VALUE-BASED SCHEDULING IN REAL-TIME DATABASE SYSTEMS

by

Jayant R. Haritsa
Michael J. Carey
Miron Livny

Computer Sciences Technical Report #1024

May 1991

Value-Based Scheduling in Real-Time Database Systems

Jayant R. Haritsa
Michael J. Carey
Miron Livny

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

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

Value-Based Scheduling in Real-Time Database Systems

Jayant R. Haritsa
Michael J. Carey
Miron Livny

Computer Sciences Department
University of Wisconsin
Madison, WI 53706

ABSTRACT

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. Most prior research on real-time database systems has focused on systems where all transactions are assigned the same value, with the performance goal being to minimize the number of missed deadlines. When transactions may be assigned different values, the goal of the system shifts to maximizing the sum of the values of those transactions that commit by their deadlines. Minimizing the number of missed deadlines becomes a secondary concern in such systems.

In this paper, we address the problem of establishing a priority ordering among transactions characterized by both values and deadlines that results in maximizing the realized value. Of particular interest is the tradeoff that needs to be established between these values and deadlines in constructing the priority ordering. Using a detailed simulation model, we evaluate the performance of several priority mappings that make this tradeoff in different, but fixed, ways. In addition, a "bucket" priority mechanism that allows the relative importance of values and deadlines to be controlled is introduced and studied. The notion of associating a penalty with transactions whose deadlines are not met is also briefly considered.

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].

In a real-time database system, an application may assign a *value* to a transaction to reflect the return the application expects to receive if the transaction completes *before* its deadline [Huan89].[1] The sum of the values of all transactions submitted to the RTDBS constitutes the *offered value*, while the sum of the values of the transactions that are completed before their deadlines constitutes the *realized value*. The goal of a real-time database management system is to maximize the realized value, since this metric is a direct measure of the real-time support provided to the application [Jens85]. Most of the prior research on the performance of real-time database systems (e.g. [Abbo88, Hari90a]) has focused on applications where all transactions are assigned the *same* value. For such a framework, the performance goal of maximizing the realized value is equivalent to minimizing the number of missed deadlines. The concern in this environment is how *many* transactions are missed, and not *which* transactions are missed. This is because when all transactions have the same value, missing the deadline of one or the other makes no difference to the total realized value. There are certainly real-time applications, however, where different transactions may be assigned *different* values [Stan88, Huan89]. The value realized by a database system supporting such applications depends on which transactions meet their deadlines.

To make the notion of transactions having different values clear, consider an airline reservation system that allows customers to call in their reservations. The time constraint on each reservation transaction is the delay that the customer is willing to endure before hanging up. Satisfying the request of a customer buying a high-priced ticket is more beneficial to the airline than satisfying the request of a customer buying a cheaper ticket since the high-priced ticket generates greater revenue. In this scenario, therefore, the value of a transaction is the fare paid by the requesting customer, and the objective of the reservation database system is to maximize the revenue received by the airline. A key point to note here is that value and deadline are fundamentally different properties [Biya88, Huan89]. The fact that a transaction has a tight deadline does not necessarily mean that it has a high value, nor does a loose deadline imply a low value. Transactions with similar values may have different deadlines, while those with similar deadlines may have different values. The value reflects the transaction's *worth* while the deadline reflects the transaction's *urgency*.

In this paper, we address the issue of resource scheduling and concurrency control in a real-time database system where transactions may differ in their assigned values. We assume that the value of a transaction is obtained by the application if the database system completes the transaction before its deadline. If the deadline is missed,

[1] In certain applications, there may be some diminished value to completing a transaction even after its deadline. For the sake of simplicity, we consider in this paper only transactions that have zero value after their deadline.

however, no value is realized. This time-coupling between transaction value and system-realized value can be naturally expressed by the use of *value functions*, a powerful mechanism for expressing time constraints that was developed in the seminal work of [Jens85, Lock86]. The key idea of the value function concept is that the completion of a task has a value to the application that can be expressed as a function of the completion time. Our model, for instance, is captured by the value function shown in Figure 1a. The figure shows a transaction T that has an arrival time A_T, a deadline D_T, and a value V_T in the interval (A_T, D_T). The application receives V_T if transaction T is completed before D_T, and zero otherwise.

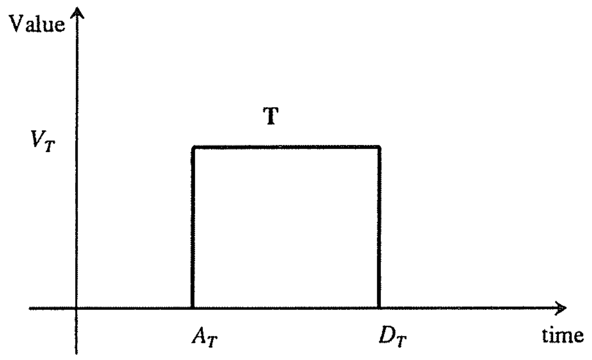

Figure 1a: Transaction Value Function

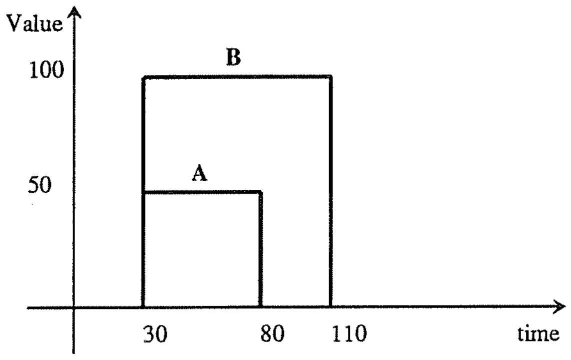

Figure 1b: Priority Order Dilemma

In order to resolve contention for hardware resources and data objects, the RTDBS needs to establish a priority ordering among the transactions executing in the system. The priority ordering should reflect the objective of maximizing the total realized value. In the absence of detailed knowledge of transaction resource requirements and data access semantics, two basic principles, Earliest Deadline and Highest Value, can be used to guide such a priority ordering. The Earliest Deadline principle is that transactions with closer deadlines should be run first since delaying them might cause their deadlines to be missed and result in their value being lost. The Highest Value principle is that transactions with higher values should be run first since it would be beneficial to make certain that their deadlines are met and thereby realize their high values.

When the conflicting transactions have similar deadlines, it would appear, from an intuitive standpoint, that the ordering established by the Highest Value principle is the right ordering. This is because if transactions are of similar urgency, completing the more valuable transactions first ensures that more value is realized. Conversely, when the conflicting transactions have similar values, it would seem that the Earliest Deadline principle provides the right priority ordering. This is because if transactions are of similar utility, completing the more urgent transactions first should result in more realized value. When transactions differ in both their value and deadline characteristics, however, it is not obvious which principle should be followed. To provide a simple example, consider the scenario where a pair of transactions, A and B, compete for service. Assume that V_A=50 and D_A=80, while V_B=100 and D_B=110, as shown in Figure 1b. In this case, following the Earliest Deadline principle would yield the priority ordering (A, B), while following the Highest Value principle would yield the priority ordering (B, A). It is not clear which of these priority orderings would realize more value. The dilemma here is to decide whether the value difference of 50 between A and B is more important or less important than their deadline difference of 30 time units. In a more general sense, deciding on a priority ordering when transactions differ in both value and deadline requires the value and deadline characteristics to be *weighted* in some fashion. A succinct statement of this requirement is that a priority mapping has to be established from the pair (D_T, V_T) to P_T, where P_T denotes the priority of transaction T. For example, a possible priority mapping, which gives equal weight to value and deadline, is[2] $P_T = \frac{D_T}{V_T}$. In the example mentioned above, this mapping would result in the priority ordering (B, A).

In this paper, we use a detailed simulation model of a real-time database system to evaluate the impact of various priority mappings on the value realized by the system. These mappings were implemented and studied in the real-time testbed RT-CARAT [Huan89], and create *fixed* tradeoffs between value and deadline. Those studies profiled the performance of the mappings for workloads where the transaction value distribution was both uniform and limited in its range of values. In our study, we expand on this initial work by investigating the performance effects of having different degrees of spread in transaction values and high degrees of skew in the value distribution. The impacts of resource contention and data contention are examined in isolation and in combination, and the effects of correlation in transaction workload characteristics are discussed. In addition, a "bucket" based priority mapping that integrates value and deadline based on fundamental real-time scheduling principles is introduced and evaluated in this paper. The bucket mapping allows the relative importance of values and deadlines to be *varied.* The notion of associating a penalty with transactions whose deadlines are not met is also briefly considered.

Our simulation model captures the modular architecture shown in Figure 2. The *priority mapper* unit generates a priority for a transaction on its arrival, and this priority is subsequently used throughout the system. It is possible that there may be feedback in the priority assignment process, causing the priority of a transaction to change with time; in this case, the change in priority is transmitted by the priority mapper to the transaction, thus shielding the internal database mechanisms from the details of the priority generation process. This design is modular since it allows *priority generation* to be separated from *priority usage.*

We restrict our attention in this study to the case where all transactions have step-shaped value functions of the type shown in Figure 1a. This means that transactions that miss their deadline can be immediately discarded, since completing them late generates no value to the application. In such situations, we say that the system is operating under *firm* deadlines. This is to be distinguished from *hard* deadlines, where catastrophic results may occur if a deadline is missed, and from *soft* deadlines, where even late transactions retain some completion value. A second assumption that we make is that the database system has no a-priori knowledge of transaction hardware resource requirements or data access semantics, since such information is not available in most cases.

The remainder of this paper is organized in the following fashion: Section 2 reviews related work on real-time database systems. Section 3 contains a description of several different priority mappings that have been presented in the real-time literature, and also describes the "bucket" mechanism. Section 4 describes the functioning of the concurrency control algorithms evaluated in the study. Then, in Section 5, we describe our RTDBS model and its parameters, while Section 6 highlights the results of the simulation experiments. Finally, Section 7 summarizes the main conclusions of the study and outlines future avenues to explore.

[2] Smaller P_T values indicate higher system priority.

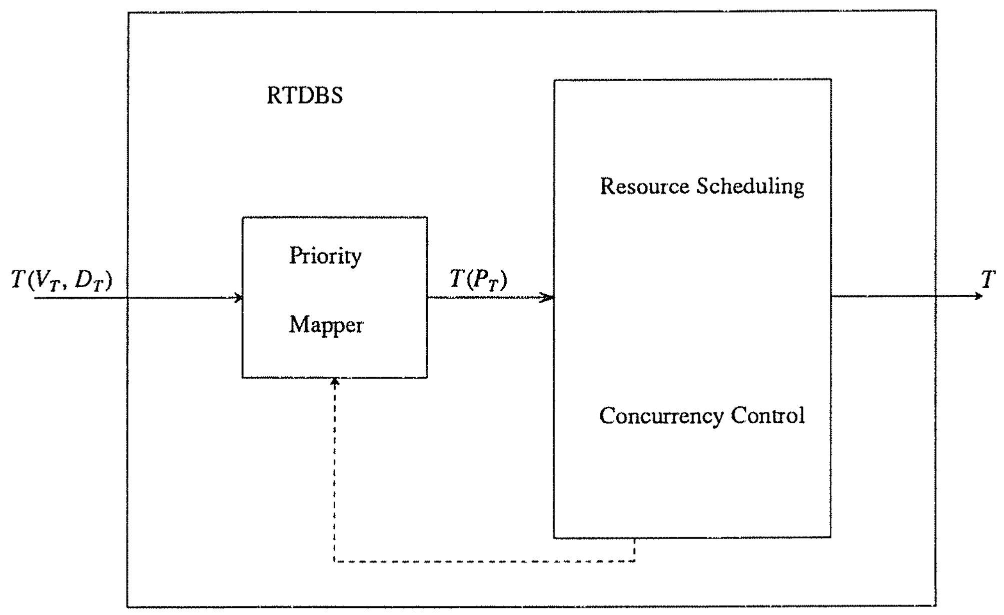

Figure 2: Priority Architecture

2. RELATED WORK

The first study of the role of value in real-time systems was presented in the seminal work of [Jens85]. A well-constructed "Best Effort" priority mapping mechanism for integrating value and deadline was developed and shown to outperform several "classical" priority mappings. Similar schemes for integrating value and deadline in distributed real-time systems were described in [Biya88]. In order to use these schemes, which were developed in the context of task scheduling in real-time operating systems, a-priori knowledge of task service requirements is required. Unfortunately, knowledge about transaction resource and data requirements is usually unavailable in database applications [Stan88], and the schemes therefore cannot be used in most real-time database systems.

