accessed/transaction
WriteProb	Write probability/accessed page
LSF	Low Slack Factor
HSF	High Slack Factor

Table 1: Workload Model Parameters

Parameter	Meaning
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: Resource Model Parameters

6. EXPERIMENTS AND RESULTS

In this section, we present performance results for our experiments comparing the various priority mappings in a real-time database system environment.[8] The simulator used to obtain the results is written in the Modula-2-based DeNet simulation language [Livn88]. We first describe the performance metrics and then list the baseline values for the system parameters. Subsequently, we discuss our results with regard to the impact of resource contention, data contention, and variations in workload characteristics.

6.1. Performance Metrics

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.[9] The simulator was instrumented to generate statistics about resource utilizations, mean transaction population, etc. These secondary measures help explain the behavior of the algorithms under various loading conditions. All the experiments evaluate the Miss Percent as a function of the transaction arrival rate.

6.2. Parameter Settings

The resource parameter settings (see Table 3) are such that the mean CPU time to process a page is 10 milliseconds while mean disk access times are 20 milliseconds. In all of our experiments, the number of processors and number of disks were set to 8 and 16, respectively. While describing the AED mapping in Section 3.3, 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.[10]

Having described the simulation model, transaction workload composition, and hardware resource configuration, we now go on to describe the results of the experiments evaluating the performance of the various priority mappings.

6.3. 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 resource 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

[8] All graphs in this paper show mean values with relative half-widths about the mean of less than 5% at the 90% confidence interval, with each experiment having been run until at least 20,000 transactions had been processed by the system. Only statistically significant differences are discussed here.

[9] Any long-term operating region where the loss percent is large is obviously unrealistic for a viable RTDBS. Exercising the system to high miss levels, however, provides valuable information on the response of the algorithms to brief periods of stress loading.

[10] The choice of sample size is constrained by two opposing considerations: A large sample size reduces the responsiveness of the feedback system. On the other hand, a small sample size makes the system unduly sensitive to short-term input fluctuations. Extensive experimentation with several different sample sizes showed that a setting of 20 delivers a reasonable tradeoff between these opposing considerations. A setting of exactly 20 is not *critical* to the performance of the policy, however; settings between 10 and 30 performed equally well.

experiment since all transactions belong to the *query* (read-only) class.

For this experiment, Figures 3a and 3b 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 become high priority transactions 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 by ED, however, is about the same as that of 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.

Turning our attention to the RP (Random Priority) mapping, we observe that it behaves poorly at normal loads since it does not 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.

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

Table 3: RC Parameter Settings

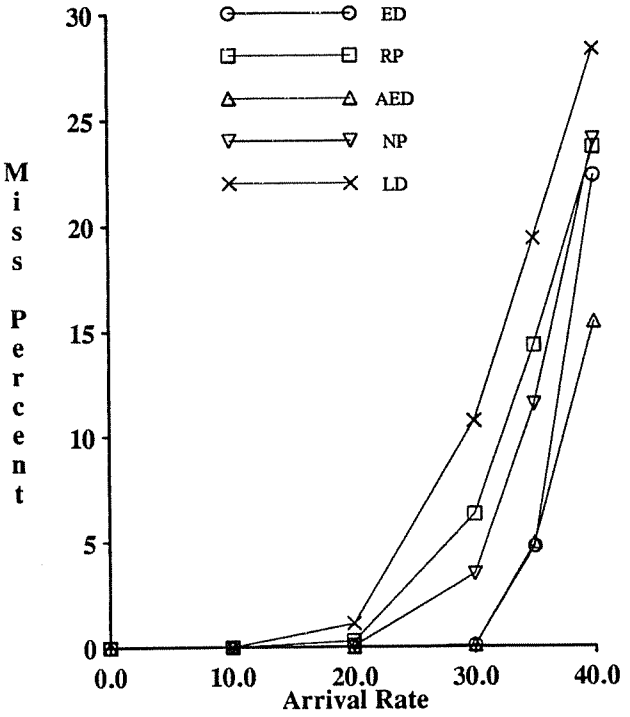

Figure 3a: RC (Normal Load)

