Using Dynamic Priority in Real-Time Database Systems

Prasad Wagle and Sang H. Son

Computer Science Report No. TR-90-14
July 5, 1990

Using Dynamic Priority in Real-Time Database Systems

Prasad Wagle
Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, VA 22903
USA

Abstract

A real-time database system has timing constraints associated with transactions and the database. To ensure that such a system completes as many transactions as possible without violating their timing constraints, its scheduling strategy should be dynamic and use information about the timing constraints associated with transactions and the database. Ideally, to enhance the predictability of the system; such a scheduling strategy should be used in all situations where there is resource contention. This paper describes an intelligent dynamic scheduling strategy for scheduling transactions in real-time database systems. The scheduling strategy uses timing information about transactions and the database to enhance the system's ability to meet transaction deadlines. The performance of the scheduling strategy is tested by using it in a simulated pulse detection system.

Key words: real-time database, concurrency control, time-critical scheduling, priority, locking

---

This work was supported in part by the Office of Naval Research under contract # N00014-88-K-0245, By IBM Federal Systems Division, and by the Center for Innovative Technology under contract # CIT-INF-90-011.

1. Introduction

1.1. What are Real-time Database Systems?

Real-time database systems are database systems that support real-time computing. *Real-time computing* is that type of computing where the correctness of the system's response depends not only on the logical result of the computation, but also on the time at which the results are produced [Stan88A]. The timing constraint on the system's response is called *deadline*. Traditional real-time systems have concentrated on systems which have hard deadlines. If a system misses a hard deadline, the consequences can be disastrous. On the other hand, if the system misses a soft deadline, there may still be some value for computing the response of the system. Real-time systems are assuming an increasingly important role in our society. Examples of current real-time computing systems are command and control systems, aircraft avionics, robotics, network management, and program trading.

Most of the complex real-time computing applications need to access large amount of data. Thus, we need database systems which are cognizant of the requirements of real-time computing, i.e. *real-time database systems*. Transactions in a real-time database systems are required to do operations on the database, like read, write, insert and delete, subject to timing constraints. An example of a real-time database system is a pulse detection system. A pulse detection system is used to track objects using radars. The information about objects in reality is maintained in a database of emitter files. Typically, a pulse detection system consists of simultaneously active transactions, with different timing constraints and resource requirements, which read and update the database of emitter files.

1.2. Comparison with Conventional Real-time Systems and Database Systems

A real-time database system has similarities as well as differences with conventional real-time systems and database systems.

The following are the similarities between real-time database systems and conventional database systems. First, both systems process transactions which access data items according to the consistency constraints of the database. Second, both systems have transactions with complex and unpredictable data requirements.

The following are the differences between real-time database systems and conventional database systems. First, transactions in conventional database systems have no timing constraints. The goal of conventional database systems is to reduce the average response time of the transactions being processed rather than trying to satisfy the timing constraint of individual transactions. Second, the consistency constraints that exist in conventional database systems are strict serializability constraints which are not always needed in real-time database systems.

The similarity between real-time database systems and conventional real-time systems is that both systems process entities (tasks and transactions) which have timing constraints.

The following are the differences between real-time database systems and conventional real-time systems. First, tasks in conventional real-time systems have hard deadlines whereas transactions in real-time database systems can have soft deadlines. Second, data in conventional real-time systems normally do not have consistency constraints. Third, tasks in a conventional real-time system have simple and predictable data or resource requirements.

1.3. Validity Constraints

Deadlines are timing constraints associated with transactions. There exist another kind of timing constraints which are associated with transactions and data objects in the database. In a database, there may be some data objects which get old or out-of-date if they are not updated within a certain period of time. To quantify this notion of age we associate with each data object a degree of validity which decreases with time. The validity curve associated with each data object is a plot of the degree of validity of the data object with respect to the time elapsed after the object was last modified. Fig. 1 shows an example validity curve for data objects.

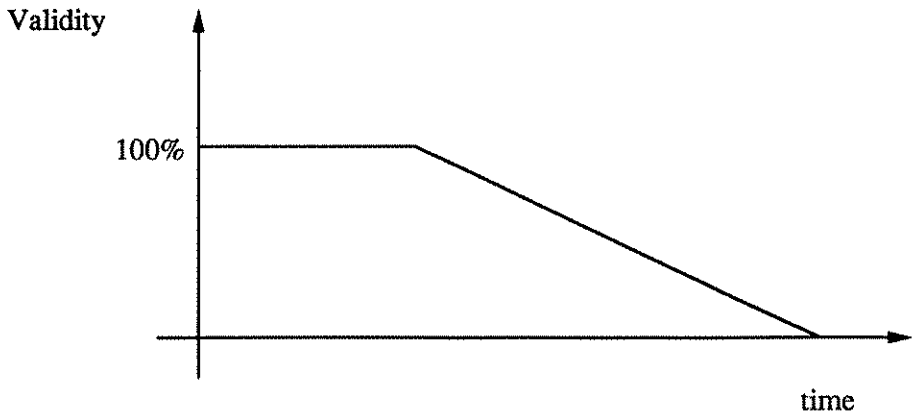

Fig. 1. Validity curve

If w is the time of last modification of a data object and t is the current time, we can calculate the validity of the data object at time t from its validity curve. Now, a transaction may require all the data objects it reads to have a minimum degree of validity. This constraint could be either hard or soft, like deadlines. Scheduling decisions could be made more intelligent by incorporating this validity information about transactions and data objects they read.

1.4. Scheduling Problem in Real-Time Database Systems

Scheduling theory is used in widely different areas like general computer systems, operations research, real-time systems, database systems, and finally, real-time database systems. The common aspects about scheduling in all the above disciplines are:

- there is a scarce resource
- there is more than one entity wishing to use the resource.

- the scheduling decision is choosing the entity to which the resource should be granted next.

This is the most general and abstract description of the scheduling problem. The scheduling problem is made more specific to the application depending on the characteristics of the resource, characteristics of the entities using the resource and the way the scheduling decision is made. The resource could be preemptible or non-preemptible. The entity to be scheduled could be a task or a transaction. The scheduling decision could be made with the aim of optimizing some performance metric with regard to certain resource and/or timing constraints.

