VLDB Journal, 2 117–152 (1993) Fred J. Maryanski, Editor

Value-Based Scheduling in Real-Time Database Systems

Jayant R. Haritsa, Michael J. Carey, and Miron Livny

Received May 15, 1991; revised version received June 23, 1992; accepted October 6, 1992.

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 research on real-time database systems has focused on systems where all transactions are assigned the same value, the performance goal being to minimize the number of missed deadlines. When transactions are 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 this article, 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 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.

Key Words. Transaction values and deadlines, priority mapping, resource and data contention, priority and concurrency algorithms.

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*. Accordingly,

---

Jayant R. Haritsa, Ph.D., is currently Assistant Professor, Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore–560012, India. When this work was done he was with the Computer Sciences Department, University of Wisconsin–Madison. Michael J. Carey, Ph.D., is Professor; Miron Livny, Ph.D., is Associate Professor, Computer Sciences Department, University of Wisconsin, Madison, WI 53706.

a higher quality of service is associated with processing transactions before their deadlines as compared to completing them late. In contrast to conventional DBMSs, where the goal usually is to minimize response time, the emphasis here is on meeting transaction deadlines. RTDBSs thus have the task of enforcing data integrity constraints and satisfying transaction time constraints (Stankovic and Zhao, 1988; Buchmann et al., 1989).

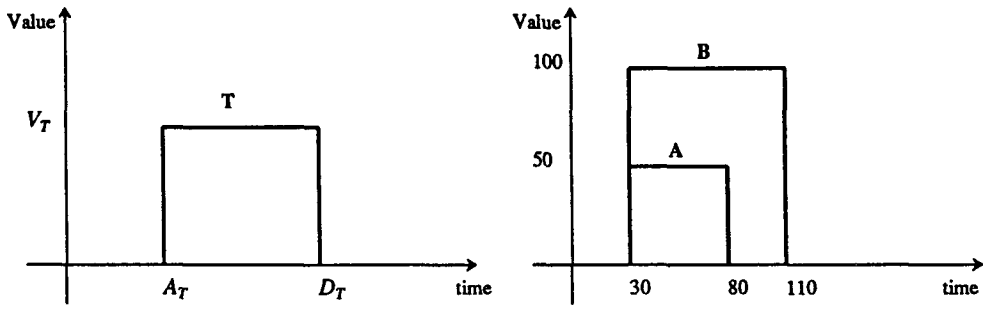
In RTDBSs, an application may assign a *value* to a transaction to reflect the return that the application expects to receive if the transaction is completed before its deadline (Huang et al., 1989).[1] The sum of the values of all input transactions constitutes the *offered value,* while the sum of the values of the transactions that are completed before their deadlines constitutes the *realized value.* The goal of an RTDBS is to maximize the realized value, because this metric is a direct measure of the real-time support provided to the application (Jensen et al., 1985). Most research on RTDBS performance has focused on applications where all transactions are assigned the *same* value (Abbott and Garcia-Molina, 1988; Haritsa et al., 1990*a*). For such a framework the goal of maximizing the realized value is equivalent to minimizing the number of missed deadlines. The concern is how *many* transactions are missed, not *which* transactions are missed. There are certainly real-time applications, however, where different transactions may be assigned different values (Stankovic and Zhao, 1988; Huang et al., 1989). The value realized by a database system supporting such applications depends on which transactions meet their deadlines.

To clarify the notion of transactions having different values, 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, because the high-priced ticket generates greater revenue. In this scenario 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. A key point to note here is that value and deadline are fundamentally different properties (Biyabani et al., 1988; Huang et al., 1989). 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. The value reflects the transaction's *worth,* while the deadline reflects the transaction's *urgency.*

In this article, we address the issue of resource scheduling and concurrency control in RTDBSs 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, 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 by Jensen et al. (1985) and Locke (1986). 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

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 only transactions that have zero value after their deadline.

Figure 1*a*. Transaction value function. **1*b*. Priority order dilemma.**



function shown in Figure 1*a*. 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.

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. This 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 (ED) and Highest Value (HV), can be used to guide the priority ordering. The ED principle is that transactions with closer deadlines should be executed first (i.e., given higher priority) because delaying them might cause their deadlines to be missed and result in their value being lost. The HV principle is that transactions with higher values should be executed first because it would be beneficial to make certain that their deadlines are met and thereby realize high values.

When transactions have similar deadlines, it is obvious that the HV principle should guide the priority ordering. Conversely, when transactions have similar values, it would seem that the ED principle provides the right priority ordering. But when transactions differ in both value *and* deadline characteristics, it is not obvious which of the above principles should be followed. Consider the simple scenario where a pair of transactions, A and B, with value functions as shown in Figure 1*b*, compete for service. Following the ED principle would result in priority ordering (A, B) while the HV principle would yield (B, A). In a more general sense, deciding on a priority ordering requires value and deadline to be weighted in some fashion, i.e., 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, $P_T = D_T/V_T$, a priority mapping that gives equal weight to value and deadline, would result in the priority ordering (B, A) of the two transactions (smaller P_T values indicate higher system priority).

Our problem is to identify the priority combination between transaction value and deadline that results in the maximum system-realized value. This is not a simple task because there is no obvious "right" combination. In fact, the combination that results in the best performance may not be the same in all circumstances, but rather

a function of the workload and system characteristics. In this article, we investigate this issue by using a detailed simulation model of a real-time database system. We evaluate the performance of several priority mappings which combine value and deadline in different ways.

In our study we investigate 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 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. 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.

For the most part, we restrict our attention to transactions with step-shaped value functions of the type shown in Figure 1*a* (the only exception is the penalty scenario). This means that transactions that miss their deadline can be immediately discarded, because 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 because such information is not available in most cases.

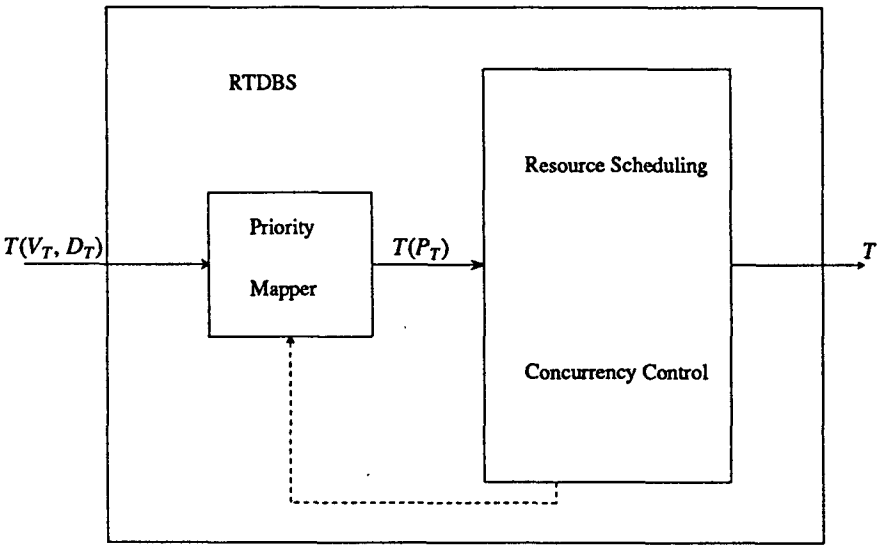
Our simulation model captures the modular architecture shown in Figure 2. The *priority mapper* unit generates a priority for a transaction on its arrival, and this priority is subsequently used throughout the system. It is possible that there may be feedback in the priority assignment process, causing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

The remainder of this article is organized as follows: Section 2 reviews related work on integrating value and deadline. Section 3 describes several different priority mappings and also the "bucket" mechanism. Section 4 describes the functioning of concurrency-control algorithms evaluated in the study. Section 5 presents our RTDBS model and its parameters, while Section 6 highlights results of the simulation experiments. Finally, Section 7 summarizes the main conclusions and outlines future research avenues to explore.

2. Related Work

In this section we review some of the related work in algorithms for integrating value and deadline. We discuss this work from the perspective of both real-time operating systems and real-time database systems.

Figure 2. Priority Architecture



2.1 Real-time operating systems

Thus far, research on integrating value and deadline has been almost exclusively restricted to the context of real-time operating systems. The first such study (Jensen et al., 1985) was developed as part of the CMU Archons project. A well-constructed "Best-Effort" priority mapping mechanism for integrating value and deadline was developed and shown to outperform several "classical" priority mappings. The Best Effort algorithm is based on two observed value function and scheduling characteristics: (1) Given a set of processes whose deadlines can all be met by some schedule, it can be shown that a uniprocessor schedule in which the earliest deadline is scheduled first always meets all deadlines and; (2) Given a set of processes with values (ignoring deadlines), it can be shown that a schedule organized in decreasing order by *value density* (value/execution time), produces a total value at each point in time that is at least as high as any other schedule.