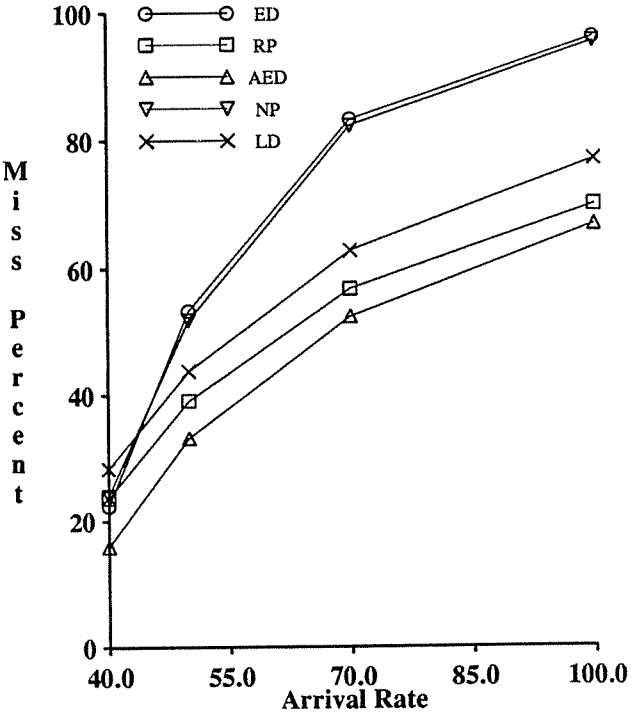

Figure 3b: RC (Heavy Load)

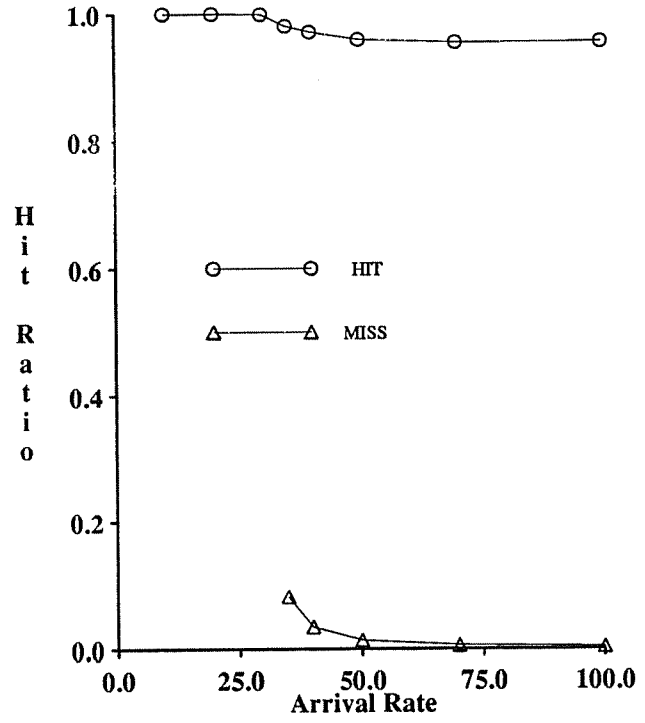

Figure 3c: AED Policy (Group Hit Ratio)

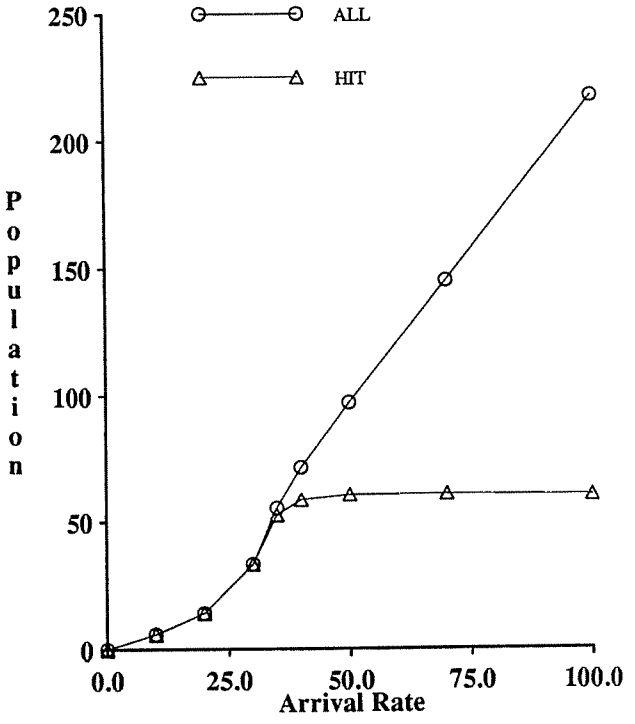

Figure 3d: AED Policy (Population)

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 3c, 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.[11] This indicates that the feedback mechanism used to divide transactions into *HIT* and *MISS* groups is effective and achieves our goal of having a high hit ratio in the *HIT* group and a low hit ratio in the *MISS* group. In Figure 3d, 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 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 of them to miss their deadlines.