In *operations research* scheduling problems, there is a fixed system having completely specified and static service characteristics. The goal is to find optimal static schedules which minimize the response time for a given task set [Stan88A].

In *database systems*, a scheduler accepts database operations from transactions and schedules them appropriately for the data manager [Bern87]. In conventional database systems, the scheduler is entrusted with the task of enforcing serializability constraints of the database. In this case, the resource is a data item, the entity to be scheduled is a database operation and the scheduling decision is made according to the consistency constraints of the database.

In normal *operating systems*, there are tasks waiting for resources like the CPU or the I/O processor. The scheduling decision can be made according to well known scheduling algorithms like priority based scheduling or round robin scheduling.

In *real-time systems*, there exist tasks contending for scarce resources. But, unlike the above disciplines, the entities to be scheduled (tasks) have timing constraints (deadlines). There is generally no incentive to minimize the response time other than meeting deadlines.

In *real-time database systems*, transactions which have timing constraints contend for scarce resources. But, the scheduling strategies devised for conventional database systems or real-time systems cannot be applied to real-time database systems because of the differences that exist between them (see section 1.2). In real-time database systems, it is necessary to take into

account the timing constraints associated with the transactions as well as the consistency constraints associated with the database while making scheduling decisions.

1.5. Static versus Dynamic Scheduling

It is possible to statically guarantee real-time constraints by pre-calculating all possible schedules of transactions off-line. There are two reasons why this approach is infeasible [Stan88B]. First, the task of finding all possible schedules of transactions is NP hard. Therefore, the task becomes computationally intractable when there are a large number of simultaneously active transactions. Second, the demands on a real-time database system can change frequently. For example, aperiodic transactions, by their very nature, can be activated at unpredictable times. Therefore, a dynamic scheduling strategy is needed to make the system more flexible. Also, to make "intelligent" scheduling decisions, the scheduling strategy should use as much timing information as possible about transactions and the data objects they access.

A scheduler in database systems accepts database operations from transactions and schedules them appropriately for the data manager [Bern87]. In a distributed system, each site has its own scheduler which can receive database operations from transaction managers at different sites. In conventional database systems, the scheduler is entrusted with the task of enforcing the serializability constraints. In real- time database systems, it is also necessary to take into account the timing constraints associated with the transactions and the database while making scheduling decisions.

However, to guarantee real-time constraints, it may be insufficient to use the extra information about transactions only while scheduling database operations. This is because transactions interact with the operating system and the I/O subsystem in extremely unpredictable ways. For example, we have no control over the way the scheduling decisions are made for scarce resources at the operating system level. Therefore, to improve the predictability of real-time database systems, i.e., to enhance the guarantee of meeting real-time constraints, we should use the additional

information about transactions to make scheduling decisions at all places where more than one transactions try to use (or access) a scarce resource. This scarce resource could be the CPU, a data object, or the communications subsystem.

2. The Scheduling Algorithm

In this section, we describe a dynamic scheduling strategy for transactions in real-time database systems. The scheduling strategy uses timing and validity information about transactions and data objects to calculate dynamic priorities of transactions. These priorities are then used to make scheduling decisions at all places where transactions contend for scarce resources.

2.1. Information required for intelligent scheduling

This section discusses the nature of information about transactions required by the scheduling strategy and a way to represent it.

A transaction can be represented as a tuple (SP, RS, WS, A, D, E, MV). The elements of the tuple are described below.

(1) System priority (SP):

This is the static component of the dynamic priority associated with a transaction. It is a measure of the value to the system of completing the transaction within its timing constraints. For example, transactions dealing with emergency situations should have a higher priority than routine transactions.

(2) Read set (RS):

This is the set of data objects which the transaction reads.

(3) Write set (WS)

This is the set of data objects which the transaction writes.

(4) Arrival time (A):

This is the time at which the transaction arrives in the system.

(5) Deadline (D):

This is the time before which the transaction has to finish its execution. The transaction specifies whether the deadline is hard or soft.

(6) Runtime estimate (E):

This is the estimate of the processing time required by a transaction. This includes the time required for CPU as well as I/O operations.

(7) Minimum Validity(V_{min}):

This is the minimum degree of validity required of all objects read by the transaction. The transaction specifies whether this validity constraint is hard or soft.

The above information about the transaction is available to the system before the transaction is started and remains constant throughout the transaction execution. Since the scheduling strategy is dynamic, it needs information about the transaction which varies with time. The information which varies with time is described below.

(8) Read set validity(RSV):

This is the degree of validity of data objects in the transaction's read set. The degree of validity of a data object can be calculated from its validity curve. The validity curve of a data object defines a function of the degree of validity of the data object with respect to the time elapsed after the data object was last modified. Therefore, if we know the time the object was last modified, we can calculate the degree of validity of the data object at the current time from the validity curve.

(9) Processing time(P):

This is the processing time already received by a transaction. This includes the time required for CPU as well as I/O operations.

(10) Current time(C):

This is the time at which the scheduling decision is made.

2.2. Scheduling design issues

Before implementing any scheduling strategy, it is important to consider the overhead it requires. Obviously, a complicated scheduling strategy requires more time. This factor can be crucial in deciding whether it is of any practical benefit to use the extra information about transactions and the database in the scheduling strategy.

For instance, if the database is disk-resident and the transactions are I/O intensive, the time required for I/O operations would be large compared to the time required for doing CPU operations. In that case, it would not make a big difference whether or not we use a complicated scheduling policy at the CPU level. The bottleneck in this case would be the data objects and it would be imperative to schedule the database operations in an intelligent way. But if the database is memory resident and the transactions are CPU intensive then it would become necessary to use the extra information about transactions in the scheduling decision at the CPU level. Given below is a scenario which illustrates a situation where an intelligent scheduling strategy at the CPU level would be helpful.