The last few years have seen quite a few studies published on the performance of resource scheduling policies and concurrency control algorithms in the context of real-time database systems. All these studies consider RTDBSs that operate under either *firm* or *soft* deadlines.[3] The studies can be divided into two general groups – those that treat all transactions as being equally important [Abbo88, Abbo89, Abbo90, Hari90a, Hari90b, Huan90a], and those that incorporate the notion of transactions having different values [Huan89, Huan90b]. A brief summary of the studies in these two groups is presented in the remainder of this section.

The problem of scheduling transactions in a RTDBS was first addressed by Abbott and Garcia-Molina [Abbo88, Abbo89]. Their work focused on evaluating the performance of various real-time scheduling policies, all of which policies enforced data consistency by using a two-phase locking protocol as the underlying concurrency control mechanism. In [Hari90a], the focus was shifted to studying the performance of optimistic and pessimistic

[3]It is generally considered that *hard* deadline RTDBSs are infeasible since it is difficult to determine a-priori the computation time and execution pattern of a transaction [Abbo88, Stan88].

methods of concurrency control in a real-time environment. This work was extended in [Hari90b] with the development of new optimistic algorithms that delivered improved performance. Algorithms for buffer allocation and buffer replacement in real-time database systems were proposed and evaluated in [Huan90a]. In [Abbo90], a study of algorithms for scheduling disk requests with deadlines was made. Each of these studies assumed that all transactions have the same value, and the primary performance metric was therefore the number of missed deadlines.

The only studies so far to have incorporated transaction value in their performance evaluation framework are [Huan89, Huan90b]. Using a basic locking scheme for concurrency control, [Huan89] investigated several algorithms for resource scheduling and data conflict resolution. In [Huan90b], this work was extended to include optimistic methods of concurrency control. These studies were conducted on a real-time database testbed (RT-CARAT), and form an important first step in understanding the effect of multiple transaction values on RTDBS performance. There are some aspects of these studies, however, which leave room for further investigation: First, the range of values that transactions could take on was limited and the value distribution was uniform. Second, the concurrency control algorithms that were compared in [Huan90b] are priority-indifferent flavors of two-phase locking and optimistic concurrency control. Finally, testbed limitations constrained the study to using a closed queueing system with a fixed amount of resources.

Our work differs from [Huan89, Huan90b] in that we consider a variety of transaction workloads that have different degrees of spread and skew in transaction value. Also, an open system with different levels of resource availability is modeled. In addition, prioritized flavors of locking and optimistic algorithms are implemented and compared. Lastly, a mechanism that allows the tradeoff between value and deadline to be varied is presented and evaluated.

3. PRIORITY ASSIGNMENT ALGORITHMS

In order to resolve contention for hardware resources and data, the RTDBS has to establish a priority ordering among the transactions. The ordering should reflect the goal of maximizing the realized value. When transactions may be distinguished by both value and deadline, the priority mapping has to take both these characteristics into account. In earlier studies [Stan88, Huan89, Huan90b], several priority mappings that combine the Earliest Deadline and Highest Value principles with different tradeoffs between value and deadline have been examined. A representative subset of these mappings that cover a range of value/deadline tradeoffs are evaluated in our performance study. This subset is described first in this section. Subsequently, a new "bucket" mechanism for the integration of value and deadline is presented. In the following discussion, A_T, D_T, V_T and P_T will be used to denote the arrival time, deadline, value, and priority, respectively, of transaction T. The priority assignments of all of the mappings are such that smaller P_T values reflect higher system priority. The first two mappings presented below implement extreme tradeoffs between value and deadline, while the others implement intermediate tradeoffs.

3.1. Earliest Deadline (ED)

The Earliest Deadline mapping follows the Earliest Deadline principle, and the transaction priority assignment is $P_T = D_T$. It represents an extreme tradeoff as the value of the transaction is not taken into consideration. Several studies (e.g. [Jens85, Abbo88]) have observed that in lightly-loaded or moderately-loaded real-time systems, using an Earliest Deadline schedule results in the fewest missed deadlines. This mapping is generally used as the scheduling policy in real-time systems where all tasks have the same value and details of task characteristics are not available.

3.2. Highest Value (HV)

The Highest Value mapping follows the Highest Value principle and the transaction priority assignment is $P_T = \frac{1}{V_T}$. It represents the other extreme tradeoff as the deadline of the transaction is not taken into consideration. Note that this mapping does not distinguish between transactions that have the same value in terms of the priority assigned to them. Therefore, if all transaction values are the same, this mapping is equivalent to having *no* priority in the system.

3.3. Value-inflated Deadline (VD)

The Value-inflated Deadline mapping combines the Earliest Deadline and Highest Value principles by using the transaction priority assignment $P_T = \frac{D_T}{V_T}$. It gives *equal* weight to deadline and value. Moreover, within a group of transactions that have the same value, the priority ordering established by this mapping is identical to that of the ED mapping; within a group of transactions that have the same deadline, the priority ordering established is identical to that of the HV mapping.

3.4. Value-inflated Relative Deadline (VRD)

The Value-inflated Relative Deadline mapping is similar in flavor to VD, but it uses the *relative* deadline, instead of the absolute deadline, in combining the Earliest Deadline and Highest Value principles. The transaction priority assignment is $P_T = \frac{D_T - A_T}{V_T}$. It gives equal weight to *relative deadline* and value. Note that if all transactions have their deadlines at a fixed distance from their arrival times (i.e. $D_T - A_T = constant$), this mapping produces a priority ordering identical to that established by the HV mapping.

3.5. Bucket Algorithm (BA)

The "bucket" algorithm takes the approach of combining value and deadline based on the application of two key scheduling observations:

(1) 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.[4]

(2) Given a set of similar tasks and a system that has resources sufficient only to meet the deadlines of a subset of these tasks, choosing the highest valued tasks to form the subset results in maximizing the realized value.

From these scheduling characteristics, the implication 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

[4] For uniprocessor systems, Earliest Deadline is guaranteed to meet *all* of the task deadlines [Dert74].

completed by their deadlines. Also, we would like this set of feasible transactions to be composed of the highest valued transactions. The bucket mechanism tries to capture these two objectives in its combination of value and deadline. The mechanism is implemented as follows: The *priority mapper* unit of Figure 2 maintains a *value-ordered* list of all the transactions currently in the system. When a new transaction T arrives in the system, it is inserted into the list and its position in the list, pos_T, is noted. The transaction is assigned, based on its position, to one of the buckets in an array of *NumBuckets* buckets, where *NumBuckets* is a parameter of the mechanism. The bucket assignment is done using the formula

$$B_T = \begin{cases} \left\lceil \dfrac{pos_T * NumBuckets}{NumTrans} \right\rceil & \text{if } NumTrans > NumBuckets \\ pos_T & \textit{otherwise} \end{cases} .$$

where *NumTrans* is the number of transactions currently in the system. The priority assignment[5] for transaction T is then computed as

$$P_T = (B_T, D_T, I_T)$$

where the I_T component is a randomly chosen[6] integer key. The key is intended to serve as a "noise factor" and establish a priority ordering among transactions of the same bucket that may have *identical* deadlines, thus ensuring a complete ordering among all the transactions in the system.

The physical meaning of this algorithm is that the transactions in the system are evenly split up into a set of buckets. The splitting is based on transaction value and transactions of bucket i tend to have lower value than transactions of bucket $i-1$ and higher value than transactions of bucket $i+1$. The corresponding priority assignment is such that transactions of bucket i have lower priority than transactions of bucket $i-1$ and higher priority than transactions of bucket $i+1$. The priority assignment is also arranged such that *within* each bucket, the transaction priority ordering is based on the Earliest Deadline principle.

By using an earliest deadline priority ordering within a bucket, the algorithm tries to incorporate the first scheduling characteristic described above. By splitting the transactions into buckets based on value, the bucket mechanism tries to incorporate the second scheduling characteristic.

3.6. Tradeoffs

By analyzing the priority mappings described above, we notice several interesting features. For instance, if transaction relative deadlines are linearly correlated to their execution times, the VRD mapping gives priority to transactions that can deliver the most value for the smallest amount of resource consumption. In this scenario, if transactions also all have the same value, the VRD mapping establishes a Shortest Job First priority ordering. For

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

[6] 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.

these cases, therefore, the VRD mapping behaves like a simple "greedy" algorithm that tries to maximize short-term benefits without taking transaction time constraints into account.

Turning our attention to the bucket algorithm, we see that if the *NumBuckets* parameter is set to 1, the priority mapping is identical to the Earliest Deadline mapping. This is because all transactions are assigned to the same bucket and the priority ordering within a bucket is Earliest Deadline. If the *NumBuckets* parameter is set to ∞, the priority mapping is identical to the Highest Value mapping, since each transaction is assigned to a different bucket and the buckets are ordered by value. A *NumBuckets* setting between these two extremes establishes intermediate tradeoffs between value and deadline. Therefore, this parameter provides a mechanism for adjusting the tradeoff between value and deadline to the desired level.

4. CONCURRENCY CONTROL ALGORITHMS

The resource scheduling policies used in most studies of real-time database systems (e.g. [Abbo88, Abbo89]) are preemptive 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]). In this section, we describe the concurrency control algorithms that were chosen for evaluation in this study. These algorithms are a subset of those that were investigated in our earlier studies on the performance of concurrency control algorithms in a RTDBS environment [Hari90a, Hari90b]. The selected algorithms are 2PL-HP, OPT-BC and OPT-WAIT. 2PL-HP is a prioritized locking algorithm, OPT-BC is a priority-indifferent (conventional) optimistic algorithm, and OPT-WAIT is a prioritized variant of the OPT-BC algorithm. The details of these algorithms are given below.

4.1. 2PL-HP

In 2PL-HP, classical two-phase locking [Eswa76] is augmented with a *High Priority* [Abbo88] conflict resolution scheme to ensure that high priority transactions are not delayed by low priority transactions. This scheme resolves 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 scheme also serves as a deadlock prevention mechanism.[7]

4.2. OPT-BC

In OPT-BC, classical optimistic concurrency control [Kung81] is modified to implement the notion of a *Broadcast Commit* [Mena82, Robi82]. Here, when a transaction commits, it notifies other executing transactions that conflict with it and these transactions are immediately restarted. A validating transaction conflicts with an

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

executing transaction if the validating transaction wishes to update a data object that has been read by the executing transaction. Note that there is no need for a validating transaction to check for conflicts with any already-committed transactions since any such transaction would have, in the event of a conflict, already restarted the validating transaction at its (the committed transaction's) own earlier commit time. This also means that a validating transaction is always certain to commit. The broadcast commit method detects conflicts earlier than the basic optimistic algorithm of [Kung81], resulting in less wasted resources and earlier restarts; this increases the chances of meeting transaction deadlines. An important point to note is that transaction priorities are *not* used in resolving data conflicts.

4.3. OPT-WAIT

The OPT-WAIT algorithm [Hari90b] is a variant of the OPT-BC algorithm that incorporates transaction priorities. It features a *priority wait* mechanism: a transaction that reaches validation and finds higher priority transactions in its set of conflicting transactions is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. While a transaction is waiting, it may be restarted due to the commit of one of the conflicting higher priority transactions. If the deadline of the waiter is reached during the waiting process, and higher priority transactions still exist in the conflicting set, then the waiter is aborted and discarded. OPT-WAIT and OPT-BC represent the extremes with regard to waiting – OPT-WAIT always waits for a higher priority transaction, while OPT-BC never waits and unilaterally commits the validating transaction.

We include the OPT-BC algorithm in this study, although it is a priority-indifferent algorithm, for the following reason: In [Hari90a], it was shown that OPT-BC, in spite of being priority-indifferent, provided better performance than 2PL-HP in a firm deadline environment. That study assumed that all transactions have the same value. In this study, we wish to find out whether the above results also carry over to real-time database systems that operate with transactions that have different values.

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]. In this model, the database system consists of a shared-memory multiprocessor operating on disk resident data.[8] The database itself is modeled as a collection of pages. Transactions arrive in a Poisson stream and each transaction has an associated value and 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.[9] A transaction that is restarted 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

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

[9] 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.

basic structure of the model is shown in Figure 3.

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.

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 that a page which is read will also be updated.