As promised in the description of the AED algorithm in Section 3.5, 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 *MISS* group would be much larger than the number in the *HIT* group.

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.

Earlier studies [Jens85, Hari91a] have observed that the No Priority mapping performs worse than Earliest Deadline at low loads and about the same or worse at heavy loads. These behavioral characteristics were also seen

[11] In Figure 3c, the HitRatio(MISS) is not shown at low arrival rates because *no* transactions get assigned to the *MISS* group in this region due to the *HitCapacity* being greater than the maximum number of transactions in the system.

in the experiments of this study. In addition, we observed Latest Deadline to consistently perform worse than Random Priority for the workloads considered in this study. Therefore, for subsequent experiments, we will present results only for the Earliest Deadline, Random Priority and Adaptive Earliest Deadline priority mappings.

6.4. RESOURCE & 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 4a and 4b shows 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 3b).

From the above set of experiments, we can conclude that the AED policy delivers good performance across the entire loading range under both resource contention and data contention. We conducted several further experiments to examine the effects of changes in deadline assignments, transaction write probabilities, hardware resource quantities, etc. The results of all of these experiments reinforced the general conclusions given above.

6.5. BURST ARRIVALS

In the previously described experiments, each simulation was run for a particular arrival rate. In practice, however, the transaction arrival rate is usually a time-varying parameter. Therefore, we conducted experiments to determine how well the AED mapping could adapt to fluctuations in transaction arrival patterns. For this experiment, the transaction arrival process was constructed in the following manner: The transaction arrival rate is repeatedly toggled between a base arrival rate and a secondary arrival rate. The time period for which each arrival rate is in effect is chosen from a uniform distribution. This means that the *effective* transaction arrival rate is the average of the base and secondary arrival rates.

In our experiments for this type of transaction arrival process, the base arrival rate was kept fixed at 20 transactions/second and the experiment was run for different secondary arrival rates. The time period for which each arrival rate was in effect ranged uniformly between 10 and 40 seconds. For this workload, Figure 5 shows the MissPercent characteristics as a function of the *secondary* arrival rate for the pure resource contention workload. Figure 6 shows the corresponding MissPercent characteristics for the workload where there is both resource and data contention. From these figures, it is evident that the performance characteristics of the AED mapping are

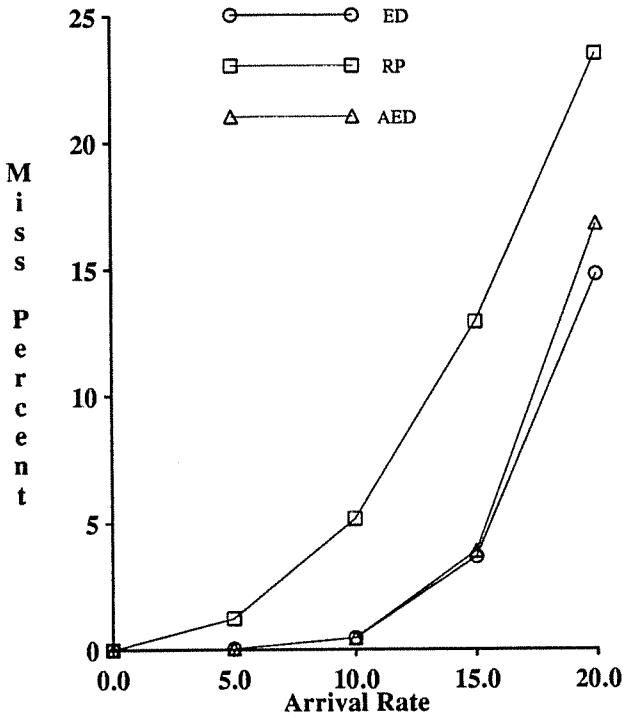

Figure 4a: RC+DC (Normal Load)