Assume that transactions execute CPU and I/O instructions alternately. Let the time required for one session of CPU computation be 10 time units and the time required for one I/O operation be 2 time units (if there is no blocking). Let the transactions to be scheduled (T_1 and T_2) have the characteristics given below. This situation can arise if both T_1 and T_2 wait for some other transaction to release a data object. The transaction releases the data object at time 5. Thus, the scheduling decision has to be made at time 5.

Transaction	Arrival time	Estimate	Deadline (Hard)	Operations
T_1	0	12	30	read(1)
T_2	5	12	20	read(1)

According to an elementary FCFS scheduling strategy, T_1 is scheduled first and it completes at time 12. T_2 starts at time 10, but since it requires 12 time units to complete, it misses its deadline at time 20. (As shown in Fig. 2.1)

If the system is intelligent enough to follow the elaborate scheduling strategy to be discussed in Section 4, T_2 would be scheduled first. (According to the least slack method of assigning priorities, T_2 has a higher priority than T_1, because the slack of T_2 is less than the slack of T_1.) In that case both transactions would meet their deadlines as shown in Fig. 2.2.

An issue involved in designing a scheduling strategy is whether or not to allow preemption. The scheduling decision at the CPU level normally allows preemption. However, if we allow preemption at the data object level, we may have to abort the preempted transaction for maintaining consistency of the database. The general problem descriptions for the two cases without having a particular resource type in mind, are as the following:

Case 1. No preemption.

There are more than one transactions requesting a resource and we have to decide the tran-

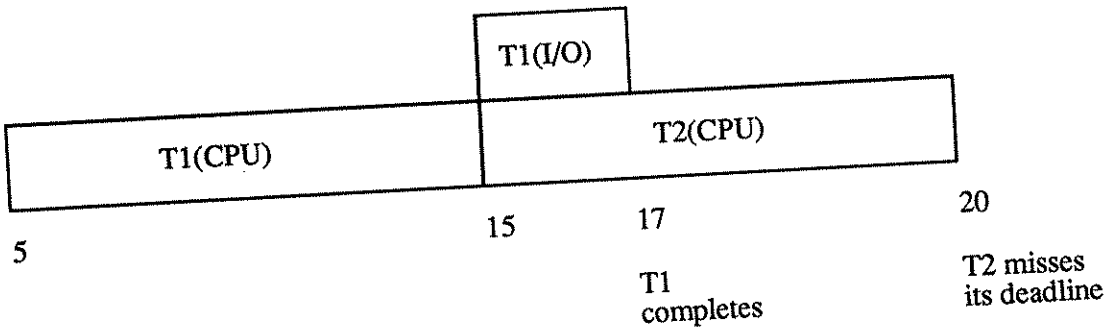

Fig. 2.1 FCFS Scheduling

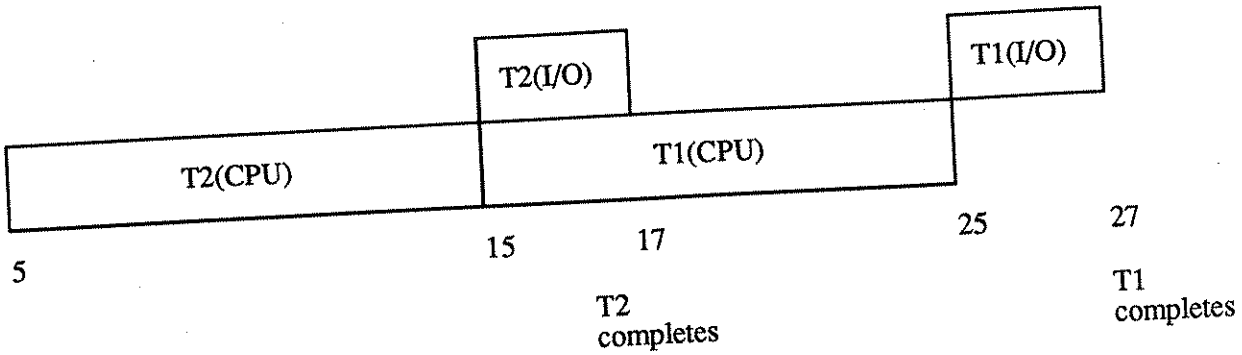

Fig. 2.2 Intelligent Scheduling

saction which should be granted the resource. Once a transaction gets the resource it runs till it finishes using the resource.

Case 2. Allow preemption.

There is a transaction currently holding a resource and there is a transaction requesting the same resource. We have to decide whether or not to preempt the transaction holding the resource and grant the resource to the transaction requesting it.

When preemption is not allowed, the scheduling decision has to be made whenever a transaction relinquishes a resource or when a transaction requests a resource which is not being used. When preemption is allowed the scheduling decision has to be made whenever a transaction either requests or relinquishes a resource.

2.3. Real-time Database Scheduler

The scheduling strategy for transactions in real-time database systems can be decomposed into three sub-parts [Abbo88], [Abbo89]:

(1) Determining eligibility

(2) Assigning dynamic priorities

(3) Making the final scheduling decision of granting the resource.

In this section we discuss each of these sub-parts in detail.

2.3.1. Determining eligibility

Before making a scheduling decision we have to decide whether the transactions involved are eligible for scheduling i.e. whether it is of any use to the system to start processing those transactions. If a transaction is ineligible for scheduling we abort it immediately.

We assume that, if a transaction misses a hard deadline, it is ineligible for scheduling and should be aborted. If a transaction misses a soft deadline, it is still eligible for scheduling. We also check whether it is possible for the transaction to finish before its deadline:

(deadline - current time) ≥ (Estimate - Processing time received)
i.e. (D - T) ≥ (E - P)

If it is not possible, and the deadline in question is hard, we consider the transaction ineligible for scheduling. However, if the deadline is soft, the transaction remains eligible for scheduling.