5.1.1. Transaction Value Assignment

The arrival stream of transactions is composed of multiple transaction classes that are distinguished by their value distribution. The number of classes is specified by the *NumClasses* workload parameter. The average value of a transaction over all classes is specified by the *GlobalMeanValue* parameter. Each transaction class is characterized by four workload parameters: *ProbClass*, *OfferedValue*, *MeanValue*, and *SprdValue*. The *ProbClass* parameter specifies what fraction of the input workload is formed of transactions belonging to the class. *OfferedValue* is the fraction of the total offered value to the system that is contributed by transactions of the class. For example, a setting of *ProbClass* = 0.2 and *OfferedValue* = 0.8 captures a "20-80" class which constitutes 20 percent of the input

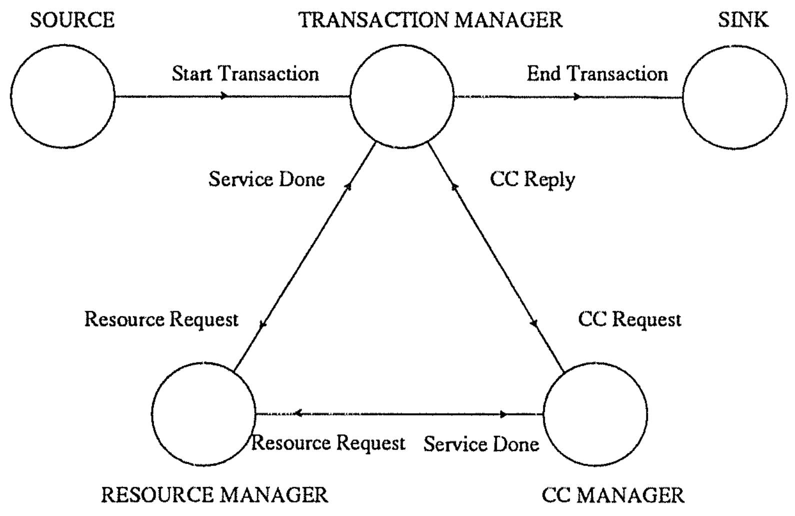

Figure 3: RTDBS Model Structure

Parameter	Meaning
ArrivalRate	Transaction arrival rate
DatabaseSize	Number of pages in database
PageCount	Avg. number of pages accessed/transaction
WriteProb	Write probability/accessed page
DeadlineFormula	DF1 or DF2
LSF	Low Slack Factor
HSF	High Slack Factor
GlobalMeanValue	Mean transaction value
NumClasses	Number of transaction classes
ProbClass [*i*]	Prob. of class *i*, *i* = 1,2,..,NumClasses
OfferedValue [*i*]	Fraction value offered by class *i*
MeanValue [*i*]	(computed) Mean value of class
SprdValue [*i*]	Percent Spread in value of class *i*

Table 1: Workload Model Parameters

transactions and accounts for 80 percent of the total offered value. The average value of transactions belonging to a class, *MeanValue*, is set by the formula $\frac{OfferedValue}{ProbClass} * GlobalMeanValue$. Therefore, for the 20-80 example, and assuming $GlobalMeanValue = 100.0$, the average value of transactions of the 20-80 class would be 400.0. The *SprdValue* parameter bounds the range of values that transactions of a class can take, and it is specified as a percentage of the *MeanValue* of the class. For the above example, a setting of 50 for this parameter would specify that the range of values would be between ± 50 percent of 400.0, that is, between 200.0 and 600.0 . The actual transaction values in the class are generated from a uniform distribution over the range established by the *MeanValue* and *SprdValue* parameters.

5.1.2. Transaction Deadline Assignment

The *DeadlineFormula* workload parameter determines how transaction deadlines are assigned. Two transaction deadline formulas are employed in this study. The first formula, DF1, which is used for most of the experiments reported here, is:

$$D_T = A_T + SF_T * R_{\max} \qquad \text{(DF1)}$$

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.

The second deadline formula, DF2, used in the study is:

$$D_T = A_T + SF * R_T \qquad \text{(DF2)}$$

Here the actual resource time of transaction *T* is used to compute its deadline, i.e. R_T replaces $R_{\max}$ in the assignment. Also, the slack factor *SF* does not vary over a range but is a constant. For this formula, the workload parameters *LSF* and *HSF* are set to the same number and *SF* takes on this value.

The first formula, DF1, makes the deadline of a transaction independent of its actual execution time, and is designed to represent workloads where there is no correlation between a transaction's deadline and its execution time. Deadline formula DF2 is designed to investigate the effects of correlation, and makes the deadline of a transaction *linearly* correlated to its execution time. Note that with DF2, all transactions have the same *slack ratio*, where slack ratio is defined to be the ratio $\frac{D_T - A_T}{R_T}$. With DF1, however, transaction slack ratios vary over a range of values based on the ratio of $R_{\max}$ to the individual R_T's (and the *LSF* and *HSF* parameter settings). It is important to note that while the workload generator uses transaction resource requirements in assigning deadlines, we assume that the system itself lacks any knowledge of these requirements. This implies that a transaction can be detected as being late only when it actually misses its deadline, since 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.

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.[10] 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, value skew, and correlation.

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

[10]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 5000 transactions had been processed by the system. Only statistically significant differences are discussed here.

6.1. Performance Metrics

The primary performance metric is *Loss Percent*, which is computed as

$$LossPercent = \left[\frac{OfferedValue - RealizedValue}{OfferedValue} \right] * 100$$

i.e., it is the percentage of the offered value that is *not* realized by the system. 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.[11] A secondary performance metric, *Miss Percent*, measures the percentage of transactions that do not complete before their deadline. Note that when all transactions have the same value, the Loss Percent and Miss Percent metrics are identical. All the experiments evaluate these metrics as a function of the transaction arrival rate.

6.2. Parameter Settings

The resource parameter settings are such that the mean CPU time to process a page is 10 milliseconds while mean disk access times are 20 milliseconds. For experiments that were intended to factor in the effect of resource contention on the performance of the mappings, the number of processors and number of disks were set to 8 and 16, respectively. For experiments intended to isolate the effect of data contention, we approximately simulated an "infinite" resource situation [Fran85, Agra87], that is, where there is no queueing for resources. This was done by increasing twenty-five-fold the number of processors and the number of disks, from their baseline values of 8 and 16 to 200 and 400, respectively. A point to note here is that while abundant resources are usually not to be expected in conventional database systems, they may be more common in RTDBS environments since real-time systems are usually sized to handle transient heavy loading. This directly relates to the application domain of RTDBSs, where functionality, rather than cost, is often the driving consideration.

Most of our experiments were conducted for a workload consisting of a single class. For experiments designed to evaluate the effect of skew in transaction values, however, the workload consisted of two transaction classes. The value for each transaction is chosen uniformly over the range of values of its class, and is independent of the transaction's other characteristics.[12] The *GlobalMeanValue* parameter was kept constant across all the experiments at a value of 100.0.

To serve as a basis for comparison, apart from the candidate priority mappings described in Section 3, the following priority mappings are also evaluated in our performance study:

(1) *No Priority (NP)*: All transactions are given the same priority in this mapping. The performance obtained under this mapping should 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.

[11]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.

[12] The values are taken from the real number domain, and are not simply integers.

(2) *Random Priority (RP)*: This mapping randomly assigns priorities to transactions without taking into account any of their characteristics. The performance obtained under this mapping reflects how much performance can be obtained by the mere existence of *some* fixed priority ordering among the transactions.

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. The results for the fixed-tradeoff priority mappings will be explained first, after which results for the performance of the bucket algorithm will be presented. Finally, the effects of penalty for missed deadlines and correlation in the workload will be discussed.

6.3. RESOURCE CONTENTION (RC)

Our first set of experiments investigated the performance of the priority mappings when resource contention is the sole performance limiting factor. We began our experiments by first developing a baseline model around which further experiments were constructed by varying a few parameters at a time. The settings of the workload parameters and resource parameters for the baseline model are listed in Tables 3 and 4. 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 necessary for this set of experiments as all transactions are *queries*. There is a single transaction class, and transaction values range between 50.0 and 150.0. Deadline formula DF1, which makes transaction deadlines to be independent of their execution time, is used for this set of experiments. The workload settings related to deadline slack assignment are such that the spread in slack factor, $\frac{HSF}{LSF} = \frac{4.0}{1.33}$, is the same as the spread in value, $\frac{150.0}{50.0}$, namely 3. These settings ensure that variations in both deadline and value play a role in determining the overall system performance.

Parameter	**Value**
DatabaseSize	1000 pages
PageCount	16 pages
WriteProb	0.0
DeadlineFormula	DF1
LSF	1.33
HSF	4.0
GlobalMeanValue	100.0
NumClasses	1
ProbClass[i]	1.0
OfferedValue[i]	1.0
MeanValue[i]	100.0
SprdValue[i]	50.0

Table 3: Baseline Model Workload Settings

Parameter	Value
NumCPUs	8
NumDisks	16
PageCpu	10 ms
PageDisk	20 ms

Table 4: Baseline Model Resource Settings

6.3.1. Baseline Model

For the baseline model, Figures 3a and 3b show the Loss Percent results under normal load and heavy load conditions, respectively. Figure 3c shows the corresponding Miss Percent results. (Note that the curves for HV and VD are identical in these figures). From this set of graphs, it is clear that at low loads, the ED (Earliest Deadline) mapping realizes the most value (smallest Loss Percent). This might be considered surprising since ED is a value-indifferent mapping, and some of the other algorithms are value-cognizant. The reason for ED's good performance can be understood, however, by examining the Miss Percent characteristics (Figure 3c) at low loads. Since ED misses the deadlines of very few (if any) transactions, it delivers the most value. The value-cognizant mappings, HV (Highest Value), and to a lesser extent, VRD (Value-inflated Relative Deadline), focus their effort on completing the high-value transactions. In the process, they prevent some lower value transactions from making their deadlines, even though most of the 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, and its performance actually is close to that of NP (No Priority) at high loads. This is because at high 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. Under ED, no progress is made initially by a transaction, but as its deadline approaches, fast progress is made. The *net* progress made by ED, however, is around the same as that of NP. This was experimentally confirmed by measuring the average progress that had been made 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. Therefore, Earliest Deadline is not the right mapping to use under overload conditions. This behavior of ED over the loading range has also been noted in [Jens85, Huan89].

Turning our attention to the RP (Random Priority) mapping, we observe that although it behaves poorly at low loads, it behaves better than ED at high loads. This might be considered surprising since RP, unlike ED, does not take into account any transaction characteristics. The reason for the observed behavior is the following: Under ED, 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, new transactions usually start off at low priority and become high priority transactions only as their deadline draws close. 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 starting when they arrive. Such transactions tend to make their deadlines, and therefore there is always some fraction of the transactions in the system that are guaranteed to make

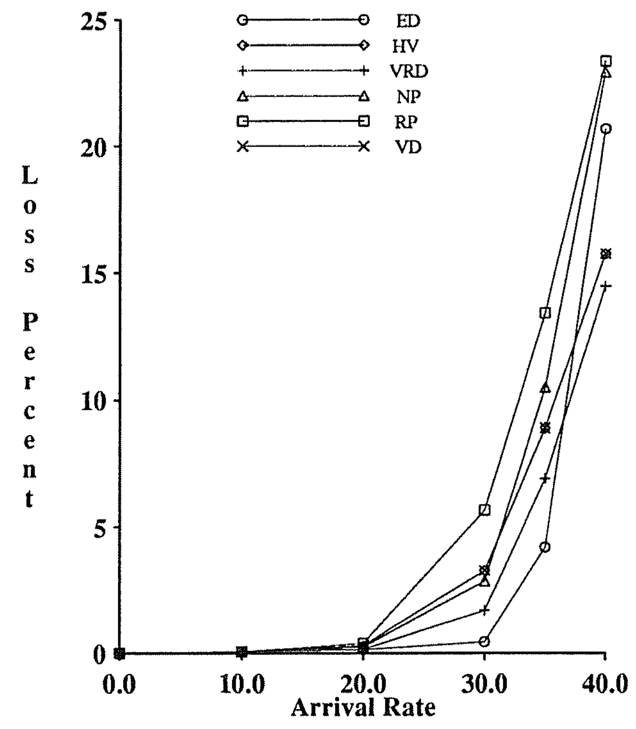

Figure 3a: RC Baseline Model (Normal Load)