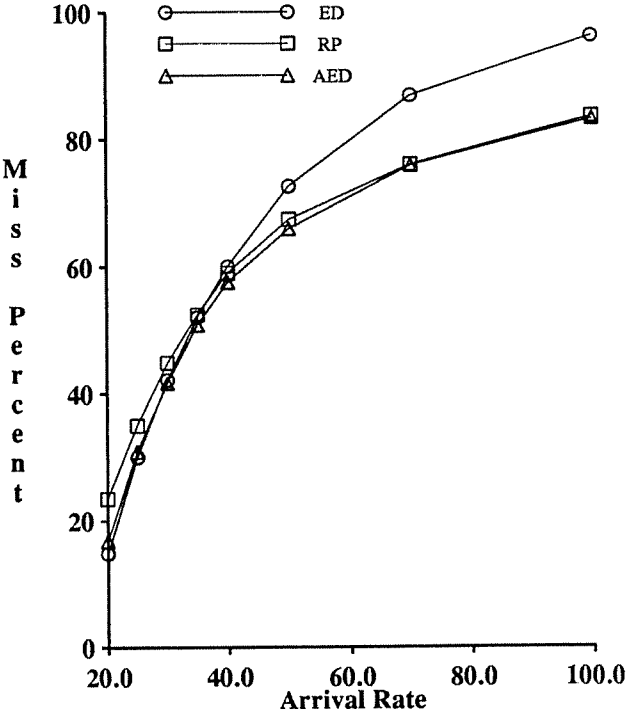

Figure 4b: RC+DC (Heavy Load)

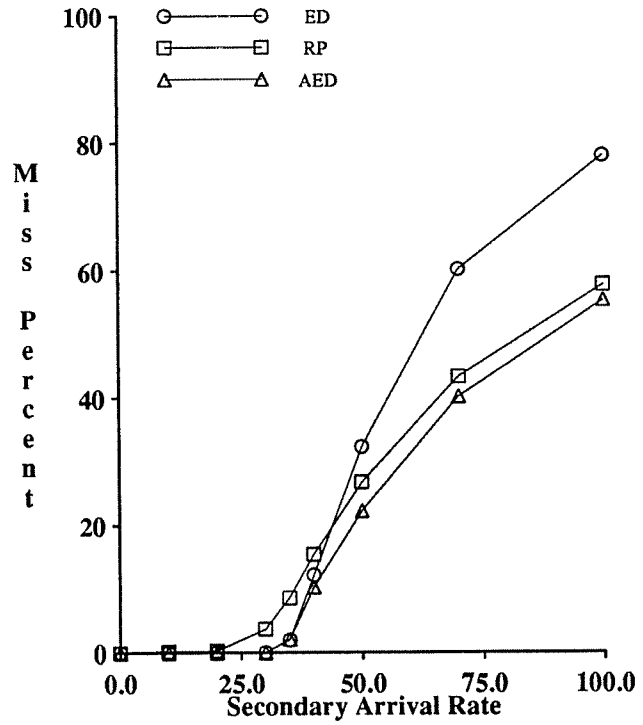

Figure 5: Burst Arrivals (RC)

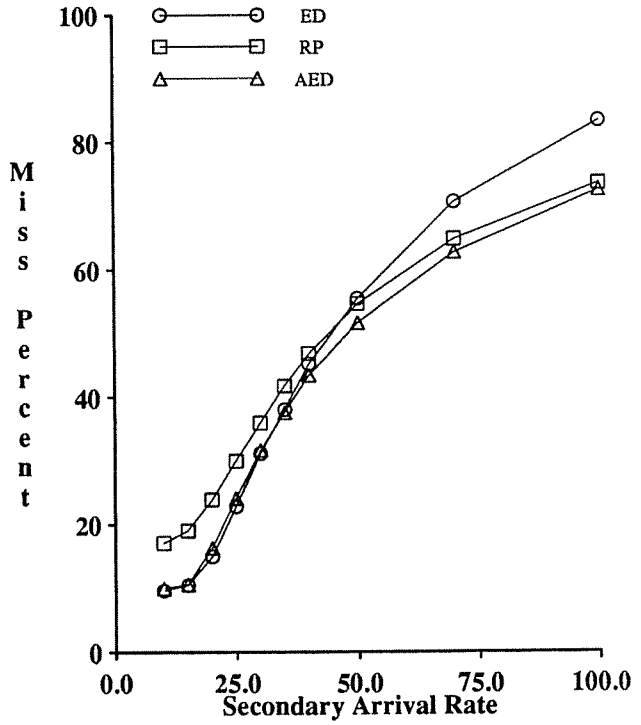

Figure 6: Burst Arrivals (RC+DC)

similar to those seen for the fixed arrival rate experiments. This implies that the algorithm is robust towards fluctuations in the transaction workload pattern.

7. EXTENDING AED FOR TRANSACTION VALUES

In a real-time database system, an application may assign a value to a transaction to reflect the return it expects to receive if the transaction commits before its deadline. In the discussion thus far, we implicitly assumed that all transactions have the same value, and therefore the performance goal was to minimize the number of missed deadlines. There are certainly real-time applications, however, where different transactions may be assigned different values [Huan89]. In this framework, the goal is to maximize the sum of the values of those transactions that commit by their deadline. Minimizing the number of missed deadlines becomes a secondary concern in such systems.