The steps taken in incorporating validity constraints are similar to those taken for deadlines. If a transaction misses a hard validity constraint then it is ineligible for scheduling and should be aborted. If the validity constraint missed is soft, then we continue executing the transaction at a different priority. We also check, for each data item read by the transaction, whether its degree of validity is greater than the minimum validity level expected by the transaction:

For all data objects d read by the transaction, $V_d(T) > V_{min}$
where, $V_d(T)$ is the degree of validity of object d at time T

If that is not the case, and the validity constraint of the transaction is hard, we consider the transaction ineligible for scheduling. However, if the validity constraint is soft, the transaction

remains eligible for scheduling.

2.3.2. Assigning dynamic priorities

The dynamic priority of a transaction is a number calculated by the scheduler while making the scheduling decision. It is a measure of the importance, to the over-all goals of the system, of scheduling that transaction before others at that point in time [Stra89]. Since this measure may change with time, it has to be calculated dynamically every time two transactions are compared during the scheduling decision making process.

Dynamic priority (DP) is a weighted sum of the following factors:

(1) System priority (SP): It is the static component of dynamic priority.

(2) Slack with respect to deadline (SDL): It is the amount of time the transaction can be delayed and still meet its deadline. It is calculated as follows:

Slack = Deadline - Current time - (Estimate - Processing time)
SDL = D - T - (E - P)

(3) Slack with respect to minimum validity constraints (SV): It is the amount of time the transaction can be delayed and still be completed without violating its validity constraints.

SV = Min { t | For all data objects d read by the transaction, $V_d(T + t) > V_{min}$ }
where, $V_d(T + t)$ is the degree of validity of object d at time (T + t),
assuming no updates between time T and (T + t).

Dynamic Priority (DP) is calculated as follows:

DP := $DP_1 + DP_2 + DP_3$
where,
$DP_1 := W_1 * SP$
$DP_2 := W_2 * SDL$
$DP_3 := W_3 * SV$

The factors involved in determining the dynamic priority of a transaction have constraints closely related to the characteristics of real- time transactions. First, $W_1 > 0$, since if SP increases, DP should increase. Also, if SDL > 0 then $W_2 < 0$, since if SDL decreases then DP should increase. If SDL < 0, then the transaction has already missed its deadline. Note that since the transaction is still eligible for scheduling, the deadline missed must have been soft. At this point, there are two options available to us. We could reason as follows: Since the transaction has missed its deadline (soft), it should be finished as soon as possible, and hence its priority must be increased. In that case, $W_2 < 0$. However, we might reason that since the transaction has already missed its deadline, its priority should be reduced so that it does not interfere with other transactions in the system which are nearing their deadlines. In that case, $W_2 > 0$. Similar discussion applies to W_3 and SV.

The relative values of W_1, W_2, W_3 depend on the high level goals of the system. For example, some systems may aim at minimizing the number of transactions that miss their deadline, in which case W_1 would not be very high. Some systems might require that absolutely none of the higher priority transactions be aborted, in which case W_1 would very high.

Given below is a scenario which illustrates that a scheduling strategy at the CPU level taking validity constraints into account does prevent unnecessary aborts of transactions. Assume that transactions use the CPU and do I/O operations alternately. Let the time required for one session of CPU computation be 10 time units and the time required for one I/O operation be 2 time units (if there is no blocking). Let the transactions to be scheduled (T_1 and T_2) have the characteristics given below.

Transaction	Arrival time	Estimate	Deadline (Hard)	Minimum Validity (Hard)	Operations
T_1	0	12	30	100%	read(1)
T_2	0	12	25	50%	read(1)

Let the validity curve for object 1 be as shown in Fig. 2.3, and the time it was last modified be 0. Let the weights W_2 and W_3 for calculating dynamic priorities be -1. This implies that, in the formula for calculating dynamic priorities, the slacks with respect to deadline and validity constraints have the same weight.

If validity constraints are not considered:

In this case, DP := $DP_1 + DP_2$. The slack of T_1 with respect to deadline is 18. The slack of T_2 with respect to deadline is 13. Therefore,

$DP_2(T_1) = -18$ and $DP_2(T_2) = -13$.
i.e. $DP_2(T_2) > DP_2(T_1)$.

Assuming equal system priorities, DP(T_2) > DP(T_1), implying that T_2 would be scheduled first The execution would proceed as shown in Fig. 2.4. T_2 would finish its execution at time 12. Then T_1 would start. But, at time 20 the validity of object 1 would be 50%. This would violate the validity constraint of T_1, which would have to be aborted.