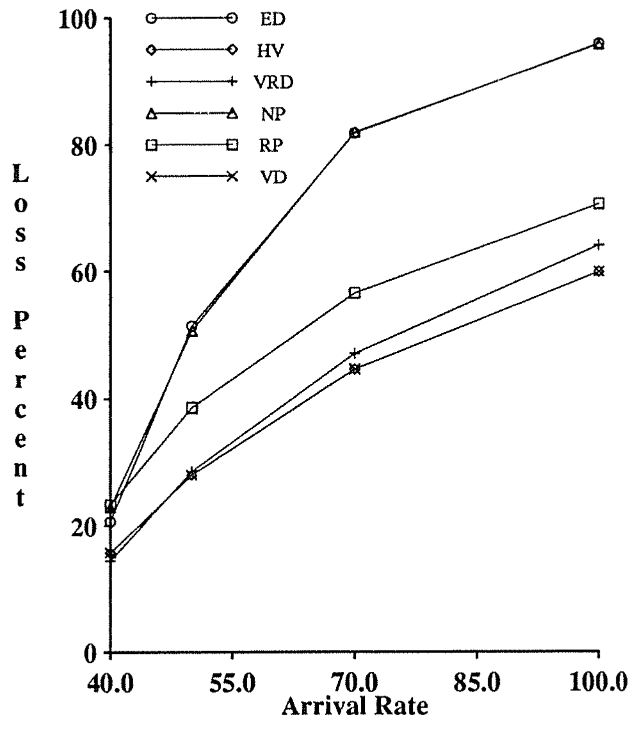

Figure 3b: RC Baseline Model (Heavy Load)

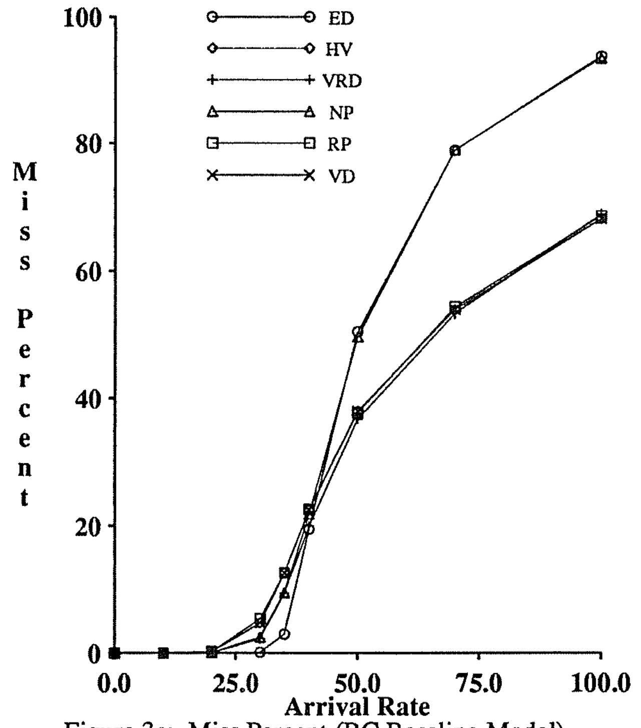

Figure 3c: Miss Percent (RC Baseline Model)

their deadlines. This explanation is confirmed by observing the Miss Percent characteristics of ED and RP at high loads, where we observe that the Miss Percent of ED is much higher than that of RP.

Focusing next on the HV (Highest Value) mapping, we observe that it performs worse than ED at low loads but improves its performance as the load increases. In fact, at high loads, it outperforms all the other algorithms. This is because following the Highest Value principle is a good 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. If we look at the Miss Percent characteristics (Figure 3c), we observe that HV and RP (Random Priority) behave identically with respect to this metric. The reason for this behavior is that the workload has transaction values being independent of other transaction characteristics and all transaction values are distinct. In such a case, an HV priority ordering is no different from an RP priority ordering in terms of the ability of the RTDBS to make transaction deadlines. Note that if there were groups of transactions that had the *same* value, then this would not be the case, as same-value transactions introduce NP-type behavior into the performance of HV.

Moving on to the VD (Value-inflated Deadline) mapping, we observe that although this mapping appears to combine the Earliest Deadline and Highest Value principles in its priority assignments, it performs identically to the HV mapping. This is not a coincidence, but is, in fact, always true: As time progresses, the D_T term in $\frac{D_T}{V_T}$ becomes large enough that it is approximately the same for all transactions. Therefore, once the clock time is sufficiently large, VD behaves exactly like HV. For this reason, we will not consider the VD mapping any further in this paper. The more general lesson that can be learned from the behavior of VD is that priority computations that combine values and absolute deadlines should be designed with care to ensure that the above problem is not encountered. In [Huan89], it was observed that a priority assignment of $P_T = V_T\ (w_1(t - A_T) - w_2 {*} D_T)$, where w_1 and w_2 are weighting factors, displayed little difference in performance with different settings for the weights. The probable reason is that with any non-zero value for w_2, the absolute deadline D_T term in the formula dominates the other term once the clock time is sufficiently large, and thus the priority assignment degenerates to a HV mapping. Therefore, the actual weights should not, in fact, be expected to impact the long-term performance of this mapping.

Turning our attention to the VRD (Value-inflated Relative Deadline) mapping, we observe that its performance is intermediate to that of ED and HV. At low loads, it is slightly worse than ED, while at high loads it is slightly worse than HV. In a sense, therefore, it delivers the best overall performance. Note that while VRD, like VD, takes both deadlines and values into account, it does not behave like HV. The reason is that the mapping uses the *relative* deadline, rather than the absolute deadline, to compute transaction priorities. This makes the VRD mapping both value and deadline cognizant for this workload. The reason that the VRD mapping does better than HV at low loads is that it has a partial Earliest Deadline effect in that jobs with small relative deadlines are given priority over jobs with larger relative deadlines. Among sets of similar valued jobs that arrive at around the same time, the priority ordering is therefore approximately Earliest Deadline. Due to this effect, at low loads, fewer deadlines are missed by VRD when compared to HV (Figure 3c). Conversely, at high loads, when a large fraction of deadlines are missed, the fact that VRD takes deadline into account works against it since a high-value transaction may not be completed due to having a large relative deadline.

The next experiment that we conducted examined the effect of increasing the spread in transaction values. For this experiment, the *SprdValue* parameter was increased from the baseline value of 50 percent up to 99 percent, while keeping the other parameters the same as in the baseline model. This means that transaction values now ranged between 1.0 and 199.0. The Loss Percent results for this experiment are shown in Figures 4a and 4b. We first note that the performance of the ED, RP and NP mappings remains the same as that in the baseline experiment. This is because these mappings are value-indifferent, and therefore changes in the value distribution do not affect their performance (as long as the mean value remains the same). The value-cognizant mappings, HV and VRD, however, improve their performance considerably. This is because these mappings concentrate on the more valuable transactions, and increasing the value spread implies that, on the average, greater value is obtained for each high-value transaction that is completed. The low-value transactions that are missed have a lesser effect on the realized value since their average value is smaller due to the increased spread. Note that the Miss Percent characteristic of HV is the same as in the baseline experiment (Figure 3c) since the workload assigns values to transactions independently of their other characteristics.

The next experiment examined the effect of decreasing, rather than increasing, the spread in transaction values. For this experiment, the *SprdValue* parameter was set to 0 percent, keeping the other parameters the same as those of the baseline model. This means that all transaction values had the same value of 100.0. The Loss Percent results for this experiment are shown in Figures 5a and 5b. The value-cognizant mappings, HV and VRD, perform worse here when compared to the baseline experiment. The HV mapping, in fact, behaves just like the NP mapping. The reason for HV behaving like NP is that when all values are the same, HV gives every transaction the same priority. While all transactions having the same value is an extreme case, similar problems would arise when the workload consists of multiple transaction classes where all transactions within a class have the same value. The VRD algorithm, unlike HV, does not behave like NP; this is because the relative deadline component of its priority mapping ensures that there is a priority ordering among the transactions. Also, at high loads, VRD behaves similar to RP rather than ED. This implies that VRD is more a value-oriented mapping than a deadline-oriented mapping at high loads since the relative deadline component has only a randomizing effect when all values are the same.

Another interesting observation that can be made here is that the RP (Random Priority) mapping actually performs quite well at high loads. This means that if a *random "noise"* is added to priority values, stability in high-load performance can be obtained even when most or all of the priority values would otherwise be the same. For example, if even an infinitesimally small noise is added to the transaction priorities generated by the HV mapping for this experiment, the heavy load performance would be like that of RP rather than that of NP. This is because the addition of the noise causes a priority ordering to *exist* where there was none originally. Note that the noise should be random, and not based on transaction characteristics. If transaction deadlines were used to generate the noise, for example, the performance would be like that of ED, and not of RP, at high loads. It should also be noted that the high-load stability obtained by the addition of noise is gained at some cost in normal load performance, as RP performs worse than NP in this loading range.

Summarizing the results of the above set of experiments, we can draw the following conclusions for the *uniform* workloads examined in this section: First, at low loads, when the Miss Percent is low, the Earliest Deadline

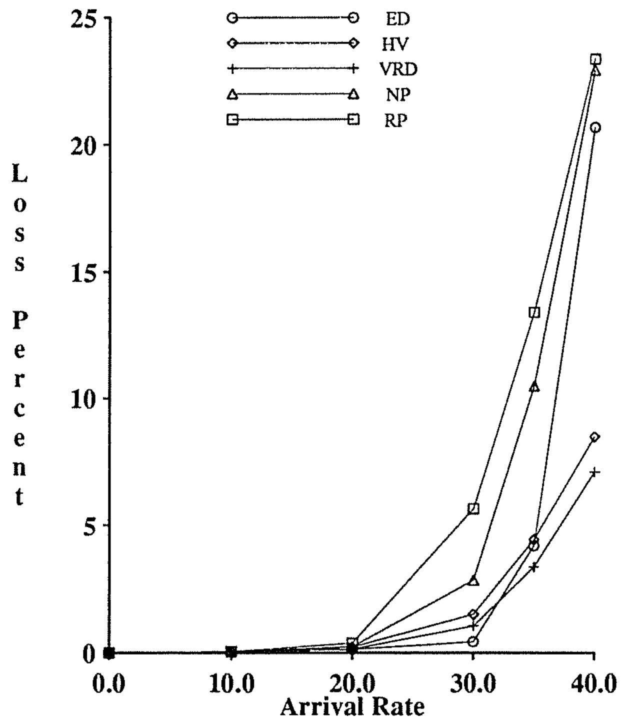

Figure 4a: Increased Spread (Normal Load)

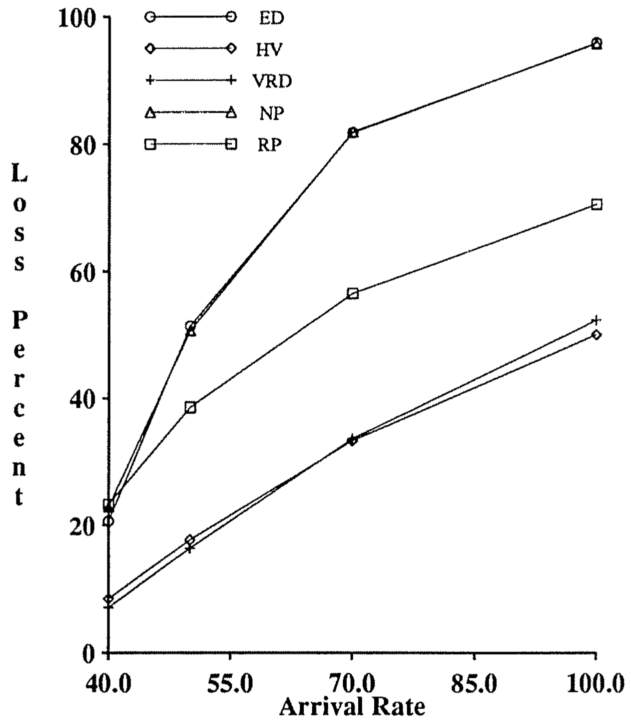

Figure 4b: Increased Spread (Heavy Load)

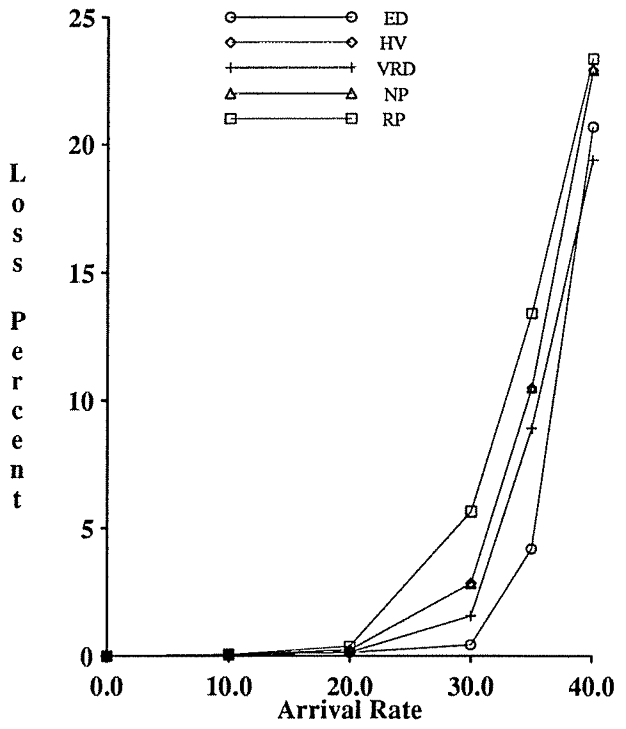

Figure 5a: Decreased Spread (Normal Load)