A fundamental problem when transactions are characterized by both value and deadline is how to construct a priority ordering. Developing a priority ordering 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. In the Earliest Deadline mapping, transactions with earlier deadlines are given higher priority 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 addition to the evaluation of fixed-tradeoff priority mappings, a "bucket" priority mechanism that allows the relative importance of values and deadlines to be varied was introduced in [Hari91a]. The actual tradeoff made between values and deadlines is controlled by a parameter of the algorithm and it was demonstrated that the algorithm could perform well at each operating point when this control parameter is set appropriately. There are some limitations, however, of this algorithm: First, the tradeoff between values and deadlines can be made only in fixed proportions since the control parameter is constrained to be an integer number. Second, an open problem is how the system might adaptively change the setting of the control parameter to optimize performance as the system load varies. In the remainder of this section, we describe a priority mechanism that, while being based on similar principles as that of the original bucket algorithm, improves on it by addressing these limitations.

7.1. Hierarchical Earliest Deadline (HED)

In this section, we present an extended version of the AED policy called Hierarchical Earliest Deadline (HED) that takes into account transaction values and is geared towards maximizing the realized value. Informally, the HED policy groups transactions, based on their value, into a hierarchy of prioritized buckets, and then uses the AED policy within each bucket to determine the relative priority of transactions belonging to that bucket. The details of the HED policy are described below, after which the rationale behind the construction of the algorithm is discussed.

The HED algorithm functions in the following manner: The *priority mapper* unit maintains a *value-based* dynamic list of buckets, as shown in Figure 7. Every incoming transaction is assigned at arrival time, based on its value, to a particular bucket in this list. Each bucket in the list has an associated *MinValue* and *MaxValue* attribute – these attributes set the bounds on the values of transactions that may be assigned to the bucket. There are two special buckets, *Top* and *Bottom*, which 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.

When a new transaction T_{new} arrives in the system, it is assigned to the bucket closest to *Top* that satisfies the constraint $MinValue \leq Value_{new} \leq MaxValue$. If no such bucket exists, then a new bucket is inserted in the list between the bucket closest to *Top* that satisfies $MinValue \leq Value_{new}$ and its predecessor, and the transaction is assigned to this bucket. The identifier of a newly created bucket is established by halving the sum of the identifiers

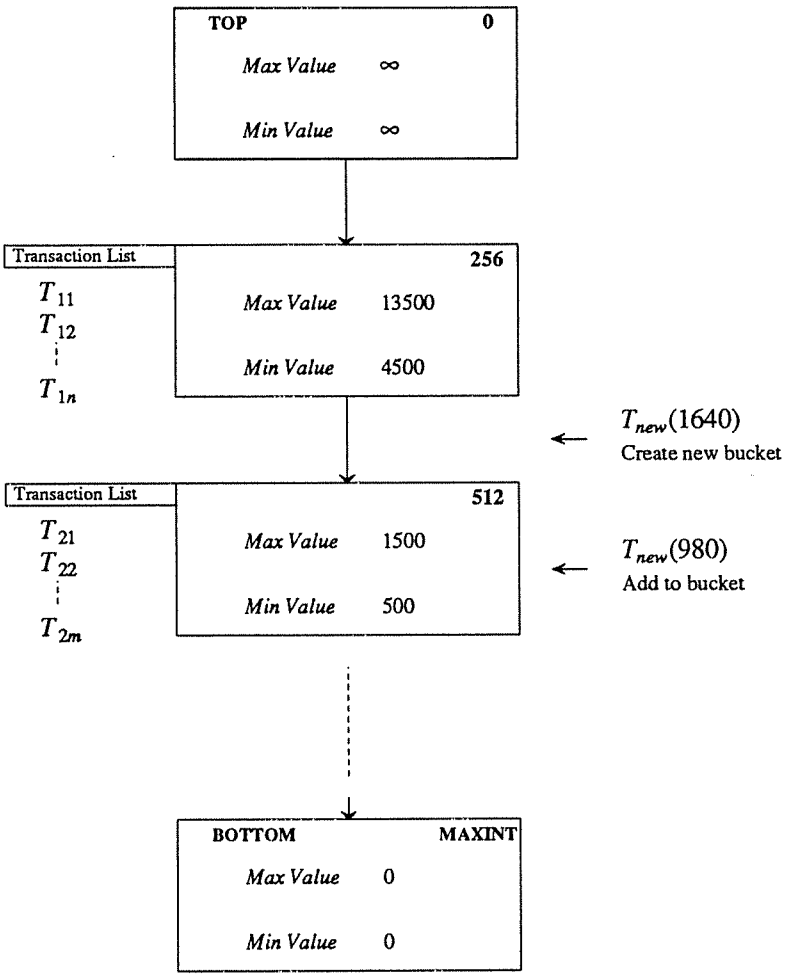

Figure 7: HED Bucket Hierarchy

of the predecessor and successor bucket. For example, the identifier of a new bucket inserted between buckets 256 and 512 will be 384. The identifiers of the *Top* and *Bottom* buckets are preset to 0 and MAXINT, respectively. When a transaction leaves the system, it is removed from its assigned bucket. If a bucket has no transactions currently assigned to it, the bucket is deleted from the list.