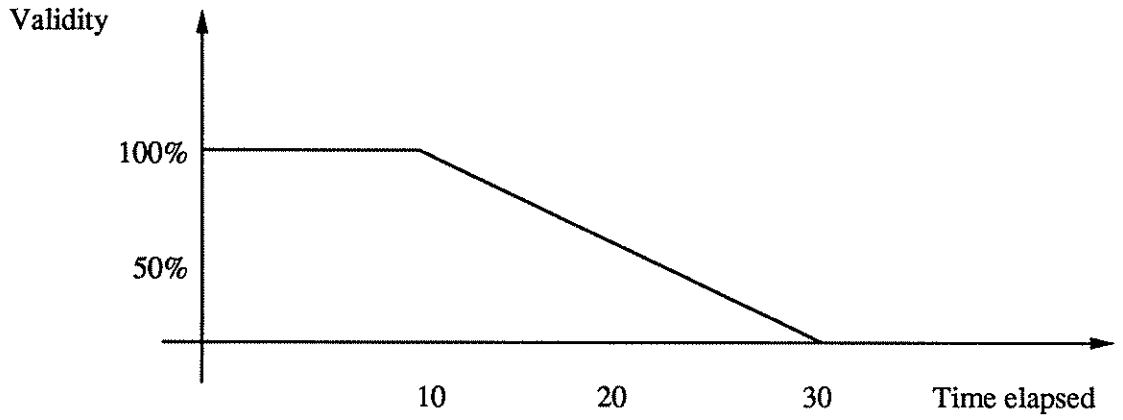

Fig. 2.3 Validity Curve

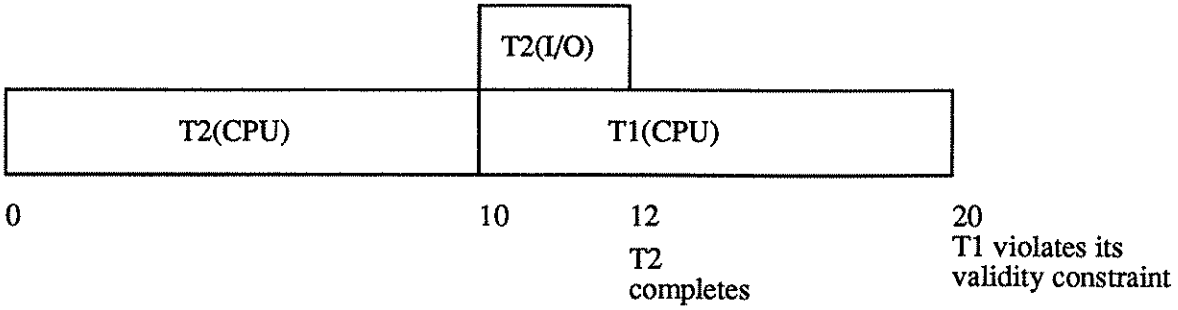

Fig. 2.4 Validity constraints ignored

If validity constraints are considered:

In this case, DP := $DP_1 + DP_2 + DP_3$. The slack of T_1 with respect to validity constraints is 10. The slack of T_2 with respect to validity constraints is 20. Therefore,

$DP_3(T_1) = -10$ and $DP_3(T_2) = -20$.
i.e. $DP_2(T_1) + DP_3(T_1) = -28$ and $DP_2(T_2) + DP_3(T_2) = -33$
i.e. $DP_2(T_1) + DP_3(T_1) > DP_2(T_2) + DP_3(T_2)$.

Assuming equal system priorities, DP(T_2) > DP(T_1), implying that T_1 would be scheduled first. The execution would proceed as shown in Fig. 2.5. At time 10 the validity of object 1 would be 100%, satisfying T_1's validity constraints. Thus T_1 would finish its execution at time 12. Then T_2 would start. At time 20, the validity of object 1 would be 50%, satisfying T_2's validity constraints. Thus T_2 would finish its execution at time 22.

Thus, incorporating validity constraints in the scheduling strategy does prevent transactions from being aborted unnecessarily.

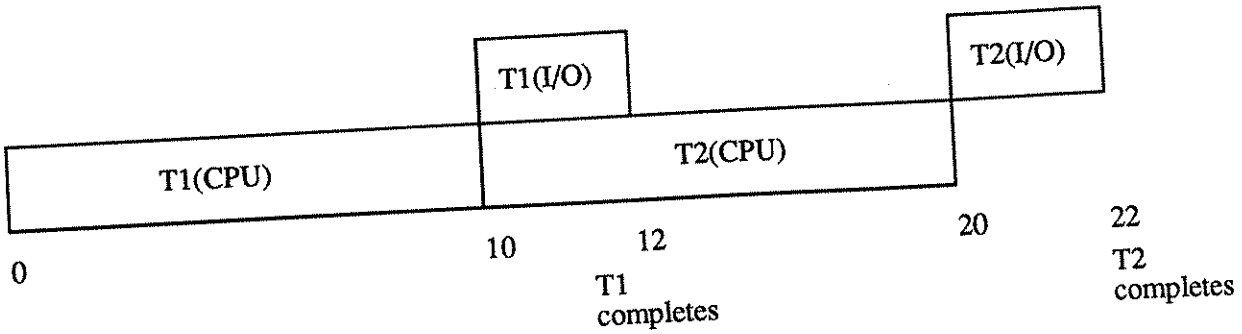

Fig. 2.5 Validity constraints considered

2.3.3. Making the final scheduling decision

The way the final scheduling decision is made depends on whether preemption is allowed or not. In the following discussion we assume that the transactions considered have already passed the eligibility test. Let us consider the scheduling algorithms for the two cases:

Case 1. No preemption.

There are more than one transactions requesting a resource and we have to decide the transaction which should be granted the resource. In this case we grant the resource to the transaction with the highest dynamic priority.

Case 2. Allow preemption.

There is a transaction currently holding a resource and there is a transaction requesting the same resource. We have to decide whether to preempt the transaction holding the resource and grant the resource to the transaction requesting it.

Let T_h and T_r be the two transactions requesting the resource. Let P(T_h) and P(T_r) be dynamic priorities of the two transactions. Let P(T_h if preempted) be the priority of T_h were it to be preempted by T_r. The algorithm is as follows:

IF P(T_r) > MAX(P(T_h) , P(T_h if preempted)) THEN
    IF RemainingTime(T_h) > Slack(T_r) THEN
        Preempt T_h;
    END;
END;
where RemainingTime(T_h) = Runtime estimate
                - Processing time received by T_h.

2.3.4. Handling Periodic Transactions

There are many applications in real-time database systems which have periodic transactions. For example, a pulse detection system used in radar tracking needs to periodically read pulse data from antennas, process them, and then display them on an operator console [Hale89]. Periodic transactions are restarted after an interval of time equal to their period. If an execution of a periodic transaction does not complete before the end of its period, it is aborted and a new instance of the same transaction is restarted. From the scheduler's viewpoint, periodic transactions can be modelled as transactions having hard deadlines equal to their periods.

If a data object is updated by a periodic transaction with period (T), its validity curve can be similar to the one shown in Fig. 2.6. The form of the validity curve implies that the validity of the data object remains 100% during an interval T after the object has been updated. Henceforth, it reduces by a fixed amount v every T time units. This makes the task of calculating the degree of validity of a data object easy. If t is the time elapsed since the data object was last modified,

Degree of validity = 100 - (t / T) * v
where, "/" signifies integer division.

This behavior of the degree of validity of a data object is similar to the concept of normalized age of data objects [Song89]. For periodic transactions, the basic scheduling strategy for determining eligibility, assigning priorities and making the final decision remains the same as for aperiodic transactions.