The algorithm operates in the following manner: The scheduler creates a deadline-ordered sequence of the available tasks, which is then sequentially checked for its probability of missing the next task's deadline. At any point in the sequence where this overload probability passes a preset user-defined threshold, the task prior to the overload with the minimum value density is removed from the sequence. The removal process is repeated until the overload probability is reduced to acceptable levels. This procedure finally results in a sequence of processes in deadline order that does not cause an overload condition. In essence, the Best Effort algorithm creates a feasible schedule of the tasks with the highest value-densities, incorporating both observations described above. A simulation-based performance study of the

algorithm showed that, for appropriate settings of the overload probability threshold, the Best Effort algorithm provides a high realized value under a wide spectrum of loads, including both transient and persistent overloads, and a variety of value functions.

A different approach to integrating value and deadline was described in Biyabani et al. (1988). The basic mechanism here is that the scheduler first schedules a new task according to its deadline without considering its value. If the task cannot be executed before its deadline without jeopardizing the completion of previously scheduled tasks, the scheduler then tries to schedule the task by removing previously scheduled *lesser-valued* tasks. If it is not possible to schedule the new task even after the removal of all lesser-valued tasks, the new task is discarded and the removed tasks are reinstated. A simulation-based study showed that algorithms based on this approach performed better than either exclusively deadline-based or value-based priority assignments.

Theoretical studies of *uniprocessor* scheduling algorithms have been made for the special case where tasks have no slack time (i.e., deadline = execution time + arrival time). These studies focused on identifying the guarantee that a scheduling algorithm can provide with respect to how well it performs compared to a clairvoyant scheduler on any sequence of task requests. The task value model is that each task's value is a user-defined multiple (called "importance") of its execution time. For underloaded systems, it was shown in Dertouzos (1974) that the Earliest Deadline algorithm achieves 100% of the input task value because it meets all the task deadlines. For overloaded systems, however, it was proved in Baruah and Rosier (1991) that no on-line uniprocessor-scheduling algorithm can guarantee a value more than $1/(1+\sqrt{k})^2$ of that obtained by a clairvoyant scheduler, where k is the ratio between the maximum and minimum task importance. Recently an algorithm called D^{over} was presented in Koren and Shasha (1992); this algorithm achieves optimal (in the above sense) on-line performance under both underloaded and overloaded conditions.

In order to use any of the above schemes or results, a-priori knowledge of task service requirements is required. Unfortunately, knowledge about transaction resource and data requirements is usually unavailable in database applications (Stankovic and Zhao, 1988), and the schemes therefore cannot be used directly in most real-time database systems. Consequently, we are forced to consider alternative methods for scheduling value-differing transactions in real-time database systems.

2.2 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 RTDBSs. All these studies consider RTDBSs that operate under either firm or soft deadlines. (It is generally considered that hard deadline RTDBSs are infeasible because it is difficult to determine beforehand the computation time and execution pattern of a transaction; Abbott and Garcia-Molina, 1988; Stankovic and Zhao, 1988.) The studies can be divided into two general groups: those that treat all transactions as equally important (Abbott and Garcia-Molina, 1988, 1989,

1990; Haritsa et al., 1990*a*, 1990*b*; Huang and Stankovic, 1990*a*), and those that incorporate the notion of transactions having different values (Huang et al., 1989; Huang and Stankovic, 1990*b*). A brief summary of the studies in these two groups is presented as follows.

The problem of scheduling transactions in RTDBSs was first addressed by Abbott and Garcia-Molina (1988, 1989). Their work focused on evaluating the performance of various real-time scheduling policies, all of which enforced data consistency by using a two-phase locking protocol as the underlying concurrency control mechanism. In Haritsa et al. (1990*a*), the focus was shifted to studying the performance of optimistic and pessimistic methods of concurrency control in a real-time environment. This work was extended in Haritsa et al. (1990*b*) 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 Huang and Stankovic (1990*a*). In Abbott and Garcia-Molina (1990) 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 that incorporated transaction value in their performance evaluation framework are Huang et al. (1989) and Huang and Stankovic (1990*b*). Using a basic locking scheme for concurrency control, Huang et al. (1989) investigated several algorithms for resource scheduling and data conflict resolution. Huang and Stankovic (1990*b*) extended this work 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, that leave room for further investigation: First, only fixed tradeoffs between value and deadline were considered. Second, the range of values these transactions could take on was limited and the value distribution was uniform. Third, the concurrency control algorithms compared in Huang and Stankovic (1990*b*) are priority-indifferent flavors of two-phase locking and optimistic concurrency control. Finally, testbed limitations constrained the studies to model a closed queuing system with a fixed amount of resources.

Our work differs from the research mentioned above in that we consider a variety of transaction workloads with different degrees of spread and skew in transaction values. 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. This mechanism is based on principles similar to those used in the construction of the Best Effort algorithm described earlier (Jensen et al., 1985), but suitably modified to account for the lack of detailed knowledge of transaction characteristics in RTDBSs.

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 are distinguished by both value and deadline, the priority mapping has to take both of these characteristics into account. In earlier studies (Stankovic and Zhao, 1988; Huang et al., 1989; Huang and Stankovic, 1990*b*) several priority mappings that combine ED and HV 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 are used to denote the arrival time, deadline, value, and priority 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 ED mapping follows the ED principle. The transaction priority assignment is $P_T = D_T$. It represents an extreme tradeoff because the value of the transaction is not taken into consideration. Several studies (Jensen et al., 1985; Abbott and Garcia-Molina, 1988) have observed that in lightly-loaded or moderately-loaded real-time systems, using an ED 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 HV mapping follows the HV principle. The transaction priority assignment is $P_T = 1/V_T$. It represents the other extreme tradeoff because the deadline of the transaction is not taken into consideration. Note that this mapping does not distinguish between transactions with 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 VD mapping combines the ED and HV principles by using the transaction priority assignment $P_T = D_T/V_T$. 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 VRD mapping is similar in flavor to VD, but it uses the *relative* deadline instead of the absolute deadline in combining the ED and HV principles. The transaction priority assignment is $P_T = D_T - A_T/V_T$. 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 combines value and deadline based on the application of two scheduling observations that are similar to those used in constructing the Best Effort algorithm of Jensen et al. (1985).

1. Given a set of transactions with deadlines that can all somehow be met, an Earliest Deadline priority ordering meets all (or most of) the deadlines.

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

The bucket 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

$$B_T = \begin{cases} \left\lfloor \frac{Pos_T * NumBuckets}{NumTrans} \right\rfloor & \text{if} \quad NumTrans > NumBuckets \\ \\ Pos_T & \text{otherwise} \end{cases}$$

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

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

where the I_T component is a randomly chosen unique integer key.[2] (Because the transaction priority is expressed as a vector, priority comparisons are made in *lexicographic* order.) The I_T 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 into a set of buckets, based on transaction value. Transactions of

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

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 so that *within* each bucket the transaction priority ordering is based on the ED principle.

By using an ED 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. If all transactions have the same value, the VRD mapping establishes a Shortest Job First priority ordering. For 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 ED mapping. This is because all transactions are assigned to the same bucket and the priority ordering within a bucket is ED. If the *NumBuckets* parameter is set to ∞, the priority mapping is similar to the HV mapping, since each transaction is usually 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 RTDBSs are preemptive-resume, based on priorities at the CPUs, and non-preemptive priority-scheduling at the disks. These policies use 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 (Gray, 1979), timestamps (Reed, 1978), and optimistic concurrency control (Kung and Robinson, 1981). 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 the RTDBS environment (Haritsa et al., 1990*a*, *b*). The selected algorithms are 2PL-HP, a prioritized locking algorithm; OPT-BC, a priority-indifferent (conventional) algorithm; and OPT-WAIT, a prioritized variant of the OPT-BC algorithm. Details of these algorithms are given as follows.

4.1 2PL-HP

In 2PL-HP, classical two-phase locking (Eswaran et al., 1976) is augmented with a *High-Priority* conflict resolution scheme (Abbott and Garcia-Molina, 1988) 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 mode, and if the requester's priority is higher than that of all lock holders, the holders are started 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 under priority assignment schemes that assign unique priority values to transactions and do not change a transaction's priority during the course of its execution.