The transaction priority assignment is then made using the formula:

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

where B_T is the identifier of the transaction's bucket, and V_T is the transaction's value.

The above priority assignment method 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, transaction priorities are assigned in similar fashion to that of the AED mapping. There are, however, a few important differences that should be noted: First, the transaction list within a bucket is ordered based on transaction *values*, instead of on transaction keys. Second, the priority ordering within the *MISS* group is Highest Value instead of Random Priority. Third, the I_T component serves to break the tie for transactions in the *HIT* group that have the same deadline and for transactions in the *MISS* group that have the same value. This ensures a *complete* priority ordering of all the transactions in the system.

The method of setting the *MinValue* and *MaxValue* attributes of a bucket is 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 are then computed as (*AvgValue* / *SpreadFactor*) and (*AvgValue* * *SpreadFactor*), respectively, where *SpreadFactor* is a parameter of the bucket algorithm. The *SpreadFactor* controls the maximum spread of values allowed within a bucket. Whenever a transaction enters or leaves the system, the bucket to which it is assigned updates its *AvgValue*, *MinValue* and *MaxValue* attributes.

7.2. 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 policy ensures that transactions having substantially different values are not assigned to the same bucket. The reason for doing this is the following: The AED policy 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 these high-valued transactions.

8. 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 are taken to represent system performance under "normal" loadings, while Loss Percent values in the range of 20 to 100 percent represent system performance under "heavy" loading.

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 parameters and resource parameters are the same as those listed in Section 6.4. The *SpreadFactor* parameter of the HED algorithm is set to 3 in the experiments described here.[12]

8.1. Uniform Value Distribution

Our first experiment investigated the performance of the priority mappings for the Uniform transaction value workload. For this experiment, Figures 8a and 8b 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.

---

[12] The choice of *SpreadFactor* is constrained by two opposing considerations: Having a high *SpreadFactor* results in high-valued transactions being grouped with low-valued transactions. On the other hand, having too low a *SpreadFactor* causes the policy to give undue importance to value over deadline. Extensive experimentation with several different value distributions showed that a setting of 3 delivers a reasonable tradeoff between these conflicting considerations. A setting of exactly 3 is not *critical* to the performance of the policy, however; settings between 2 and 4 performed equally well.

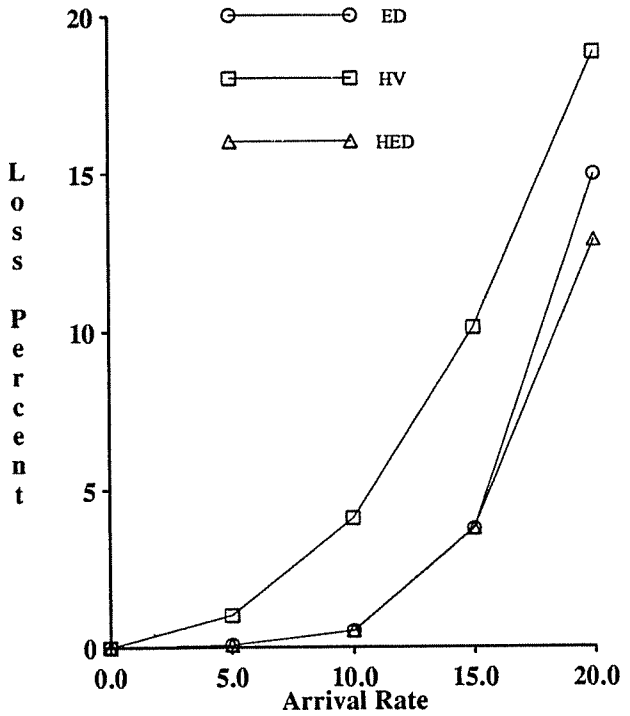

Figure 8a: Uniform Values (Normal Load)