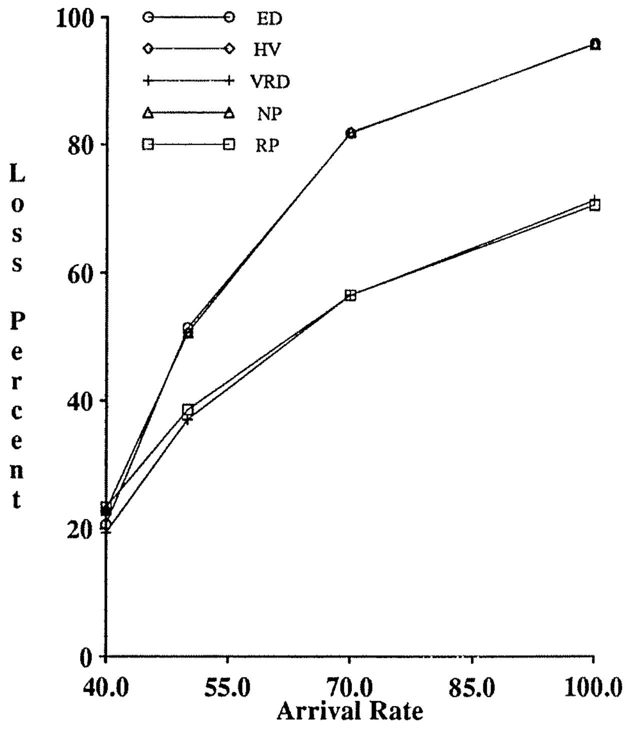

Figure 5b: Decreased Spread (Heavy Load)

priority ordering is the right choice, while at high loads, when the Miss Percent is high, the priority ordering given by the Highest Value principle realizes the most value. Second, the degree of spread in transaction values has a significant effect on the performance of the value-cognizant mappings. In particular, their performance improves with an increased spread in values. Third, the use of absolute deadlines in priority assignments should be handled with care. Finally, priority mappings should have a built-in noise factor to guard against the possibility of transactions having identical priorities, as transactions can hinder the progress of each other and thus degrade performance at high loads.

6.3.2. Transaction Value Skew

The next experiment examined the effect of having a skew in the transaction value distribution. For this experiment, the parameters are set as shown in Table 5. They construct a two-class workload where 10 percent of the transactions deliver 90 percent of the offered value. The values of the transactions from the first class vary between 450.0 and 1350.0, while the values of the second class vary between 5.5 and 16.5. (Note that the *Global-MeanValue* parameter is the same as for the baseline experiment). The Loss Percent results for this experiment are shown in Figure 6. We observe, as in the previous experiments, that the performance of the ED, RP and NP mappings remains the same as in the baseline experiment since these mappings are value-indifferent. The figure also shows that the performance of the value-cognizant mappings, HV (Highest Value) and VRD (Value-inflated Relative Deadline), improves greatly over the baseline and they are now much superior to the value-indifferent mappings. Note that even at low loads, they perform almost as well as the Earliest Deadline mapping. The value-cognizant mappings, by making certain that all of the (few) high-value transactions make their deadline, ensure that they always realize 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. Note also that the performance of the VRD mapping is almost identical to that of the HV mapping. This is because when the spread in value is much larger than the spread in relative deadline, the V_T component of the VRD mapping dominates the $(D_T - A_T)$ component in determining relative transaction priorities. Therefore, for workloads with these features, the VRD mapping generates a priority ordering

Parameter	Value
DatabaseSize	1000 pages
PageCount	16 pages
WriteProb	0.0
DeadlineFormula	DF1
LSF	1.33
HSF	4.0
GlobalMeanValue	100.0
NumClasses	2
ProbClass[i]	0.1, 0.9
OfferedValue[i]	0.9, 0.1
MeanValue[i]	900.0, 11.0
SprdValue[i]	50.0, 50.0

Table 5: Skew Workload Settings

very similar to that of the Highest Value (HV) mapping, and is only marginally deadline-cognizant.

We can conclude from this experiment that skew in transaction values causes the value-cognizant algorithms to perform much better. For workloads that have a considerable spread in transaction values, the priority ordering established by the HighestValue principle ensures good performance through the entire loading range. These results also demonstrate the significant impact of value distributions on the relative performance of the algorithms.

6.4. DATA CONTENTION (DC)

The second set of experiments investigated the performance of the priority mappings when data contention is the sole performance degradation factor. As before, we began our experiments by first developing a baseline model around which we then constructed further experiments by varying a few parameters at a time. The settings of the workload parameters for this baseline model are identical to those for the Resource Contention case (listed in Table 3) except that the *WriteProb* parameter, which gives the probability that a page which is read is also written, is set to 0.25 instead of 0.0. Deadline formula DF1 is again used for the assignment of transaction deadlines. The settings of the resource parameters are shown in Table 6. The high settings for the quantity of hardware resources ensure that resource contention levels are extremely low, and thus the performance differences observed between the mappings are primarily due to data contention. Due to space limitations (and for graph clarity), we do not discuss the RP (Random Priority) and NP (No Priority) mappings in the following sections.

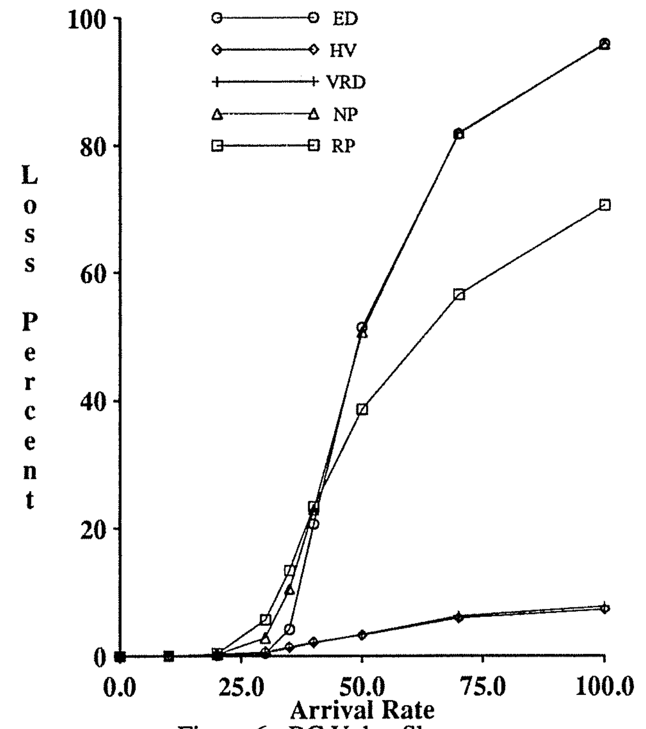

Figure 6: RC Value Skew

Parameter	Value
NumCPUs	200
NumDisks	400
PageCpu	10 ms
PageDisk	20 ms

Table 6: Data Contention Resource Settings

6.4.1. Baseline Model

For the baseline model, Figures 7a and 7b show the Loss Percent results for the various priority mappings under normal load and heavy load, respectively. Figure 7c shows the corresponding Miss Percent behavior. The results shown were separately obtained with the 2PL-HP, OPT-BC and OPT-WAIT concurrency control algorithms. Focusing our attention on the performance of 2PL-HP (the solid lines), we observe that, *qualitatively*, the mappings exhibit the same behavior as in the case of the resource contention baseline model (Figures 3a, 3b). The ED (Earliest Deadline) mapping performs the best at low loads, while the HV (Highest Value) mapping outperforms all of the other algorithms at high loads. As before, ED performs well at low loads since it misses far fewer deadlines. Moreover, data contention (unlike resource contention) is not *work-conserving* because already performed work has to be redone after a transaction restart; therefore, ensuring that the most urgent transactions are given the highest priority is even more beneficial at low loads here. At high loads, following the Highest Value principle is again the right approach, as the data contention level is high enough that only a fraction of the transactions in the system are able to complete before their deadlines; in such a situation, the transactions that should be given priority are those that can deliver high values.

Turning our attention to OPT-BC (the dashed line), here all of the priority mappings behave exactly the same since OPT-BC is a priority-indifferent algorithm and there is virtually no resource contention; therefore, transaction priority does *not* play a role in determining system performance. The important point to note, however, is that in spite of this priority indifference, OPT-BC performs better than 2PL-HP for most of the loading range, especially at higher loads. The reason for this is obvious when we compare the Miss Percent characteristics, where we observe that OPT-BC misses far fewer deadlines than 2PL-HP (Figure 7c). The primary reason for the lower number of misses is that the optimistic approach, due to its validation stage conflict resolution, ensures that eventually discarded transactions do not cause the restart of other transactions [Hari90a]. The locking approach, on the other hand, allows these soon-to-be-discarded transactions to cause other transactions to be either blocked or restarted due to lock conflicts, thereby increasing the number of late transactions.

Moving on to OPT-WAIT (the dotted lines), we observe that it performs worse than OPT-BC for all of the mappings except ED at low loads. The reason for OPT-WAIT doing better than OPT-BC at low loads for the ED priority mapping is that priority waiting is a good idea here since the more urgent transactions are not restarted by less urgent transactions. At high loads, however, the priority wait algorithm causes performance degradation due to an increase in system population (many waiters), which causes a steep increase in the number of conflicts. A more detailed explanation for this behavior of OPT-WAIT is given in [Hari90b]; although that study did not include transaction values, the explanations carry over here since ED is a value-indifferent mapping. The reason that OPT-

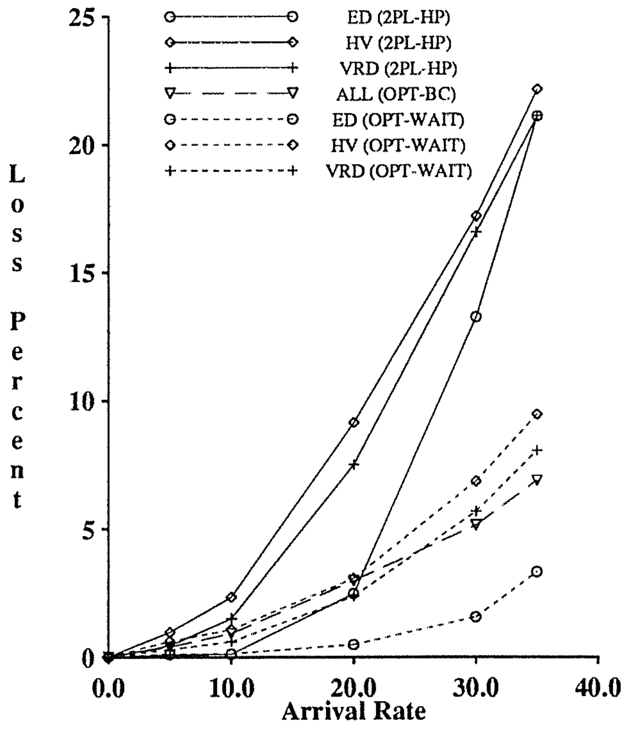

Figure 7a: DC Baseline Model (Normal Load)

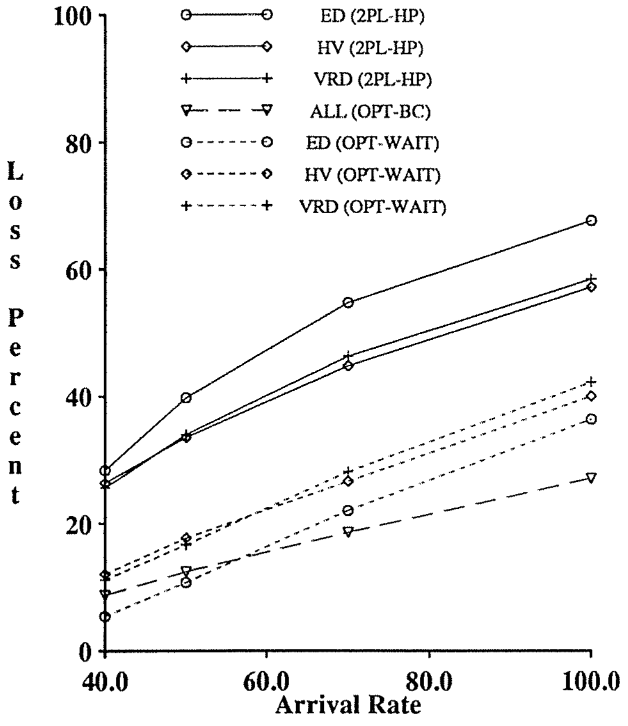

Figure 7b: DC Baseline Model (Heavy Load)