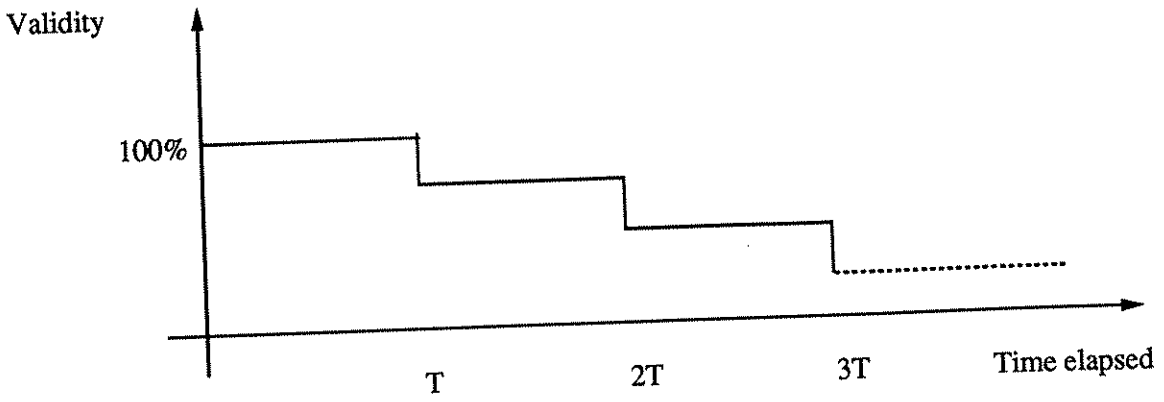

Fig. 2.6 Example Validity curve

3. Simulation Study

3.1. Need for a Real-Life Application

The research on real-time transactions scheduling is still in its infancy. There exists no formal theoretical framework to analyze the performance of the existing scheduling algorithms. For this reason, experimentation is a necessity to compare the performance of different scheduling algorithms.

Until now, none of the algorithms proposed in previous studies have been evaluated in real systems. [Abbo88] and [Abbo89] present experimental results based on simulation, whereas [Huan89] presents an integrated approach to study real-time transaction processing on a testbed system. In these studies semantically meaningless transactions are randomly generated with random system priorities, resource requirements, and timing constraints. The disadvantage of this approach is that it does not give the researcher a true feel for real-life problems. Also, for any scheduling strategy to be used in industry, it has to be supported by an extensive round of

experimentation with a *real-life* application.

We feel that in the area of real-time systems, there is a pressing need for a canonical problem which can be used to test different strategies for solving problems like scheduling or fault tolerance. An analogy can be drawn to the dining philosophers problem in the area of interprocess communication. For these reasons, we chose to simulate a pulse detection system, a real-life, real-time database system application, to evaluate the proposed scheduling algorithm.

3.2. What is a Pulse Detection System?

A pulse detection system is an example of a real-time database system [Hale89]. It is used to detect and track external objects by means of pulses (radar or sonar) received from them. The pulse detection system maintains information about each object in reality in a database of emitter files. It contains a number of simultaneously active transactions with different system priorities, timing constraints, and resource requirements.

Examples of periodic transactions are:

(1) Transactions which collect pulse data from the radar.

(2) Transactions which evaluate the pulse data received and perform the necessary operations in the database of emitter files.

(3) Transactions which remove emitter files which have not been updated for a certain interval of time.

(4) Transactions which monitor the operator's console for operator commands.

Examples of aperiodic transactions are:

(1) Transactions which shoot missiles at the enemy objects.

(2) Transactions which display information about the enemy objects.

Now, it is possible for some of these transactions to require the same resource at the same time. This is when the question of intelligent scheduling of transactions becomes extremely

important.

The simulation system we have implemented runs on a SUN Workstation (preferrably SUN 3/75 with a color monitor). It is based on the scenario of a battleship surrounded by airborne enemy objects like aircrafts or missiles. It consists of two windows: the *reality* window, and the *operator's console* window (see Fig. 3.1).