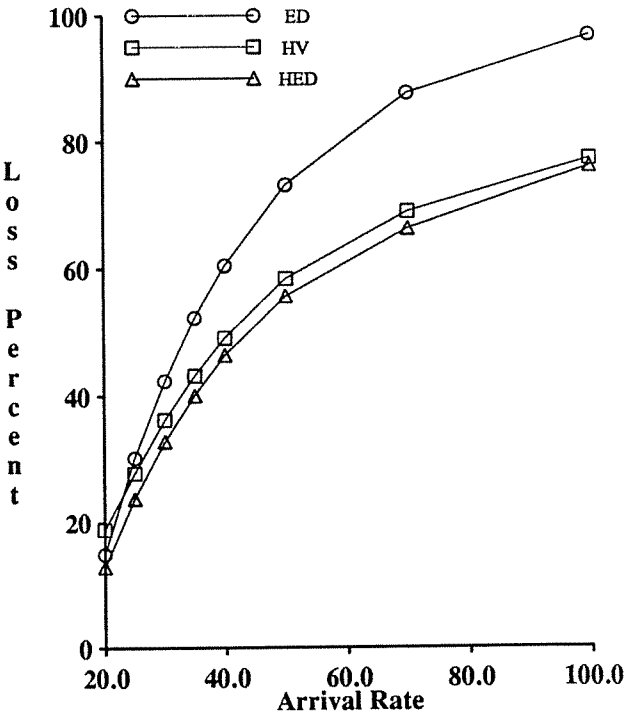

Figure 8b: Uniform Values (Heavy Load)

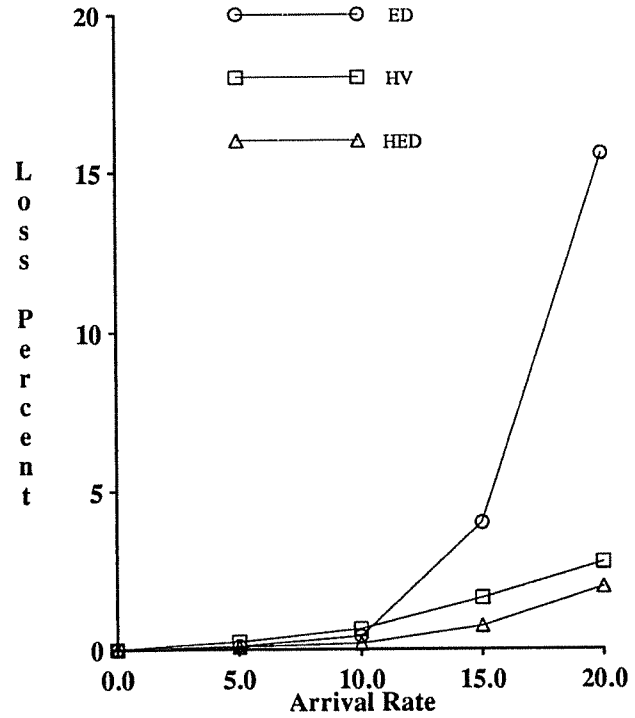

Figure 9a: Skewed Values (Normal Load)

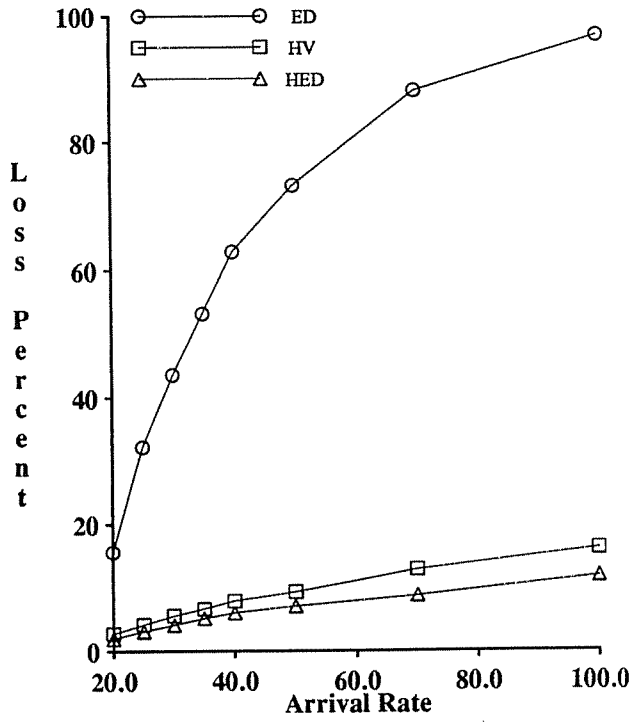

Figure 9b: Skewed Values (Heavy Load)

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 distribution workload examined in this section: First, the HED 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.

8.2. 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 9a and 9b. 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 8a and 8b). 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.