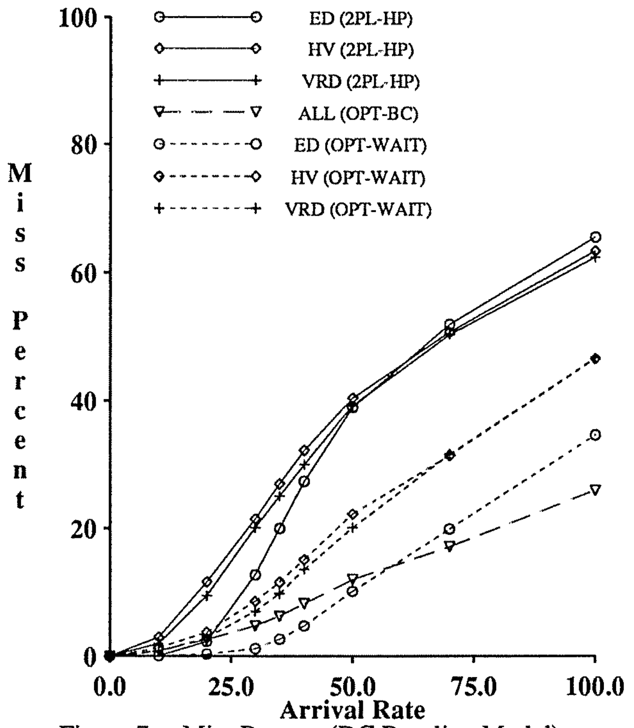

Figure 7c: Miss Percent (DC Baseline Model)

WAIT performs worse than OPT-BC over most of the loading range for the mappings other than ED is the following: With the ED priority mapping, a waiting transaction never has to wait beyond its deadline, since it will have the highest priority in the system at its deadline. For the other mappings, however, this is not necessarily the case. If we consider HV, for example, it is clear that the waiting process could extend beyond the waiter's deadline since the higher-value conflicting transactions may not have completed by the waiter's deadline. In such a case the waiter is aborted and discarded, and the waiter's value is therefore lost. This wouldn't be so bad if the higher priority transaction then made its own deadline and the system realized its value. There is no guarantee, however, that this will actually happen. We could, instead, have many *wasted sacrifices* – cases where a transaction is discarded on behalf of another transaction that later does not complete. Such sacrifices are useless and cause performance degradation. It should be noted, however, that the performance of OPT-WAIT is still better than that of 2PL-HP for all of the mappings throughout the entire loading range.

6.4.2. Transaction Value Skew

The next experiment examined the effect of having a skew in the transaction value distribution. For this experiment, the workload parameters are the same as for experiment 6.3.2 (Table 5), except that the *WriteProb* parameter is set to 0.25. The resource parameter settings for this experiment are shown in Table 6. The Loss Percent results of the experiment are shown in Figure 8 for the 2PL-HP, OPT-BC and OPT-WAIT concurrency control algorithms. Focusing our attention on 2PL-HP (solid lines), we observe that the performance of the value-cognizant mappings improves tremendously and that they are now far superior to the ED mapping, as in the pure resource contention case. The reason for this improvement is the following: 2PL-HP ensures that the highest priority transactions are virtually guaranteed to make it to their deadline. The value-cognizant mappings, in combination with 2PL-HP, realize a high value since they assign the highest priorities to the high-valued transactions. Successfully making the deadlines of the few high-value transactions is by itself sufficient to realize at least 90 percent of the offered value.

Moving on to OPT-BC (dashed line), we note that its performance remains the same as in the baseline data contention experiment (Experiment 6.4.1). This is because OPT-BC does not take into account transaction values, and therefore changes in the transaction value distribution do not affect its performance. From Figure 8, it is clear that the performance of the value-cognizant mappings under 2PL-HP is superior here to their performance under the OPT-BC algorithm. Note that this is in spite of 2PL-HP having a much higher Miss Percent than OPT-BC. Since 2PL-HP concentrates on the high-value transactions, the value it derives from them more than compensates for the value lost due to missing the deadlines of a large number of low-value transactions. OPT-BC, on the other hand, treats all transactions equally, which can cause high-value transactions to be restarted (and therefore miss their deadline) due to the commits of low-value transactions.

Turning our attention to OPT-WAIT (dotted lines) and comparing these results with the 2PL-HP results, it can be observed that all of the mappings perform better for OPT-WAIT than for 2PL-HP, including the value-cognizant algorithms. The reason for this is that, since OPT-WAIT is priority cognizant and is willing to sacrifice low priority transactions for high priority transactions, the high-value transactions are guaranteed to complete before their deadlines. This is similar to the behavior of 2PL-HP. In addition, OPT-WAIT gains some extra value over 2PL-HP due to missing the deadlines of a smaller number of low-value transactions. To sum up, as compared to

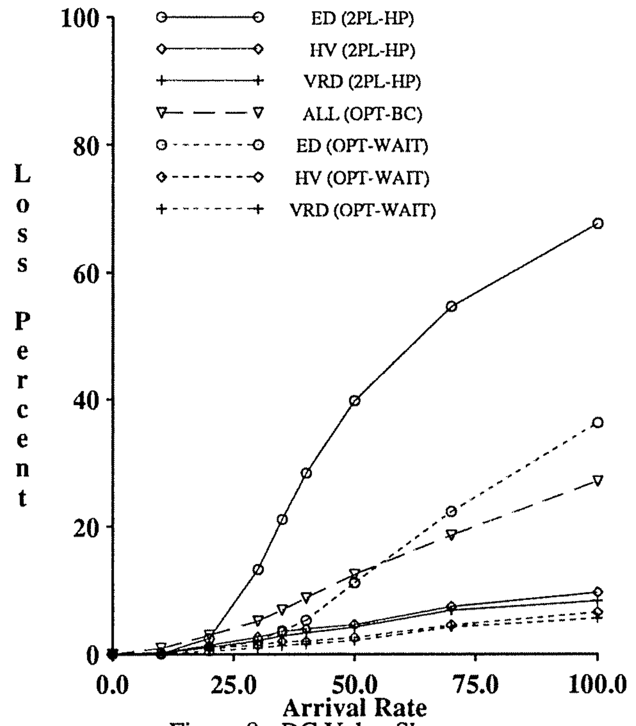

Figure 8: DC Value Skew

2PL-HP, OPT-WAIT makes all of the same high-value transactions and misses fewer low-value transactions.

In [Hari90a], it was shown that optimistic algorithms outperform locking algorithms in firm deadline systems. That study assumed that all transactions have the same value. The results from our experiments here demonstrate that optimistic algorithms can also perform better than locking algorithms when the real-time environment incorporates the notion of value and the priority mappings are value-cognizant.

6.5. DATA and RESOURCE CONTENTION COMBINED (DC+RC)

We conducted several experiments where both resource contention *and* data contention contribute towards system performance degradation. This was done by using limited hardware resources (8 CPUs and 16 disks), a write probability of 0.25, and deadline formula DF1 for assigning transaction deadlines. The qualitative results were the same as those obtained for resource contention or data contention alone. The ED (Earliest Deadline) mapping is the best at normal loads, while the HV (Highest Value) mapping is the best at high loads. Also, the performance of the value-cognizant mappings improves with the transaction value spread or with the transaction value skew. The performance of the priority mappings for 2PL-HP under the baseline workload model is shown in Figures 9a and 9b. As is clearly evident from these graphs, the results are qualitatively similar to those of Figures 3a-c or Figures 7a-c. We also conducted this experiment using the two optimistic algorithms, OPT-BC and OPT-WAIT. Both of these algorithms outperformed 2PL-HP over virtually the entire loading range for all of the mappings. At low loads, OPT-WAIT did slightly better than OPT-BC for the ED mapping and did slightly worse for the other mappings, which is similar to the results of the baseline pure data contention experiment (Experiment 6.4.1). At high loads, OPT-WAIT and OPT-BC had the same performance for all the mappings. The reason for this is that, with heavy resource contention, it is rare for a low-priority transaction to reach its validation stage before a conflicting high-priority transaction. Accordingly, the priority wait mechanism of OPT-WAIT rarely comes into play, and OPT-WAIT therefore exhibits OPT-BC-like behavior at high loads here.

The main conclusion from this set of experiments is that both resource contention and data contention affect the performance of the mappings in a similar fashion, so the results are qualitatively the same for both types of contention. It should be noted that these results can also be traced to the fact that we use priority in a consistent fashion for both resource scheduling and concurrency control.

6.6. BUCKET ALGORITHM (BA)

The experiments described in the previous sections provided insight into the behavior of the mappings that employ a fixed tradeoff between values and deadlines. In this section, we present results for experiments that evaluated the performance of the bucket algorithm. The bucket mapping is evaluated for four settings of the *NumBuckets* parameter: *NumBuckets*=1,2,4,∞. These will hereafter be referred to as BA1, BA2, BA4 and BA∞, respectively. The BA1 and BA∞ mappings produce priority orderings identical to those of the ED (Earliest Deadline) and HV (Highest Value) mappings, respectively.[13] These curves help to put the results obtained in this section

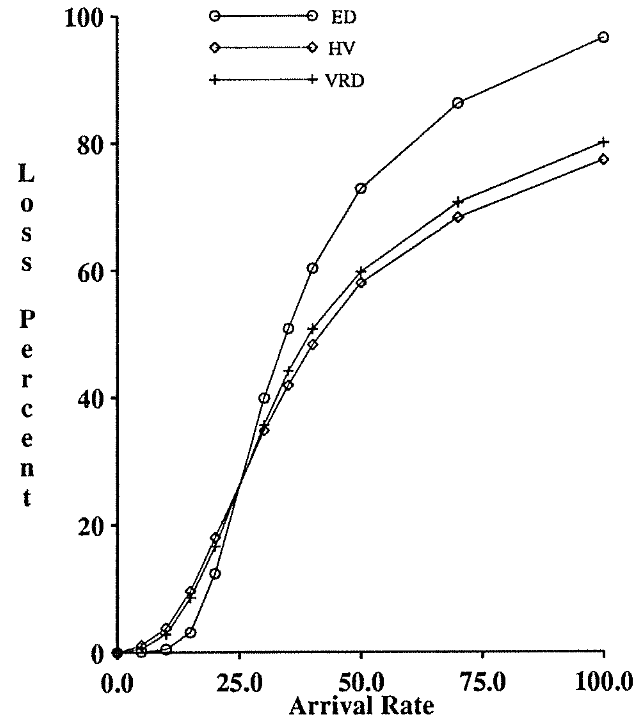

Figure 9a: DC+RC Baseline Model (2PL-HP)

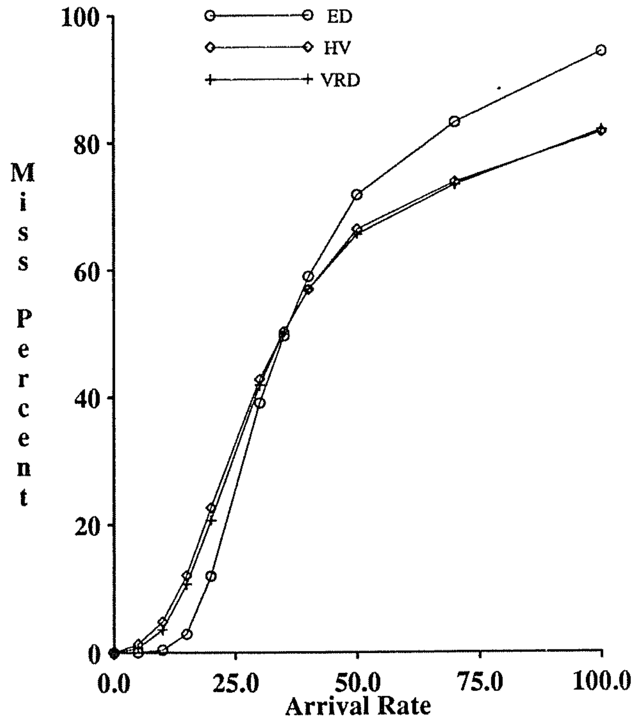

Figure 9b: DC+RC Miss Percent (2PL-HP)

[13] Note that BA∞ gives the same priority ordering as HV only when all transactions have distinct values. HV assigns equal priority to transactions with the same value, while BA∞ assigns different priorities due to the random noise component of the bucket algorithm. This also protects BA∞ from the poor performance behavior observed with HV when all values are the same (Expt. 6.3.1).

in perspective with those described in the previous sections.