4.2 OPT-BC

In OPT-BC, classical optimistic concurrency control (Kung and Robinson, 1981) is modified to implement the notion of a *Broadcast Commit* (Menasce and Nakanishi, 1982; Robinson, 1982). When a transaction commits, it notifies other executing transactions that conflict with it and these are immediately restarted. A validating transaction conflicts with an executing transaction if it wishes to update a data object that has been read by the executing transaction. There is no need for a validating transaction to check for conflicts with any already-committed transactions because 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 (Kung and Robinson, 1981), resulting in less wasted resources and earlier restarts; this increases the changes 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 (Haritsa et al., 1990*b*) is a variant of OPT-BC 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 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 waiter's deadline 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 priority-indifferent, for the following reason: It was shown that OPT-BC provided better performance

than 2PL-HP in high-contention firm deadline environments (Haritsa et al., 1992). 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 RTDBSs that operate with transactions of different values.

We do not include basic 2PL because its performance was always worse than that of 2PL-HP for the workloads considered in this study. The poor performance of 2PL is due to its priority-indifferent blocking policy which results in high priority transactions being blocked due to lock conflicts with low-priority transactions, a phenomenon know as "priority inversion (Sha et al., 1987). Under high contention, priority inversion results in long waiting times, causing urgent transactions to miss their deadlines.

5. RTDBS Performance Model

A detailed model of an RTDBS was used to study the performance of various priority mappings. The model is similar to that of our earlier studies (Haritsa et al., 1990*a*, 1990*b*). In this model, the database system consists of a shared-memory multiprocessor operating on disk-resident data (for simplicity, we assume that all data that is accessed from disk and buffer pool considerations are therefore ignored). The database itself is modeled as a collection of pages.

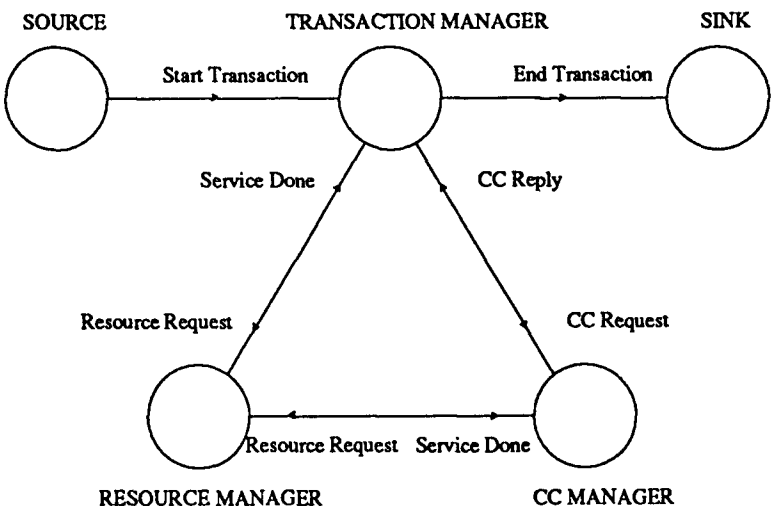
Transactions arrive in a Poisson stream and each transaction has an associated value and deadline. A transaction consists of a sequence of read and write page operations. A read operation involves a concurrency control request to get access permission, followed by a disk I/O to read the page, followed 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. Here we assume that the RTDBS has sufficient buffer space to retain updates until commit time. We also assume the use of a log-based recovery scheme where only log pages are forced to disk prior to commit. 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 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 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

Figure 3. RTDBS model structure.



transaction varies uniformly between 0.5 and 1.5 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% of the input transactions and accounts for 80% 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, 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. A setting of 50 for this parameter would specify a range of values between ±50% 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.

Table 1. Workload model parameters.

Parameter	Meaning
ArrivalRate	Transaction arrival rate
DatabaseSize	Number of pages in database
PageCount	Average no. 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*

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, which is used for most of the experiments reported here, is:

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

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* (i.e., it determines the tightness/slackness of deadlines) that varies uniformly over the range set by the workload parameters *LSF* and *HSF*.

The second deadline formula used in the study is:

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

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. The workload parameters *LSF* and *HSF* are set to the same number and *SF* takes on this value. (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

Table 2. Resource model parameters

Parameter	Meaning
NumCPUs	Number of processors
NumDisks	Number of disks
PageCPU	CPU time for processing a data page
PageDisk	Disk service time for a page

correlation, and makes a transaction deadline *linearly* correlated to its execution time. With DF2, all transactions have the same *slack ratio*, which is defined as $(D_T - A_T)/R_T$. 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.

In order to not generate "degenerate" transactions (i.e., transactions whose deadlines cannot be met even it they are executed alone in the system) deadlines have to be assigned so that each transaction has a slack ratio of at least 1. We ensure this in our experiments by always setting the values of the *LSF* and *HSF* parameters to be >1.

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 service discipline is preemptive-resume, with the preemption 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 are modeled as uniformly distributed across all of the disks.

6. Experiments and Results

In this section, we present performance results of our experiments, comparing various priority mappings in an RTDBS environment. The simulator used to obtain the results is written in Modula-2-based DeNet simulation language (Livny, 1988). 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.

6.1 Performance Metrics

The primary performance metric is *LossPercent*, 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. *LossPercent* values in the range of 0–20% are taken to represent system performance under "normal" loadings, while those in the range of 20–100% represent performance under "heavy" loading. A long-term operation region where the loss percentage is large is obviously unrealistic for a viable RTDBS. Exercising the system to high loss levels, however, provides valuable information on the response of the algorithms to brief periods of stress loading (Abbott and Garcia-Molina, 1988, 1989). (All *LossPercent* graphs in this article show mean values with relative half-widths about the mean of $<5\%$ at the 90% confidence interval, with at least 5,000 transactions processed for each experiment. Only statistically significant differences are discussed here.)

A secondary performance metric, *MissPercent*, measures the percentage of transactions that do not complete before their deadline. Note that when all transactions have the same value, the *LossPercent* and *MissPercent* 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 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 (Franaszek and Robinson, 1985; Agrawal et al., 1987), i.e., where there is no queuing for resources. This was done by increasing twenty-five-fold the number of processors and disks, from their baseline values of 8 and 16 to 200 and 400, respectively. While abundant resources usually are not to be expected in conventional database systems, they may be more common in RTDBS environments because 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 (the values are taken from the real number domain and are not simply integers). 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. *NoPriority (NP)*: All transactions are given the same priority in this mapping. The performance obtained under this mapping should be interpreted as that which would be observed if the RTDBS were to be replaced by a conventional DBMS and the feature of discarding late transactions was retained.

2. *RandomPriority (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.

6.3 Resource Contention (RC)

Our first set of experiments investigated the performance of priority mappings when resource contention is the sole performance-limiting factor. We began our experiments by developing a baseline model around which further experiments were constructed by varying a few parameters at a time. The settings of the workload 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 because all transactions are *queries*. There is a single transaction class. Transaction values range between 50.0 and 150.0. Deadline formula DF1, which makes transaction deadlines 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, $HSF/LSF = 4.0/1.33$, is the same as the spread in value, 150.0/50.0, namely 3. These settings ensure that variations in both deadline and value play a role in determining overall system performance.

6.3.1 Baseline Model. For the baseline model, Figures 4*a* and 4*b* show the *LossPercent* results under normal and heavy load conditions. Figures 4*c* and 4*d* show the corresponding *MissPercent* results. (The curves for HV and VD are identical in all of these figures.) From this set of graphs it is clear that, at low loads, the ED mapping realizes the most value (smallest *Loss Percent*). This might be considered surprising because ED is a value-indifferent mapping. However, because ED misses the deadlines of very few (if any) transactions (Figure 4*c*), it delivers most value. The value-cognizant mappings, HV and, to a lesser extent, VRD focus their effort on completing the high-value transactions. In the process, they prevent some lower-value transactions from making their deadlines, thus losing more offered value.