9. 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 semantics is not available. We introduced the Adaptive Earliest Deadline (AED) priority policy and, using a detailed simulation model of a real-time database system, studied its performance relative to Earliest Deadline and other fixed priority mappings. The performance metric underlying our simulation experiments was the number of missed transaction deadlines, and the experiments covered a range of workload characteristics and system operating conditions. In particular, the performance impacts of both resource contention and data contention were considered. Workloads that had bursts in the transaction arrival pattern were

also considered.

Our experiments showed that for workloads generating only resource contention, the AED priority policy delivered the best performance over the entire loading range. At light loads, it behaved exactly like Earliest Deadline; at high loads its behavior was better than 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 eventual completion status (in-time or late) of a transaction at arrival time. AED's policy of restricting the use of the Earliest Deadline approach to the *HIT* group delivered stabilized performance at high loads. For workloads that generated both data and resource contention, the AED policy delivered the best overall performance. At low loads AED performed similarly to Earliest Deadline, and at high loads it changed its behavior to follow that of Random Priority. The AED policy was also shown to be robust to fluctuations in the transaction arrival pattern.

In all of the experiments reported here, a locking algorithm was used as the concurrency control mechanism. For resource-constrained systems, we have observed similar behavior of the various policies with optimistic algorithms. In the absence of resource contention, however, the Earliest Deadline mapping is inherently stable if an optimistic algorithm is used to provide concurrency control [Hari91b]. Thus, the pure ED mapping provides the best performance for this particular scenario (which is essentially the so-called "infinite resources" scenario considered in some concurrency control performance studies).

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. In this paper, we introduced the Hierarchical Earliest Deadline (HED) policy to address this issue. The HED policy groups transactions, based on their value, into a hierarchy of prioritized buckets; it then uses the AED policy within each bucket. Using our RTDBS simulation model, we evaluated the performance of the HED priority policy with respect to mappings that establish fixed tradeoffs between values and deadlines. The performance metric underlying this set of simulation experiments was the total value provided by transactions that complete before their deadlines. Workloads with different degrees of spread and skew in transaction value distribution were considered.

Our experiments showed that, both for workloads with limited spread in transaction values and for workloads with pronounced skew in transaction values, the HED policy 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 policy 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 policy increased the realized value. For workloads with pronounced skew in transaction values, the hierarchical nature of the HED policy was effective in ensuring that "golden" (high-valued) transactions were completed and their value realized.

REFERENCES

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

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

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

[Buch89] Buchmann, A., McCarthy, D., Hsu, M., and Dayal, U., "Time-Critical Database Scheduling: A Framework for Integrating Real-Time Scheduling and Concurrency Control," *Proc. of the 5th Int. Conference on Data Engineering*, Feb. 1989.

[Dert74] Dertouzos, M., "Control Robotics: the procedural control of physical processes," *Proc. of the IFIP Congress*, 1974.

[Eswa76] Eswaran, K., Gray, J., Lorie, R., and Traiger, I., "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM*, Nov. 1976.

[Gray79] Gray, J., "Notes on Database Operating Systems," in *Operating Systems: An Advanced Course*, Springer-Verlag, 1979.

[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 the 1990 IEEE Real-Time Systems Symposium*, December 1990.

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

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

[Huan89] Huang, J., Stankovic, J., Towsley, D., and Ramamritham, K., "Experimental Evaluation of Real-Time Transaction Processing," *Proc. 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. IEEE Real-Time Systems Symposium*, Dec. 1985.

[Kung81] Kung, H., and Robinson, J., "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems*, June 1981.

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

[Livn88] Livny, M., *DeNet User's Guide*, Version 1.0, Comp. Sci. Dept., Univ. of Wisconsin, Madison, 1988.

[Lock86] Locke, C., "Best Effort Decision Making for Real-Time Scheduling," *Ph.D. Thesis*, Dept. of Computer Science, Carnegie-Mellon University, May 1986.

[Mok78] Mok, A. and Dertouzos, M., "Multi-processor Scheduling in a Hard Real-Time Environment," *Proc. of the Seventh Texas Conference on Computing Systems*, Oct. 1978.

[Pan88] Panwar, S. and Towsley, D., "On the Optimality of the STE Rule for Multiple Server Queues that Serve Customers with Deadlines," *COINS Technical Report 88-81*, Univ. of Massachusetts, Amherst, July 1988.

[Reed78] Reed, D., "Naming and Synchronization in a Decentralized Computer System," *Ph.D. Thesis*, Dept. of Computer Science, Massachusetts Institute of Technology, 1978.

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