All of the experiments that were carried out for the fixed-tradeoff mappings in our study were also performed for the bucket algorithm. Due to space limitations, we will present the performance of the bucket algorithm in detail for only a subset of the experiments. The experiments that will be discussed here are the baseline resource contention experiment and the resource contention with value skew experiment; the results of the other experiments will be briefly summarized.

6.6.1. Resource Contention

For the baseline model where resource contention is the sole performance degradation factor, Figures 10a and 10b show the Loss Percent behavior of the bucket algorithm under normal load and heavy load conditions, respectively. From this set of graphs, the following observations can be made: At low loads, up to an arrival rate of 35.0, the BA1 (Earliest Deadline) mapping performs the best. As the load increases beyond this point, the performance of BA1 deteriorates and BA2 starts to deliver the best performance. When the loading is increased beyond an arrival rate of 70.0, the performance of BA2 deteriorates, and BA4 starts to deliver the best performance. From this trend, we can observe that as the loading level increases, the number of buckets required to provide good performance also increases. In the limit, when the loading level is extremely high, a bucket count of ∞, which corresponds to the BA∞ (Highest Value) mapping, will provide the best performance. The reason for this observed behavior is as follows: Within a bucket the priority ordering is Earliest Deadline. Therefore, the bucket count has to be at a level such that the miss percentage in the first bucket is small enough for the Earliest Deadline policy to work well. At low loads, a single bucket is sufficient since the overall miss percentage is small. As the loading increases, and more transaction deadlines are missed, the bucket count has to be increased to ensure that the miss percentage of transactions in the first bucket is kept small. Put another way, the bucket count controls the level of *mixing* of low and high-value transactions in a single bucket. If the mix is too "thin" (too many buckets), the system may miss out on several lower value transactions whose deadlines it could have made, and will thus deliver reduced value. If the mix is too "thick" (too few buckets), the system may spend resources on transactions with low values and in the process may lose high-value transactions. Therefore, there is a bucket count at each operating point that delivers the "right" mix and thus provides the "right" tradeoff between value and deadline. It should also be noted that with the appropriate choice of the bucket count, the bucket algorithm generates superior performance to all the other mappings that were considered (see Figures 3a, 3b) over the entire loading range. Of course, an additional adaptive mechanism is still required in order that the bucket algorithm may dynamically change the number of buckets to match the system loading level.

6.6.2. Transaction Value Skew

The next experiment examined the effect of skew in transaction values on the performance of the bucket algorithm. The experiment was conducted for the 10-90 workload described earlier (see Table 5), where 10 percent of the transactions offer 90 percent of the value. The results of this experiment are shown in Figures 11a and 11b. From these figures, it is clear that all of the bucket mappings perform about the same at low loads. As the loading level is increased, however, the bucket mappings, in order of bucket count, start performing badly. The BA1

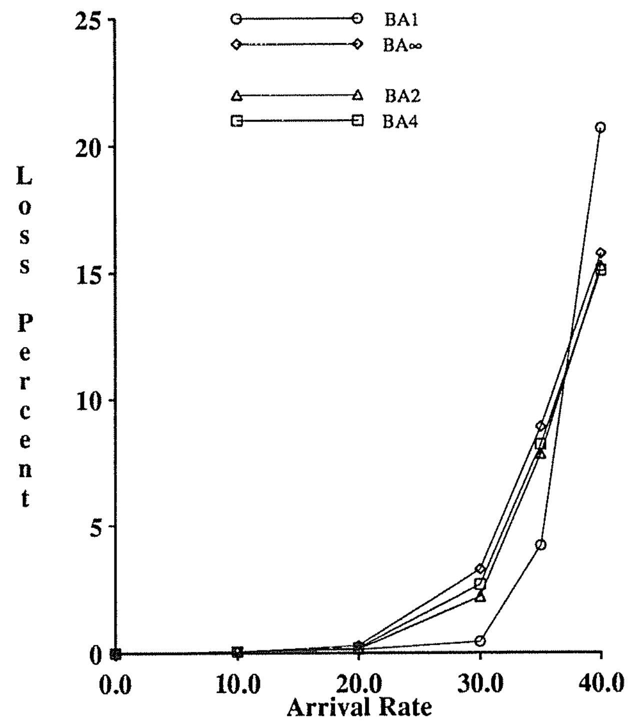

Figure 10a: Bucket RC Baseline (Normal Load)

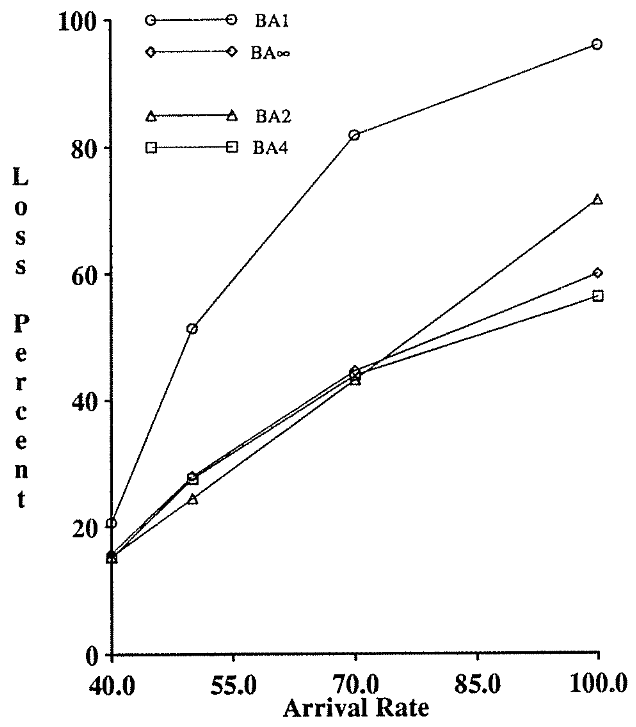

Figure 10b: Bucket RC Baseline (Heavy Load)

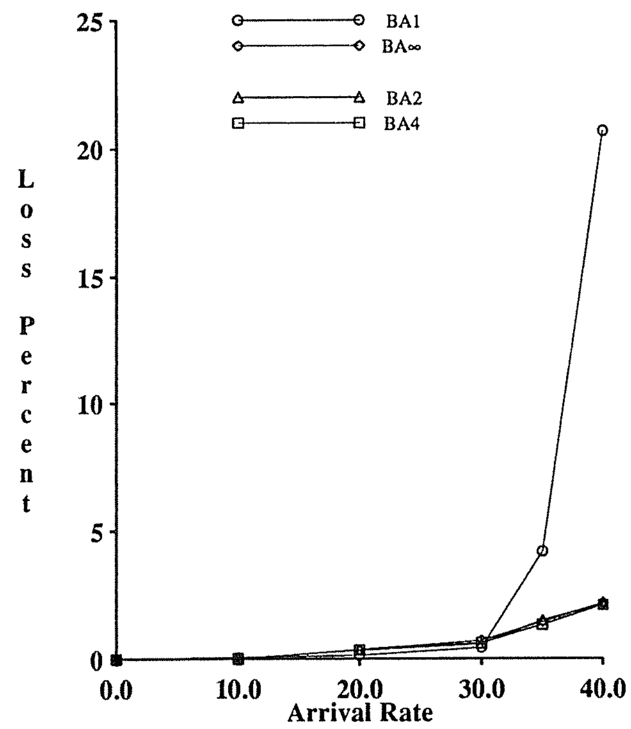

Figure 11a: Bucket RC Value Skew (Normal Load)

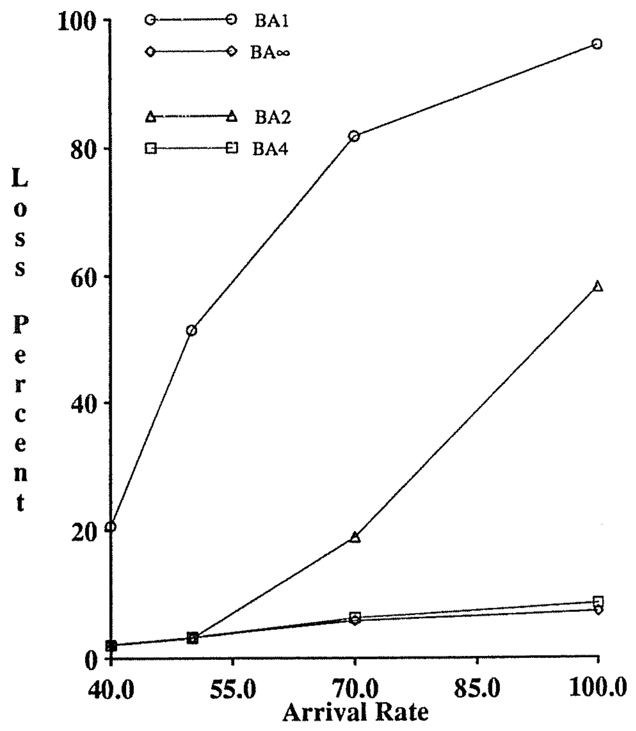

Figure 11b: Bucket RC Value Skew (Heavy Load)

mapping deteriorates from an arrival rate of 30.0 onwards, BA2 does poorly after an arrival rate of 50.0, and BA4 is just about to start behaving worse at an arrival rate of 100.0 (this was confirmed by running the experiment for higher loads). From these results, it is clear that when there is considerable skew in the value distribution, the BA∞ (Highest Value) mapping is the priority ordering of choice, just as we saw before. This is because even at low loads, although it does miss more deadlines than the other algorithms, BA∞'s poor performance on this front is compensated for by the high values of the transactions that it *does* complete. Note, however, that if the load were heavy enough that about 90 percent of the transactions are missing their deadline, then a policy like BA10 (which splits up the transactions into 10% sized buckets) should be expected to perform better than BA∞. This is because BA10 would be using Earliest Deadline among the high-value transactions which populate the first bucket, while BA∞ would be using a Highest Value ordering. Since EarliestDeadline is better than Highest Value for a set of transactions that *can* be completed by their deadlines, BA10 could be expected to outperform BA∞.

The main conclusion to be drawn from the above set of experiments is that by layering the transactions based on value, and then using Earliest Deadline within each bucket, the bucket algorithm exhibits a structured and logical approach towards the objective of maximizing the realized value. For each workload, there is a "right" bucket count that delivers good performance.

6.7. LATE PENALTY

In our experiments so far, we have assumed that there is no *penalty* associated with missing a transaction deadline. In real-life systems, however, a penalty may be paid for missing a deadline. For example, in a quality control system that tests products coming off an assembly line, a missed deadline may mean that the untested object has to be categorized as defective. The penalty here is the manufacturing cost that went into the production of the object. This notion of penalty for non-delivery of service can be used in an RTDBS to capture the loss incurred due to missing a transaction deadline. If missing a deadline has an associated penalty, then these penalties must be discounted from the value realized by the system in computing the *net* realized value. The penalty notion therefore provides a quantitative way to combine the separate metrics of Loss Percent and Miss Percent, since the total penalty that has to be paid by the system is a function of the missed deadlines. The Loss Percent metric is now computed by the formula

$$LossPercent = \left[\frac{(OfferedValue - RealizedValue) + TotalPenalty}{OfferedValue} * 100 \right]$$

where *TotalPenalty* is the sum of the penalties of all late transactions.

In order to investigate the effect of penalties on the performance of the various mappings, we conducted some preliminary experiments where missing a deadline has a penalty associated with it. We have considered only the case where all transactions have the same penalty. For this special case, the priority mappings in the presence of penalty are the same as those that are generated when value alone is the consideration. This is because the total penalty in this case is a function of how *many* transactions miss their deadline, and not *which* transactions miss their deadline.