As the system load increases the performance of ED steeply degrades, nearly paralleling that of NP. At high loads, where the resources become saturated, transactions under ED and NP make progress at similar *average* rates. Under NP, every transaction makes slow but steady progress from the moment of arrival. Under

Table 3. Baseline model workload settings

Parameter	Value
DatabaseSize	1,000 pages
PageCount	16 pages
WriteProb	0.0
DeadlineFormula	DF1
LSF	1.33
HSF	4.0
GlobalMeanValue	100.0
NumClasses	1.0
ProbClass [i]	1.0
OfferValue [i]	1.0
MeanValue [i]	100.0
SprdValue [i]	50.0

Table 4. Baseline model resource settings

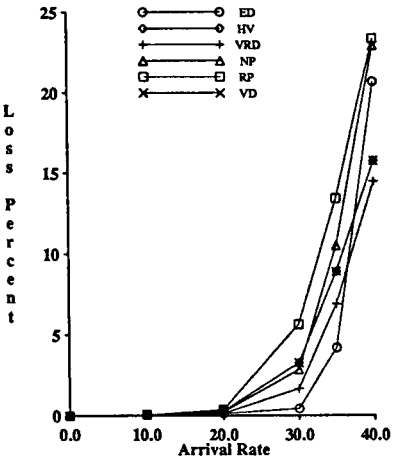
Parameter	Value
NumCPUs	8
NumDisks	16
PageCPU	10 ms
PageDisk	20 ms

ED, no progress is made initially by a transaction, but as its deadline approaches, fast progress is made. The net progress is about the same under both ED and NP. This was experimentally confirmed by measuring the average progress made by transactions that missed their deadline. Therefore, under overload conditions, ED is not the right mapping to use (Jensen et al., 1985; Huang et al., 1989).

The RP mapping behaves poorly at low loads but performs better than ED at high loads. Under ED new transactions usually start off at low priority and become high priority 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 when they arrive. Such transactions tend to make their deadlines; therefore there is always some fraction of transactions in the system that are guaranteed to make their deadlines. This explanation is confirmed by the higher *MissPercent* characteristics of ED compared to RP at high loads.

Figure 4. RC baseline model

a. **Normal load**



b. **Heavy load**

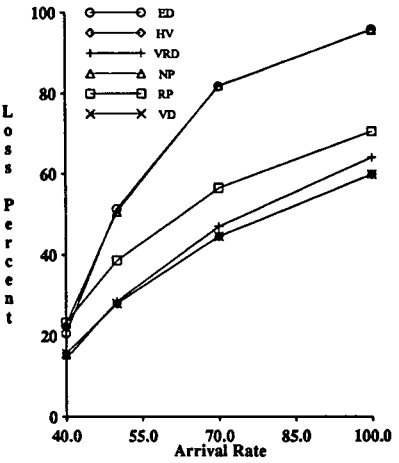
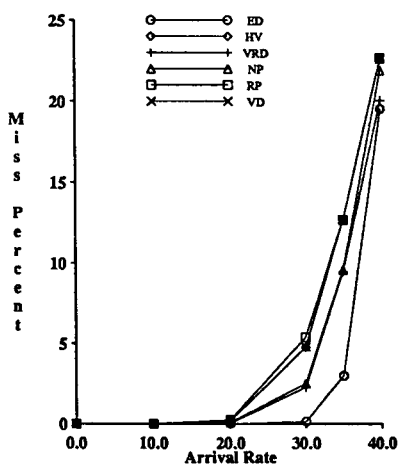
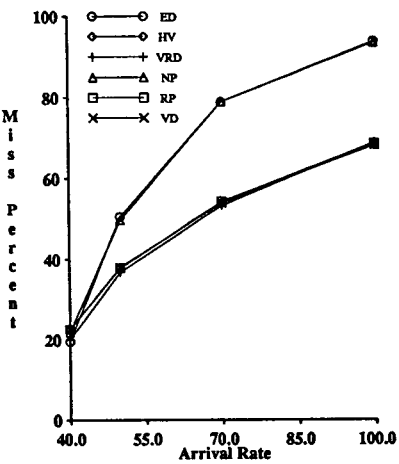


Figure 4. MissPercent (RC baseline)

c. **Normal load**



d. **Heavy load**



The HV mapping also performs worse than ED at low loads, but improves as the load increases. In fact, it outperforms all the other algorithms at high loads because the system has sufficient resources to handle only a fraction of the transactions in the system. At high loads, only transactions that can deliver high value should be run. If we look at the *MissPercent* characteristics (Figures 4*c* and 4*d*), we observe that HV and RP behave identically with respect to this metric. The reason for this behavior is that the workload has transaction values that are independent of other

transaction characteristics and all transaction values are distinct. In such a case, HV priority ordering is no different from RP priority ordering in terms of the ability of the RTDBS to make transaction deadlines. This would not be the case if there were groups of transactions with the *same* value, because same-value transactions introduce NP-type behavior into the performance of HV.

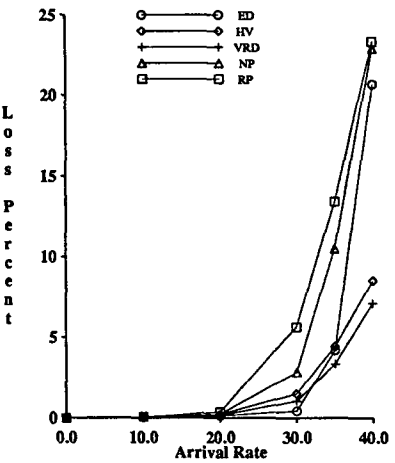
The VD mapping appears to combine ED and HV principles in its priority assignments, but it performs identically to the HV mapping. This is not a coincidence, but is, in fact, always true: As time progresses, the D_T term in D_T/V_T becomes large enough that it is approximately the same for all transactions. 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 article. 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 Huang et al. (1989), 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; thus the priority assignment degenerates to an HV mapping. Therefore, the actual weights should not be expected to impact the long-term performance of this mapping.

The VRD mapping's performance is between 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, it delivers the best overall performance. 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 ED effect, i.e., jobs with small relative deadlines are given priority over jobs with larger deadlines. Among sets of similar-valued jobs that arrive about the same time, the priority ordering is approximately ED. Due to this effect, fewer deadlines are missed by VRD when compared to HV at low loads (Figure 4*c*). Conversely, at high loads under the VRD a high-value transaction may not be completed due to having a large relative deadline.

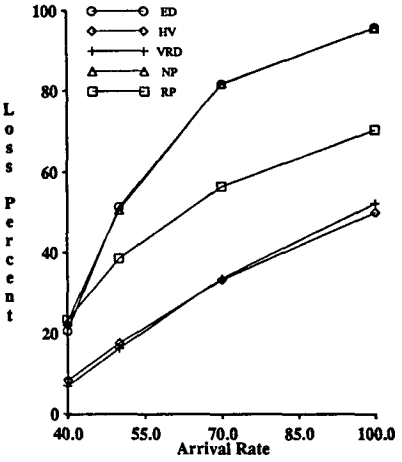
Our next experiment examined the effect of increased spread in transaction values. The *SprdValue* parameter was increased from the baseline value of 50% to 99%, while keeping the other parameters the same as in the baseline model. This means that transaction values now ranged between 1.0 and 199.0. The *LossPercent* results for this experiment are shown in Figures 5*a* and 5*b*. 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; therefore changes in the value distribution do not affect their performance (if 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 missed

Figure 5. Increased spread

a. **Normal load**



b. **Heavy load**



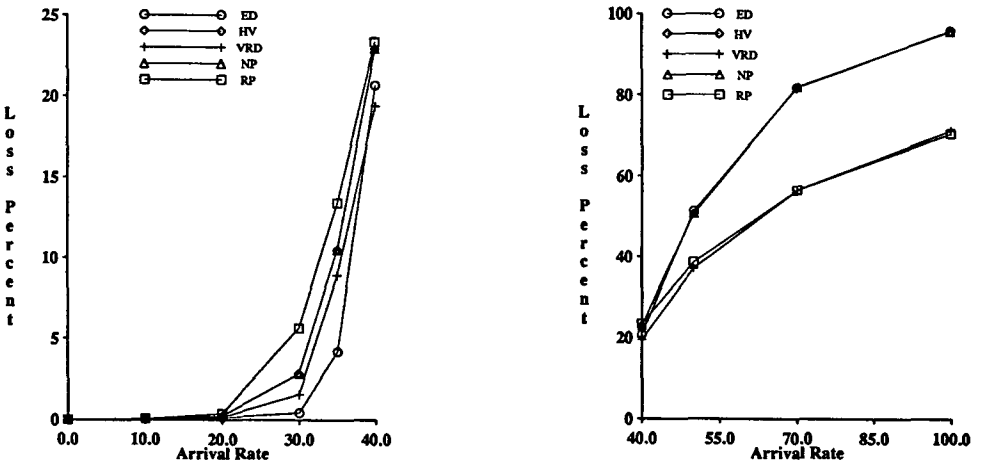
low-value transactions have lesser effect on the realized value because their average value is smaller due to the increased spread. Note that the *MissPercent* characteristic of HV is the same as in the baseline experiment because the workload assigns values to transactions independently of their other characteristics.