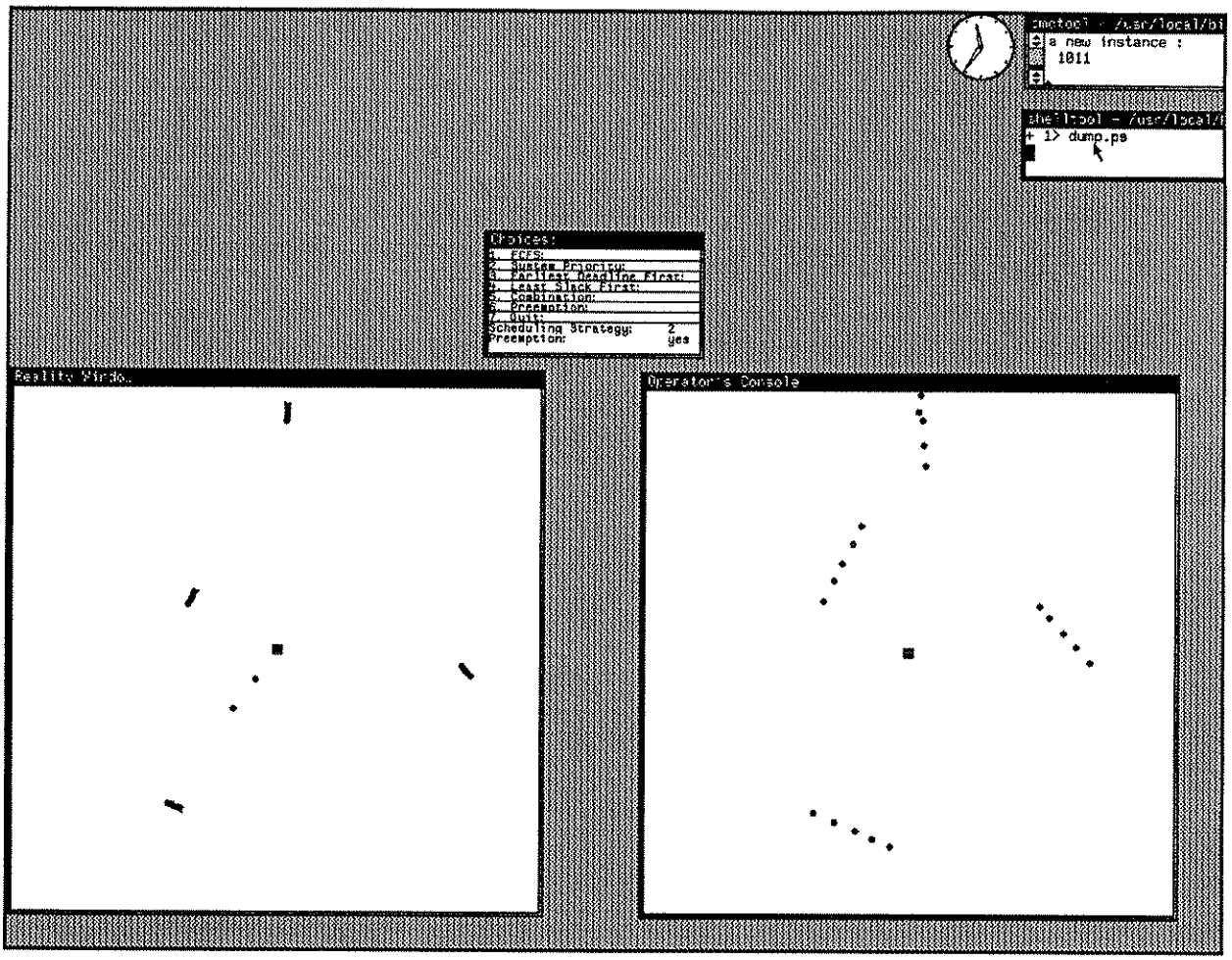

Fig. 3.1. Simulation Screen

The reality window consists of a stationary battleship at its center and the surrounding enemy objects. Each object has a position and velocity associated with it. An object is implemented as a process which calculates the new position of the object and displays it in the reality window. The reality window is managed by two modules: `Object` and `Reality`. The module `Object` is responsible for creating objects in reality, continuously updating their positions and detecting collisions. The module `Reality` is responsible for creating the reality window. It has a procedure called `GetPulseData` which simulates the operation of a radar by getting new pulse data of an object in reality.

The operator's console window displays the operator's view of reality as maintained by the pulse detection system. It is supposed to display the most current positions of enemy objects in reality. The operator's console window is managed by the modules: `Detect` and `EmitterFile`. The module `EmitterFile` maintains an emitter file to store information corresponding to each enemy object in reality.

The `Detect` module contains three periodic and two aperiodic transactions. Each transaction is implemented as a process. The following are the periodic transactions with a brief description of what they do.

(1) `Track`: It calls `Reality:GetPulseData` to get a new pulse data of an object in reality. It scans all the emitter files to find an emitter file which correlates with the pulse data received. If it finds such an emitter file, it updates it; else it creates a new emitter file with that pulse data.

(2) `Clean`: It periodically scans the emitter files and deletes emitter files which haven't been updated for a predetermined amount of time assuming that the object which they represents have been destroyed.

(3) `Operator Interaction`: This transaction accepts operator commands. For example, an operator may query the database to find more information about an emitter file, or he

may start a transaction to shoot an enemy object.

The operator interaction transaction, in turn, can start two aperiodic transactions, which are:

(1) `Display Information`: This transaction displays information about the object chosen by the operator.

(2) `Shoot Object`: This transaction shoots a missile at the object chosen by the operator.

3.3. The `Simulation` Module

Since this is a simulation of the original system, it is very important that the experimenter has control over the relative speeds of the transactions being executed and the amount of time a transaction needs to use a resource. This is done by using the `Simulation` module. The `Simulation` module is a general purpose module which contains the following procedures:

```
PROCEDURE Hold (delay : LONGINT);
(* This procedure is executed by processes whose execution
   is to be suspended by "delay" units of time. *)

PROCEDURE Open (VAR r : Resource; attr : ARRAY OF CHAR);
(* This procedure is used to create a resource with a name
   stored in attr *)

PROCEDURE Close (VAR r : Resource);
(* This procedure is used to delete a resource. A resource
   should not be deleted until its statistics are printed. *)

PROCEDURE HoldR (VAR r : Resource; delay : LONGINT);
(* This procedure is executed by processes which desire to
   use resource "r" for "delay" units of time. If more than
   one processes desire to use the same resource at the same
   time, their requests are serialized according to certain
   scheduling strategy.*)

PROCEDURE SetSchedStrat(schedStrat : CARDINAL);
(* This procedure can be called any time during the simulation
   to set the scheduling strategy to be followed. *)

PROCEDURE SetPreemption(preemption : BOOLEAN);
(* This procedure can be called any time during the simulation
   to decide whether preemption is allowed. *)
```

A process calls `Hold` to simulate the passage of time when it executes some actions. It calls `HoldR` when it uses some shared resource. If two or more processes want to use a resource at the same time, a decision has to be made in the `Simulation` as to which process should be granted the resource. This decision is made considering the attributes associated with the different processes according to some scheduling strategy.

Currently each process contending for a shared resource has the following attributes: (1) System priority; (2) Arrival time; (3) Deadline; (4) Run-time estimate; (5) Processing time it has received; and (6) Minimum validity of the data it reads.

The simulation system allows the researcher to choose the scheduling strategy followed, with or without preemption, and examine its effects on the pulse detection system. Currently the following strategies, with or without preemption, are supported: (1) First Come First Served; (2) System Priority; (3) Earliest Deadline First; (4) Least Slack First; and (5) A variant of the scheduling strategy presented in the previous chapter, which will be henceforth referred to as the *Combination* strategy. The *Combination* strategy uses the system priority (SP) and the slack with respect to deadline (SDL) while making its scheduling decisions.

Our intention is to show that the performance of the pulse detection system can be enhanced by the use of intelligent scheduling algorithms. The performance of a scheduling strategy can be judged by two ways: (1) by the visual behavior of the simulated pulse detection system; or (2) by the information about successful completion of transactions displayed each time the scheduling strategy is changed.