We conducted the pure data contention with skewed value distribution experiment (Experiment 6.4.2) for the HV (Highest Value) priority mapping using 2PL-HP and OPT-BC as the concurrency control algorithms with a penalty of 100.0 for each missed deadline. The Loss Percent results for this experiment are shown in Figure 12 (solid lines). To aid in comparison, the corresponding results obtained in the absence of penalties (from Experiment 6.4.2) are also shown in Figure 12 (dotted lines). We observe that with penalty, the performance of the 2PL-HP algorithm is considerably worse than that of OPT-BC. In the absence of penalties, however, 2PL-HP performs euch better than OPT-BC. The reason for this behavior can be understood by looking at the Miss Percent characteristics for Experiment 6.4.2 (Figure 7c). There we see that 2PL-HP misses many more deadlines than OPT-BC. Therefore, when penalties are levied for missed deadlines, the total penalty loss for 2PL-HP is much higher than that for OPT-BC. The difference in loss here is sufficiently large for 2PL-HP to actually cause it to perform worse than OPT-BC. Of course, the degree of change in Loss Percent results relative to the no-penalty case is a function of the magnitude of the penalty that is levied for missed deadlines.

From the above experiment, we learn that algorithms that realize a high value by selectively completing only high-value transactions may suffer a significant performance degradation if a penalty is levied for each missed deadline. The penalty notion is therefore a mechanism for combining the value realized due to completed transactions with the loss suffered due to late transactions.

6.8. CORRELATION

In order to investigate the effects of correlation in transaction workload characteristics, we conducted one experiment where deadline formula DF2 was used to generate transaction deadlines. This formula introduces a

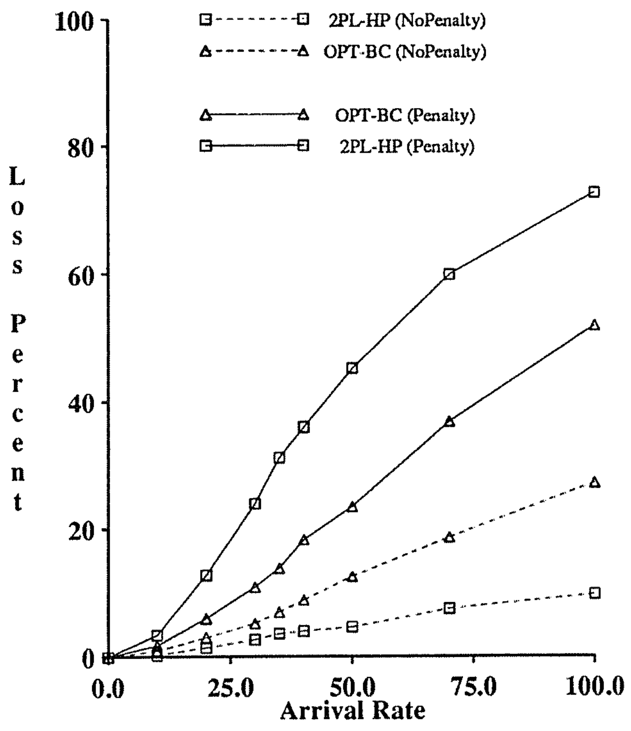

Figure 12: Late Penalty (HV)

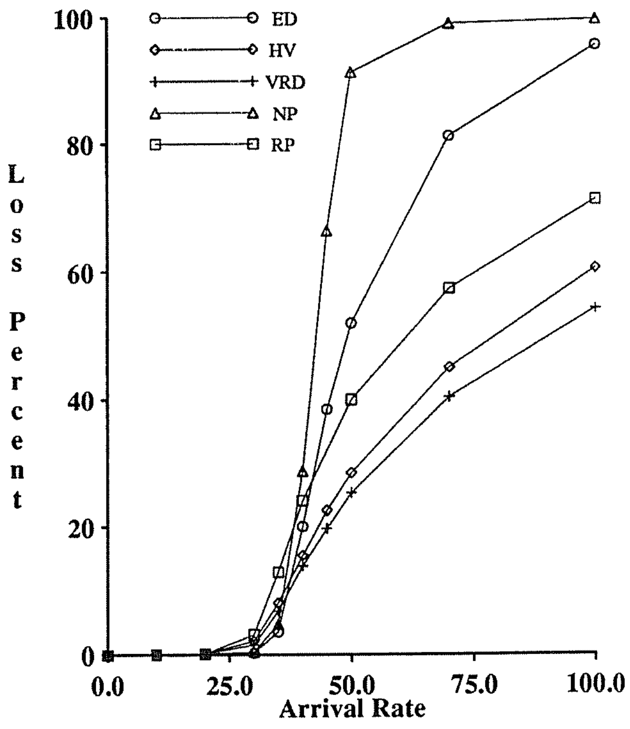

Figure 13: Deadline / Execution Time Correlation

linear correlation between deadline and execution time. For this experiment, the workload parameters *LSF* and *HSF* were both set[14] to 4.0, keeping all the remaining parameter settings to be the same as those of the baseline resource contention model (Section 6.3.1). The Loss Percent behavior for this experiment is shown in Figure 13. There are some important differences between the mapping behaviors shown in this figure and those shown in Figures 3a and 3b. Focusing on the performance of the VRD (Value-inflated Relative Deadline) mapping, we observe that its performance is better than that of the HV (Highest Value) mapping throughout the entire loading range. Recall that with deadline formula DF1, VRD had performed better than HV at low loads but worse at high loads. The reason for the different behavior here is that, with DF2, the VRD mapping generates a priority ordering that is identical to the ordering generated by the Value Density [Jens85] mapping. In the ValueDensity mapping, task priorities are determined by the ratio $\frac{R_T}{V_T}$, where R_T is the task execution time; this means that the task with the highest value density is given the highest priority. When tasks *do not* have time constraints, the ValueDensity mapping is known to produce a total value at every point in time that is at least as high as any other schedule [Jens85]. This can be intuitively understood by realizing that the mapping gives higher priority to those tasks that can yield more value in a shorter time period. The VRD mapping is equivalent to the ValueDensity mapping in this experiment due to the fact that the relative deadline and the execution time are linearly correlated in DF2. Therefore, since the VRD mapping concentrates on "quick-paying" transactions, its performance is even better than that of the Highest Value mapping (which concentrates only on "high-paying" transactions).

Turning our attention to the NP (No Priority) mapping, we observe that it performs much worse here than all the other mappings at intermediate and high loads. Recall that when deadline formula DF1 was used, the performance of NP was close to that of ED (Earliest Deadline) in these loading ranges. The reason for the change in behavior with DF2 is as follows: With the NP mapping, all transactions in the system make progress at the same rate. If DF2 is used, all transactions have the same slack ratio, whereas with DF1, short transactions (in terms of their processing requirements) tend to have *greater* slack ratios than long transactions. Transactions that have high slack ratios tend to complete before their deadline. Therefore, with DF1, the short transactions have a greater chance of completing before their deadline than long jobs. Also, since the long jobs have small slack ratios, they are discarded earlier. This skew in slack ratios has a beneficial effect on the MissPercent characteristic. With DF2, however, since all jobs have the same slack ratio, all transactions have the same chance of making their deadline, and this results in more missed deadlines. This explanation was confirmed by comparing the Miss Percent characteristics in the two experiments.

The conclusion that we can draw from this experiment is that correlation in workload characteristics can have an appreciable effect on the performance of the mappings, and it therefore is an area that should be investigated in greater detail.

[14] These parameters were chosen to match the *mean* slack ratio of the baseline model.

7. CONCLUSIONS

In this paper, we have addressed the issue of how to assign priorities to transactions in a real-time database system (RTDBS) when transactions are characterized by both values and deadlines. Using a detailed simulation model of an RTDBS, we studied the performance of several alternative approaches to combine values and deadlines into transaction priorities. While most of the mappings established a fixed tradeoff between values and deadlines, we also proposed and studied a more flexible mapping scheme. The performance metric underlying our simulation experiments was the total value provided by transactions that complete before their deadlines, and the experiments covered a range of workload characteristics and system operating conditions. In particular, workloads with different degrees of spread and skew in the transaction value distribution were considered, and the performance impacts of resource contention and data contention were considered both in isolation and in combination. We also conducted preliminary investigations of the effects of correlation among transaction characteristics and the impact of having penalties for late transactions.

Our experiments showed that for workloads with a limited, uniform spread in the transaction values, the Earliest Deadline (ED) mapping provided the best performance among the fixed-tradeoff mappings under light loads. Although ED is a value-indifferent mapping, the database system had sufficient resources at low loads to meet most transaction deadlines; consequently, prioritizing transactions according to their urgency led to the fewest missed deadlines and generated the most value. Under heavy loads, however, it was the Highest Value (HV) mapping that delivered the best performance in spite of being deadline-indifferent. A large fraction of the deadlines were missed at high loads under all the mappings, and the fact that HV prioritizes transactions by value alone ensured that high-value transactions rarely missed their deadlines. The Value-inflated Deadline (VD) mapping, which combines both values and deadlines by weighting them equally, was found (perhaps surprisingly) to behave identically to HV. Finally, the Value-inflated Relative Deadline (VRD) mapping, which equally weights *relative* deadlines and values, provided the best overall performance among the fixed-tradeoff mappings; it was almost as good as ED at low loads, and was close to HV at high loads.

For workloads that had a large spread or pronounced skew in the distribution of transaction values, the HV mapping was found to deliver the best performance throughout virtually the entire loading range. Although HV missed more deadlines than the ED mapping at low loads, the value gained by HV's ability to complete virtually all of the high-value transactions more than compensated for its missing more deadlines of low-value transactions. When transaction deadlines were linearly correlated with their execution times, the VRD mapping was found to perform especially well, outperforming even the HV mapping under high loads. This is because, in the presence of such correlation, the VRD mapping gives priority to those transactions that can return the most value in the shortest period of time. In addition to these results regarding the relative performance of the fixed-tradeoff mappings, our experiments also showed that they are susceptible to performance breakdown based on workload characteristics. For example, workload characteristics that lead a priority mapping to assign the same priority to a number of high-value transactions were shown to be quite detrimental to performance at high loads; adding a random noise component to the priority mappings alleviated this problem by constructing a total priority ordering among the transactions. Finally, it was also shown that associating a penalty with transactions whose deadlines are not met can seriously degrade the performance of some of the mappings.

As mentioned earlier, experiments were also conducted to explore the impact of data contention on the performance of the various priority mappings. These experiments were conducted with several concurrency control algorithms in order to evaluate their performance and to study their impact, if any, on our priority mapping results. The same qualitative behavior that was observed in the presence of resource contention was obtained in the data contention experiments; this was also the case when data and resource contention were combined. In earlier work [Hari90a, Hari90b], we showed that optimistic concurrency control outperforms locking in a firm real-time environment. That work employed an Earliest Deadline priority mapping and assumed that all transactions have the same value. The conclusion of the present study is that our earlier results generally carry over to the value-based RTDBS domain for all of the priority mappings that we have considered.

Finally, in addition to evaluating a variety of fixed-tradeoff priority mappings, this paper also introduced a bucket algorithm that allows the transaction value/deadline tradeoff to be varied. In the bucket algorithm, a structured approach is taken to combining these characteristics based on basic real-time scheduling principles. The actual tradeoff made between values and deadlines is controlled by a parameter of the algorithm. A series of experiments demonstrated that the algorithm can perform well at each operating point when its control parameter is set appropriately. An open problem, which is part of our near-term research agenda, is the question of how to *adaptively* change the setting of this parameter to optimize performance as the system load varies. Also on our near-term agenda is further exploration of the performance impact of workload correlations, as we expect the bucket algorithm (unlike the fixed-tradeoff mappings) to be relatively immune to correlation-related performance degradation.

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.

[Abbo90] Abbott, R., and Garcia-Molina, H., "Scheduling I/O Requests with Deadlines: A Performance Evaluation," *Proc. of the 11th Real-Time Systems Symposium*, December 1990.

[Agra87] Agrawal, R., Carey, M., and Livny, M., "Concurrency Control Performance Modeling: Alternatives and Implications," *ACM Trans. on Database Systems*, December 1987.

[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.

[Fran85] Franaszek, P., and Robison, J., "Limitations of Concurrency in Transaction Processing," *ACM Trans. on Database Systems* 10(1), March 1985.

[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.

[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.

[Huan90a] Huang, J., and Stankovic, J., "Buffer Management in Real-Time Databases", *COINS Technical Report* 90-65, Univ. of Massachusetts, Amherst, July 1990.

[Huan90b] Huang, J., and Stankovic, J., "Concurrency Control in Real-Time Database Systems: Optimistic Scheme vs. Two-Phase Locking", *COINS Technical Report* 90-66, Univ. of Massachusetts, Amherst, July 1990.

[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.

[Mena82] Menasce, D., and Nakanishi, T., "Optimistic versus Pessimistic Concurrency Control Mechanisms in Database Management Systems," *Information Systems*, vol. 7-1, 1982.

[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.