We then examined the effect of decreasing the spread of transaction values. The *SprdValue* parameter was set to 0%, 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 *LossPercent* results for this experiment are shown in Figures 6*a* and 6*b*. (In Figure 6*b* the top set of lines corresponds to ED, HV, and NP, while the bottom set corresponds to VRD and RP.) The value-cognizant mappings, HV and VRD, perform worse here when compared to the baseline experiment. HV behaves just like NP because it gives every transaction the same priority when all values are the same. This is an extreme case, but similar problems would arise when the workload consists of multiple transaction classes where all transactions within a class have the same value. The VRD mapping does not behave like NP because its relative deadline component ensures a priority ordering among the transactions. Also, at high loads the VRD behaves similarly to RP rather than ED. This implies that VRD is more value-oriented than deadline-oriented at high loads because the relative deadline component has only a randomizing effect when all values are the same.

Another interesting observation is that the RP mapping 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. If even an infinitesimally small noise is added to the transaction priorities generated by the HV mapping, the heavy load performance would be like that of RP rather than that of NP. This is because the addition of

Figure 6. Decreased spread

a. **Normal load** *b.* **Heavy load**



noise causes a priority ordering to exist where there was none. The noise should be random, not based on transaction characteristics. If transaction deadlines are used to generate noise, the performance at high loads would be like that of ED, not RP. It should be noted that the high-load stability obtained by the addition of noise is gained at some cost in normal load performance, because RP performs worse than NP in this loading range.

We can draw the following conclusions for the uniform workloads examined in this section:

1. At low loads, when the *MissPercent* is low, the ED priority ordering is the right choice. At high loads, when the *MissPercent* is high, the priority ordering given by the HV principle realizes the most value.

2. 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.

3. The use of absolute deadlines in priority assignments should be handled with care.

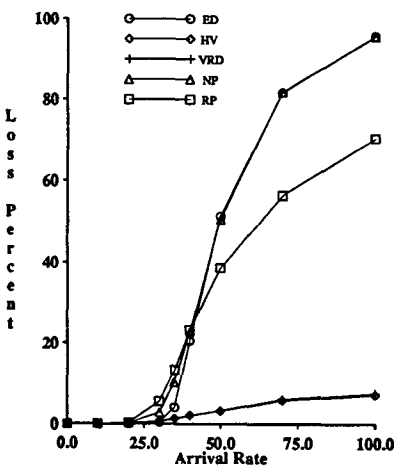
4. Priority mappings should have a built-in noise factor to guard against the possibility of transactions having identical priorities because such transactions can hinder each other's progress and thus degrade performance at high loads.

6.3.2 Transaction Value Skew. The next experiment examined the effect of skew in transaction value distribution. The parameters are set as shown in Table 5.

Table 5. Skew workload settings

Parameter	Value
DatabaseSize	1,000 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

Figure 7. RC value skew



They construct a two-class workload where 10% of the transactions deliver 90% of the offered value. The transaction values from the first class vary between 450 and 1,350, while the values of the second class vary between 5.5 and 16.5. (The *GlobalMeanValue* parameter is the same as for the baseline experiment.) *LossPercent* results are shown in Figure 7. As in previous experiments, the performance of the ED, RP, and NP mappings remains the same as in the baseline experiment, because these mappings are value-indifferent. The figure also shows that the performance

Table 6. Data contention resource settings

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

of the value-cognizant mappings, HV and VRD, improves greatly over the baseline; they are now much superior to the value-indifferent mappings. Even at low loads, they perform almost as well as ED. By making sure that all of the (few) high-value transactions make their deadline, HV and VRD ensure that they always realize at least 90% of the offered value. In addition, the value of the missed transactions constitutes a very small fraction of the total value at low loads; the performance impact of a higher number of missed deadlines than ED is therefore negligible. The performance of VRD is almost identical to that of HV. 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 very similar to that of the HV mapping, and is only marginally deadline-cognizant.

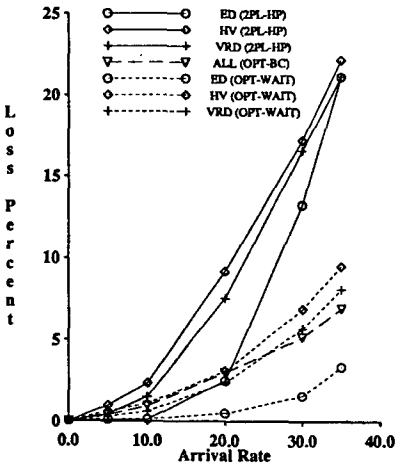
We can conclude that skew in transaction values causes the value-cognizant algorithms to perform much better. For workloads with a considerable spread in transaction values, the priority ordering established by the HV principle ensures good performance through the entire loading range. These results 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 priority mappings when data contention is the sole performance degradation factor. As before, we began our experiments by developing a baseline model around which we 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 Resource Contention (Table 3), except that the *WriteProb* parameter is set to 0.25 instead of 0.0. Deadline formula DF1 is used again 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 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 and NP mappings in the following sections.

Figure 8. DC baseline model

a. **Normal load**



b. **Heavy load**

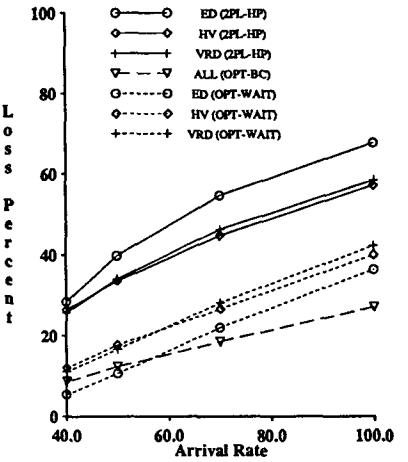
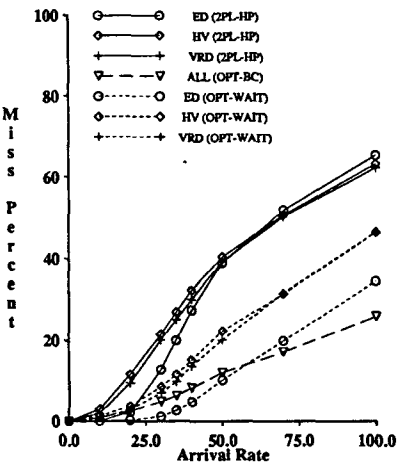


Figure 8*c. MissPercent* **(DC Baseline)**



6.4.1 Baseline Model. For the baseline model, Figures 8*a* and 8*b* show *LossPercent* results under normal and heavy load. Figure 8*c* shows the corresponding *MissPercent* behavior. The results were obtained separately with 2PL-HP, OPT-BC, and OPT-WAIT concurrency control algorithms. The 2PL-HP mappings (solid lines) *qualitatively* exhibit the same behavior as in the RC baseline model (Figures 4*a*, 4*b*). The ED mapping performs the best at low loads, while HV outperforms all the other algorithms at high loads. Data contention (unlike resource contention) is not work conserving; already-performed work has to be redone after a transaction restart. Therefore, ensuring that the most urgent transactions are given highest priority is even more beneficial at low loads here. At high loads, following the HV principle is again the right approach because the data contention level is high enough that

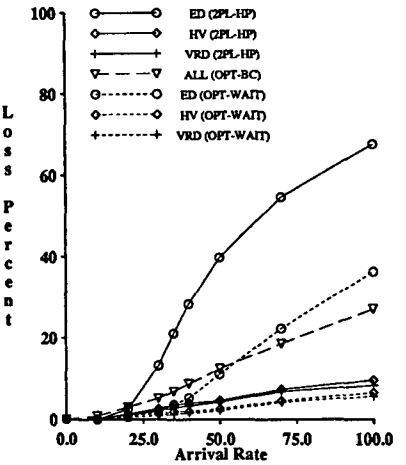
only a fraction of transactions are able to complete before their deadlines. In such a situation the transactions that should be given priority are those that can deliver high values.

All the priority mappings under OPT-BC (dashed line) behave exactly the same, because 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 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 *MissPercent* characteristics, where we observe that OPT-BC misses far fewer deadlines than 2PL-HP (Figure 8*c*). 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 (Haritsa et al., 1990*a*). 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.

OPT-WAIT (dotted lines) performs worse than OPT-BC for all mappings except ED at low loads because of priority waiting. Priority waiting is a good idea with ED because 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, which causes a steep increase in the number of conflicts. A more detailed explanation of OPT-WAIT behavior is given in Haritsa et al. (1990*b*). Although that study did not include transaction values, the explanations carry over because ED is a value-indifferent mapping. With the ED priority mapping, a waiting transaction never has to wait beyond its deadline. For other mappings, this is not necessarily the case. Under HV, for example, it is clear that the higher-value conflicting transactions may not have completed by the waiting transaction's deadline. In such a case the waiter is aborted and its value is 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 have many *wasted sacrifices*, i.e., cases where a transaction is discarded on behalf of another transaction that later does not complete. It should be noted that the performance of OPT-WAIT is still better than that of 2PL-HP for all mappings throughout the entire loading range.

6.4.2 Transaction Value Skew. This experiment examined the effect of skew in transaction value distribution. Workload parameters are the same as for experiment 6.3.2 (Table 5), except that the *WriteProb* parameter is set to 0.25. Resource parameter settings for this experiment are shown in Table 6. *LossPercent* results are shown in Figure 9 for the 2PL-HP, OPT-BC, and OPT-WAIT concurrency control algorithms. The performance of the value-cognizant mappings improves tremendously under 2PL-HP and they are now far superior to the ED mapping, as in the pure resource contention case. 2PL-HP ensures that the highest priority transactions are virtually guaranteed to make it to their deadline. Successfully meeting deadlines of the few high-value transactions is, by itself, sufficient to realize at least 90% of the offered value.

Figure 9. DC value skew



The performance of OPT-BC (dashed line) remains the same as in the baseline data contention experiment (6.4.1), because OPT-BC does not take into account transaction values; therefore changes in the transaction value distribution do not affect its performance. Figure 9 shows that the performance of a value-cognizant mapping under 2PL-HP is superior to its performance under the OPT-BC algorithm, in spite of 2PL-HP having a much higher *MissPercent*. The value that 2PL-HP derives from concentrating on the high-value transactions more than compensates for the value lost due to missing the deadlines of a large number of low-value transactions. OPT-BC treats all transactions equally, which can cause high-value transactions to be restarted (and therefore miss their deadlines) due to the commits of low-value transactions.

All the mappings performed better for OPT-WAIT (dotted lines) than for 2PL-HP, including the value-cognizant algorithms. OPT-WAIT is priority-cognizant and is willing to sacrifice low-priority transactions for high-priority transactions, similar to 2PL-HP. In addition, OPT-WAIT gains some extra value due to missing the deadlines of a smaller number of low-value transactions. OPT-WAIT makes all of the same high-value transactions and misses fewer low-value transactions.

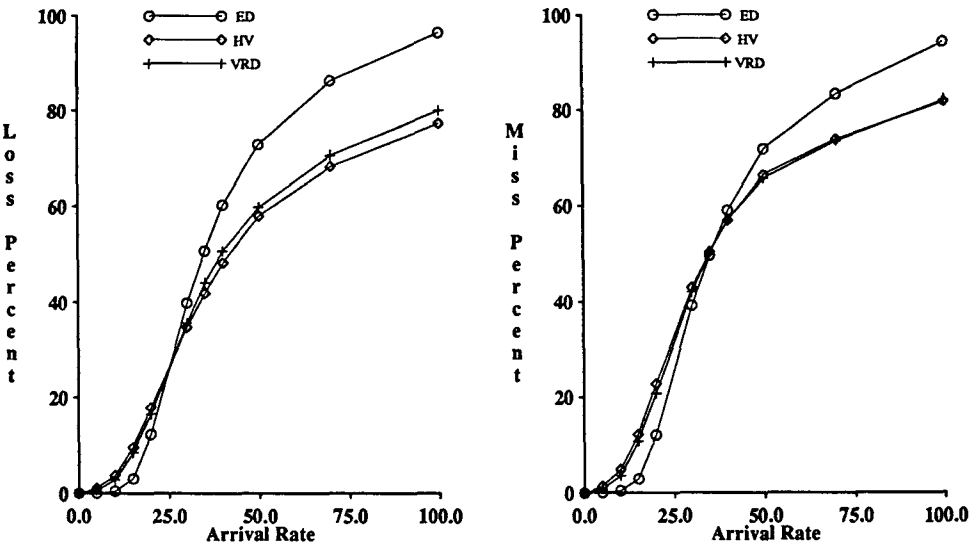
With increasing data contention, optimistic algorithms outperform locking algorithms in firm-deadline systems (Haritsa et al., 1992), assuming that all transactions have the same value. Our experiments here demonstrate that optimistic algorithms can also perform better than locking algorithms when the 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 RC and DC contribute towards system performance degradation. This was done using limited hardware resources (8 CPUs

Figure 10. DC + RC (2PL-HP)

a. **Baseline model** *b. MissPercent*



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 RC or DC alone. The ED mapping is the best at normal loads, while the HV mapping is the best at high loads. The performance of the value-cognizant mappings improves with value spread or skew. The performance of priority mappings for 2PL-HP under the baseline workload model is shown in Figures 10*a* and 10*b*. The results are qualitatively similar to those of Figures 4*a–c* or Figures 8*a–c*.

We also conducted this experiment using the two optimistic algorithms, OPT-BC and OPT-WAIT. Both outperformed 2PL-HP over virtually the entire loading range for all mappings. At low loads, OPT-WAIT did slightly better than OPT-BC for the ED mapping, and slightly worse for the other mappings (similar to the results of the baseline pure data contention experiment, 6.4.1). At high loads, OPT-WAIT and OPT-BC had the same performance for all mappings. The reason 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.

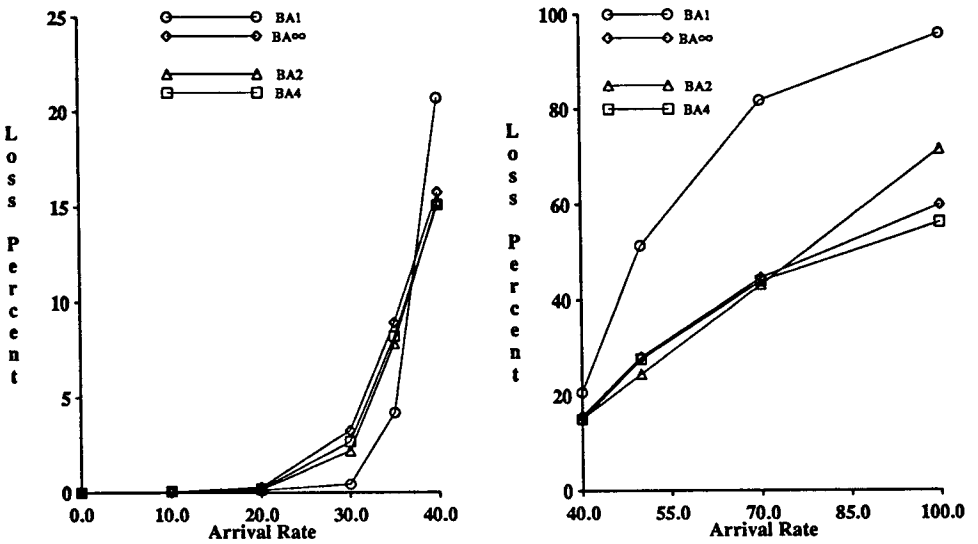
The conclusion is that both resource and data contention affect mapping performance in similar fashion, so results are qualitatively the same for both contention types. These results can also be traced to the fact that we use priority in a consistent fashion for both resource scheduling and concurrency control.

6.6 Bucket Algorithm (BA)

In this section we present results for experiments that evaluated the performance of the bucket algorithm. Bucket mapping is evaluated for four settings of the

Figure 11. Bucket RC baseline

a. **Normal load** *b.* **Heavy load**



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 orderings similar to those of the ED and HV mappings, respectively. These curves help to put the results in perspective with those described in the previous sections. Note that BA∞ generates similar 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 its random noise component. This protects BA∞ from the poor performance behavior observed with HV when all values are the same (see experiment 6.3.1).

All of the experiments we carried out for the fixed-tradeoff mappings were also performed for the bucket algorithm. Due to space limitations, we will present the bucket algorithm performance in detail for only a subset of the experiments. The experiments discussed here are the baseline resource contention 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 11*a* and 11*b* show the *LossPercent* behavior of the bucket algorithm under normal and heavy loads. At low loads up to an arrival rate of 35.0, the BA1 (ED) mapping performs best. As the load increases, 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.

When the loading level is extremely high, a bucket count of ∞, which corresponds to the BA∞ (HV) mapping, will provide the best performance. Within a bucket the priority ordering is ED. Therefore, the bucket count has to be at a level such that the miss percentage in the first bucket is small enough for the ED policy to work well. At low loads, a single bucket is sufficient because 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 several lower value transactions whose deadlines it could have made. If the mix is too "thick" (too few buckets), the system may spend resources on transactions with low values and lose high-value transactions. Therefore, there is a bucket count at each operating point that delivers the "right" mix and the "right" tradeoff between value and deadline. With the appropriate choice, the bucket algorithm generates superior performance to all the other mappings (see Figures 4*a*, 4*b*) over the entire loading range. Of course, an additional adaptive mechanism is still required so that the bucket algorithm may dynamically change the number of buckets to match the system loading level.

6.6.2 Transaction Value Skew. This experiment examined the effect of skew on the performance of the bucket algorithm. It was conducted for the 10-90 workload (Table 5), where 10% of the transactions offer 90% of the value. The results in Figures 12*a* and 12*b* show that all 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 mapping deteriorates from an arrival rate of 30.0 onwards; BA2 does poorly after an arrival rate of 50.0; 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). The results show that, when there is considerable skew in the value distribution, the BA∞ (HV) mapping is the priority ordering of choice, just as we saw before. Although it misses more deadlines than other algorithms, BA∞'s poor performance on this front is compensated for by the high values of the transactions that it does complete, even at low loads. Note, however, that if the load were heavy enough that 90% of the transactions missed their deadlines, then a policy like BA10 (which splits transactions into 10%-sized buckets) should be expected to perform better than BA∞. BA10 would be using ED among the high-value transactions in the first bucket, while BA∞ would be using HV. Because ED is better than HV for a set of transactions that can be completed by their deadlines, BA10 could be expected to outperform BA∞.

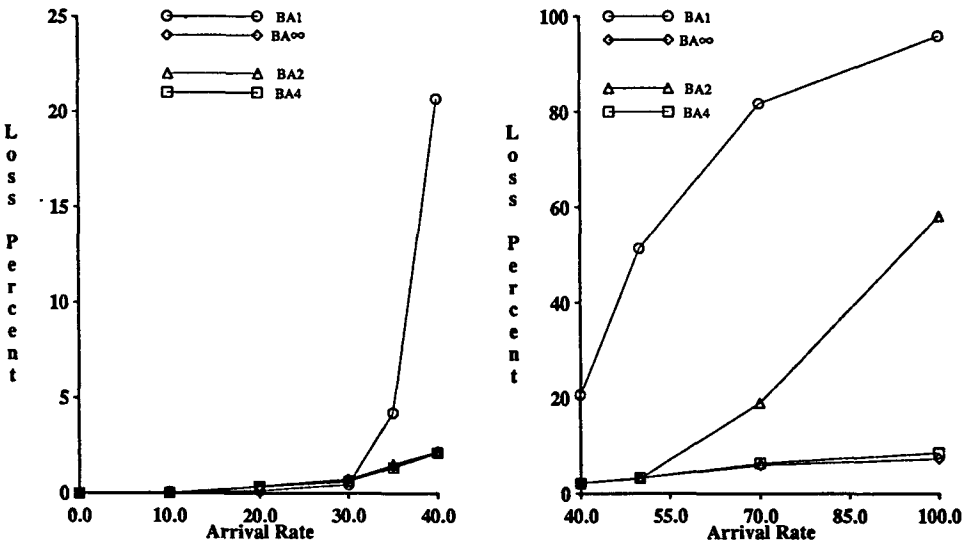
The main conclusion here is that, by layering transactions based on value, and then using ED within each bucket, the bucket algorithm exhibits a structured and logical approach towards the objective of maximizing the realized value. For each workload there is a "right" bucket count that delivers good performance.

6.7 Late Penalty

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 example, in

Figure 12. Bucket RC value skew

a. **Normal load** *b.* **Heavy load**



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. The 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. (Note that when missed deadlines result in penalties, the transaction value function is no longer as shown in Figure 1*a* where $V_T = 0$ for $t > D_T$. Instead, the value function now has $V_T = -P_T$ for $t > D_T$, where P_T is the penalty associated with missing the deadline.)

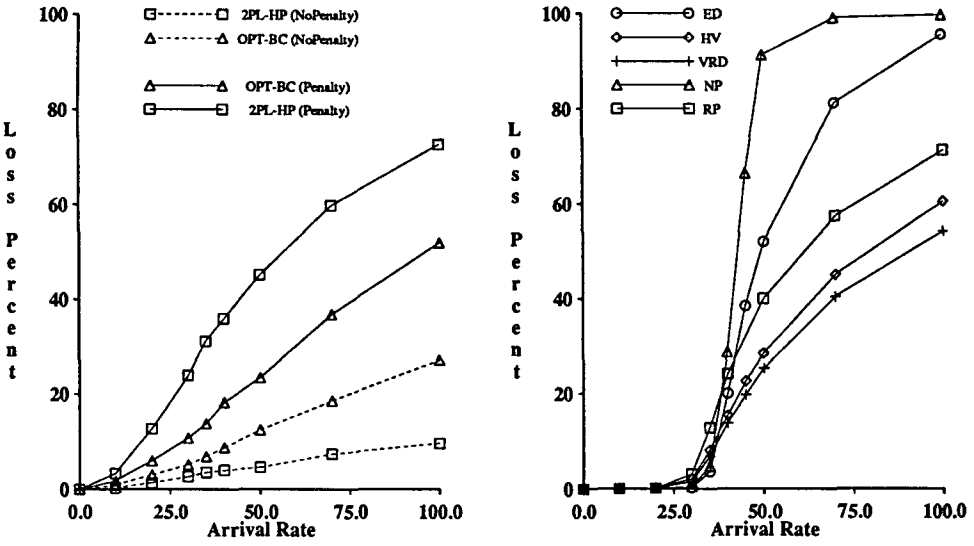
If missing a deadline has an associated penalty, then that penalty must be discounted from the value realized by the system in computing the net realized value. The penalty notion therefore provides a quantitative way to combine the separate metrics of *LossPercent* and *MissPercent*, because the total penalty is a function of the missed deadlines. The *LossPercent* 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.

We conducted preliminary experiments where missing a deadline has an associated penalty in order to investigate the effect on the various mappings. Here we consider only the case where all transactions have the same penalty. 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 is a function of how *many*, and not *which* transactions miss their deadlines. As a result of the priority mappings remaining the same, the performance numbers are a linear

Figure 13. Late penalty (HV) **Figure 14. Deadline/execution time correlation**



combination of the loss-percent and miss-percent statistics of the corresponding no-penalty experiments. This means that the performance numbers can be directly computed from the statistics of the corresponding no-penalty experiments.

For experiment 6.4.2, we computed the *LossPercent* results for the HV priority mapping using 2PL-HP and OPT-BC as the concurrency control algorithms with a penalty of 100.0 for each missed deadline (Figure 13, solid lines.) (The results obtained in the absence of penalties are also shown, dotted lines). With penalty, the performance of the 2PL-HP algorithm is considerably worse than that of OPT-BC because it misses many more deadlines. Therefore, when penalties are levied, the total penalty loss for 2PL-HP is much higher than for OPT-BC. The degree of change in *LossPercent* results, relative to the no-penalty case, is a function of the magnitude of the penalty 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. We have considered only the case where all transactions have the same penalty. Generally, each transaction may have a different penalty. A complete analysis of the impact of penalty on real-time priority mappings is a challenging problem that needs to be addressed, but is outside the scope of this article.

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 linear correlation between deadline

and execution time. The workload parameters *LSF* and *HSF* were both set to 4.0 (to match the mean slack ratio of the baseline model). All remaining parameter settings were kept the same as those of the baseline resource contention model (Section 6.3.1). *LossPercent* behavior is shown in Figure 14. There are important differences between the mapping behaviors shown in this figure and those shown in Figures 4*a* and 4*b*. The performance of VRD mapping is better than HV mapping throughout the entire loading range. With deadline formula DF1, VRD performed better than HV at low loads, but worse at high loads. Here, with DF2, the VRD mapping generates a priority ordering that is identical to the ordering generated by the Value Density (VD) mapping (Jensen et al., 1985). In VD mapping, task priorities are determined by 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 VD mapping is known to produce a total value at every point in time that is at least as high as any other schedule (Jensen et al., 1985), because the mapping gives higher priority to those tasks that can yield more value in a shorter time period. The VRD mapping is identical to VD because relative deadline and execution time are linearly correlated in DF2. Therefore, because the VRD mapping concentrates on "quick-paying" transactions, its performance is even better than that of the HV mapping (which concentrates only on "high-paying" transactions).

At intermediate and high loads, the NP mapping performs much worse here than all other mappings. When DF1 was used, the performance of NP was close to that of ED in these loading ranges. Here, with DF2, all transactions in the system make progress at the same rate. All transactions have the same slack ratio and, therefore, the same chance of making their deadline; this results in more missed deadlines. (With DF1 short transactions tend to have greater slack ratios than long transactions. Transactions with high slack ratios tend to complete before their deadlines. Transactions with low slack ratios are discarded earlier. This skew in slack ratios has a beneficial effect on the *MissPercent* characteristic.)

This experiment shows that correlation in workload characteristics can have an appreciable effect on mapping performance, and is therefore an area that should be investigated in greater detail.

7. Conclusions

Our experiments showed that for workloads with a limited, uniform spread in transaction values, the ED mapping provided the best performance among 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 the HV mapping delivered the best performance in spite of being deadline-indifferent. A large fraction of the deadlines were missed at high loads under all mappings, and the fact that HV prioritizes transactions by value alone ensured that high value transactions rarely missed their deadlines.

The VD mapping, which weights both values and deadlines equally, was found to behave identically to HV. The VRD mapping, which weights *relative* deadlines and values equally, provided the best overall performance among fixed-tradeoff mappings.

For workloads with 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 its ability to complete virtually all high-value transactions more than compensated. When transaction deadlines were linearly correlated with their execution times, the VRD mapping performs especially well by giving priority to those transactions that can return the most value in the shortest period of time.

In addition to results regarding the performance of fixed-tradeoff mappings, our experiments also showed susceptibility to performance breakdown based on workload characteristics. For example, assigning the same priority to a number of high-value transactions was 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.) It was also shown that associating a penalty with transactions whose deadlines are not met can seriously degrade the performance of some mappings.

Experiments were also conducted to explore the impact of data contention on the performance of various priority mappings. These experiments were conducted with several concurrency control algorithms in order to evaluate their performance and study their impact on 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 Haritsa et al., (1992) we showed that, with increasing data contention, real-time optimistic concurrency control algorithms outperform real-time locking algorithms in a firm deadline environment. That work employed an ED 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 the priority mappings we have considered.

Our experiments show that no single fixed tradeoff between value and deadline is appropriate under all circumstances. Rather, the right tradeoff is a function of the workload and system-operating conditions. This result highlighted the need for a priority assignment algorithm that could adaptively vary the value-deadline tradeoff to match the operating environment. To address this need, a bucket algorithm that allows the transaction value/deadline tradeoff to be varied was introduced in this article. In the bucket algorithm a structured approach is used to combine value and deadline based on basic real-time scheduling principles. The actual tradeoff made 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 interesting question is how to *adaptively* change the setting of this parameter to optimize performance as the system load varies. A mechanism to accomplish this goal was recently developed and reported in Haritsa et al. (1991); the mechanism requires further work to make it a fully

functional algorithm. We are currently researching this issue.

Another issue that we plan to explore further is the performance impact of different types of workload correlations, because we expect the bucket algorithm (unlike the fixed-tradeoff mappings) to be relatively immune to correlation-related performance degradation. Finally, the effect of transactions having different penalties on the performance of real-time priority mappings is an interesting open problem.

Acknowledgments

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

References

Abbott, R. and Garcia-Molina, H. Scheduling real-time transactions: A performance evaluation. *Proceedings of the Fourteenth International Conference on Very Large Database Systems,* Los Angeles, 1988.

Abbott, R. and Garcia-Molina, H. Scheduling real-time transactions with disk-resident data. *Proceedings of the Fifteenth International Conference on Very Large Database Systems,* Amsterdam, 1989.

Abbott, R. and Garcia-Molina, H. Scheduling I/O requests with deadlines: A performance evaluation. *Proceedings of the Eleventh IEEE Real-Time Systems Symposium,* Orlando, FL, 1990.

Agrawal, R., Carey, M., and Livny, M. Concurrency control performance modeling: Alternatives and implications. *ACM Transactions on Database Systems,* 12(4):609-654, 1987.

Baruah, S. and Rosier, L. Limitations concerning on-line scheduling algorithms for overloaded real-time systems. *Proceedings of the Eighth IEEE Workshop on Real-Time Operating Systems and Software,* Atlanta, GA, 1991.

Biyabani, S., Stankovic, J., and Ramamritham, K. The integration of deadline and criticalness in hard real-time scheduling. *Proceedings of the Ninth Real-Time Systems Symposium,* Huntsville, AL, 1988.

Buchmann, A., McCarthy, D., Hsu, M., and Dayal, U. Time-critical database scheduling: A framework for integrating real-time scheduling and concurrency control. *Proceedings of the Fifth International Conference on Data Engineering,* Los Angeles, 1989.

Dertouzos, M. Control robotics: The procedural control of physical processes. *Proceedings of the IFIP Congress,* 1974.

Eswaran, K., Gray, J., Lorie, R., and Traiger, I. The notions of consistency and predicate locks in a database system. *Communications of the ACM,* 19(11):624-633, 1976.

Franaszek, P. and Robinson, J. Limitations of concurrency in transaction processing. *ACM Transactions on Database Systems,* 10(1):1-28, 1985.

Gray, J. Notes on database operating systems. In: Bayer, R., Graham, R., and Seegmuller, G., eds., *Operating Systems: An Advanced Course,* Springer-Verlag, 1979.

Haritsa, J., Carey, M., and Livny, M. On being optimistic about real-time constraints. *Proceedings of the Ninth ACM SIGACT-SIGART-SIGMOD Symposium on Principles of Database Systems,* Nashville, TN, 1990*a*.

Haritsa, J., Carey, M., and Livny, M. Dynamic real-time optimistic concurrency control. *Proceedings of the Eleventh IEEE Real-Time Systems Symposium,* Orlando, FL, 1990*b*.

Haritsa, J., Livny, M., and Carey, M. Earliest-deadline scheduling for real-time database systems. *Proceedings of the Twelfth IEEE Real-Time Systems Symposium,* San Antonio, TX, 1991.

Haritsa, J., Carey, M., and Livny, M. Data access scheduling in firm real-time database systems. *Journal of Real-Time Systems,* 4:203-241, 1992.

Huang, J., Stankovic, J., Towsley, D., and Ramamritham, K. Experimental evaluation of real-time transaction processing. *Proceedings of the Tenth IEEE Real-Time System Symposium,* Santa Monica, CA, 1989.

Huang, J. and Stankovic, J. Buffer management in real-time databases. *COINS Technical Report 90-65,* University of Massachusetts, Amherst, MA, 1990*a*.

Huang, J. and Stankovic, J. Concurrency control in real-time database systems: Optimistic scheme vs. two-phase locking. *COINS Technical Report 90-66,* University of Massachusetts, Amherst, MA, 1990*b*.

Jensen, E. Locke, C., and Tokuda, H. A time-driven scheduling model for real-time operating systems. *Proceedings of the Sixth IEEE Real-Time Systems Symposium,* 1985.

Koren, G. and Shasha, D. D^{over}: An optimal on-line scheduling algorithm for overloaded real-time systems. *Technical Report CS TR 594,* Courant Institute, New York University, New York, NY, 1992.

Kung, H. and Robinson, J. On optimistic methods for concurrency control. *ACM Transactions on Database Systems,* 6(2):213-226, 1981.

Livny, M. *DeNet User's Guide,* Version 1.0, Computer Science Department, University of Wisconsin, Madison, WI, 1988.

Locke, C. Best effort decision-making for real-time scheduling. *Ph.D. Thesis,* Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1986.

Menasce, D. and Nakanishi, T. Optimistic vs. pessimistic concurrency control mechanisms in database management systems. *Information Systems,* 7(1):13-27, 1982.

Reed, D. Naming and synchronization in a decentralized computer system. *Ph.D. Thesis,* Department of Computer Science, Massachusetts Institute of Technology, Boston, MA, 1978.

Robinson, J. Design of concurrency controls for transaction processing systems, *Ph.D. Thesis,* Carnegie-Mellon University, Pittsburgh, PA, 1982.

Sha, L., Rajkumar, R., and Lehoczky, J. Priority inheritance protocols: An approach to real-time synchronization. *Technical Report CMU-CS-87-181,* Departments of Computer Science, Electrical and Computer Engineering, and Statistics, Carnegie-Mellon University, Pittsburgh, PA, 1987.

Stankovic, J. and Zhao, W. On real-time transactions. *ACM SIGMOD Record,* 17(1):4-18, 1988.