3.4. Simulation Assumptions

The following are the assumptions made about the simulations.

(1) The timing parameters of transactions like the run-time estimate or deadline are arbitrary and do not correspond to any realistic system. This is done because data about real systems is of a highly classified nature.

(2) The scheduling overhead is ignored. This assumption is supported by [Huan89].

(3) The consistency of the database is maintained using exclusive locks which are non-preemptible. A more efficient concurrency protocol would be the priority ceiling protocol using shared locks [Sha88].

(4) All transactions have hard timing and validity constraints. When a periodic transaction or an instance of a periodic transaction is started, the run-time estimate and the deadline parameters of the transaction are set.

(5) A transaction cannot use more than one resource at the same time.

3.5. Simulation Results

To make the differences in the performance of the different scheduling strategies obvious two periodic dummy transactions were added to the system. This is justified, since, real-time systems do have certain background tasks which are not directly connected to the real-time application. The following are the dummy transactions and their characteristics:

(1) Dummy1: Low system priority, Tight deadline.

(2) Dummy2: High system priority, Loose deadline.

The simulation results can be grouped into three cases:

(1) Case 1: Dummy1, but not Dummy2, is activated.

(2) Case 2: Dummy2, but not Dummy1, is activated.

(3) Case 3: Both Dummy1 and Dummy2 are activated.

To quantitatively evaluate the results of a particular scheduling strategy, we calculate its *figure of merit* as follows:

$$\textit{figure of merit} = \sum_{Transactiontypes} (\%\ success)(System\ Priority)$$

where

$$\%\ success = \frac{(No.\ of\ successful\ completions)}{(No.\ of\ instances\ started)}$$

The system priorities of the different transaction types is shown in the following table.

Transaction Type	System Priority
Track	2
Clean	1
User Interaction	3
Shoot	3
Display Information	3
Dummy1	0
Dummy2	2

The simulation results based on the above performance metric are summarized in the following tables. The entries in the table are either quantitative (figures of merit) or qualitative (good or bad). The qualitative assessment is done by taking into account the visual behavior of the system.

3.5.1. When Preemption is Allowed

Quantitative Assessment:

Scheduling Strategy	Case 1	Case 2	Case 3
FCFS	300	500	511
System Priority	1159	504	500
Earliest Deadline First	1056	1220	828
Least Slack	305	1036	306
Combination	1114	1350	1194

Qualitative Assessment:

Scheduling Strategy	Case 1	Case 2	Case 3
FCFS	bad	bad	bad
System Priority	good	bad	bad
Earliest Deadline First	bad	good	bad
Least Slack	bad	good	bad
Combination	good	good	good

We observe that the FCFS strategy performs poorly in all the three cases. This is because the FCFS strategy does not possess the requisite intelligence to prevent the dummy transactions from using the resources. This causes the more important transactions to miss their deadline.

In Case 1, the dummy transaction activated has low priority but a tight deadline. The scheduling strategy based on system priority can filter out the dummy transaction. But the earliest deadline first and least slack first strategies do process the dummy transaction, thus causing the system to behave poorly.

In Case 2, the dummy transaction activated has high priority but a loose deadline. The earliest deadline first and least slack first strategies can filter out the dummy transaction. But, the scheduling strategy based on system priority does process the dummy transaction, thus causing the system to behave poorly.

In Case 3, dummy transactions of both kinds are activated. The Combination strategy works well since it uses information about system priority as well as information about the timing constraints while making its scheduling decision.

Thus, we observe that adding intelligence to the scheduling strategy does improve the system performance.

3.5.2. When Preemption is Not Allowed

Quantitative Assessment:

Scheduling Strategy	Case 1	Case 2	Case 3
FCFS	400	1200	608
System Priority	400	600	600
Earliest Deadline First	400	600	600
Least Slack	400	600	600
Combination	400	600	600

Qualitative Assessment:

Scheduling Strategy	Case 1	Case 2	Case 3
FCFS	bad	bad	bad
System Priority	bad	bad	bad
Earliest Deadline First	bad	bad	bad
Least Slack	bad	bad	bad
Combination	bad	bad	bad

As seen above, in general, scheduling strategies perform poorly when preemption is not allowed. From the output of the simulation runs it is observed that almost all of the track transactions miss their deadlines, implying that the operator's console is empty most of the time. Due to this, the clean transactions trivially complete, since they have no emitter files to clean. But, it is almost impossible to start any transactions to shoot or display information about objects. Thus, the entire purpose of the pulse detection system is defeated.

4. Conclusion

Real-time database systems have timing and validity constraints associated with transactions. To ensure that such a system completes as many transactions as possible without violating their timing and validity constraints, its scheduling strategy should have the following characteristics. First and foremost, the scheduling strategy should be dynamic. Second, it should use the timing and validity information associated with transactions and the database. Third, the scheduling strategy should be used at all places where there is resource contention. Fourth, preemption should be allowed wherever possible.

In this project, we described a dynamic scheduling strategy for transactions in real-time database systems. The scheduling strategy uses timing and validity information about transactions and data objects to calculate dynamic priorities of transactions. These priorities are then used to make scheduling decisions at all places where transactions contend for scarce resources. The extra information used by the scheduler enables it to schedule transactions intelligently so that the system completes as many critical transactions as possible.

For any scheduling strategy to be used in industry, it has to be supported by an extensive round of experimentation with a *real-life* application. The simulation study conducted in this project used a pulse detection system as a real-life, real-time database application. The simulation results obtained showed that scheduling strategies for real-time database transactions can be made more intelligent by making use of extra information about transactions such as their system priority, resource requirements and timing constraints.

REFERENCES

[Abbo88] Abbott, A., Garcia-Molina, H., "Scheduling Real-time Transactions: a Performance Evaluation", Proceedings of the 14th VLDB Conference, 1988.

[Abbo89] Abbott, A., Garcia-Molina, H., "Scheduling Real-time Transactions with Disk Resident Data", Proceedings of the 15th VLDB Conference, 1989.

[Bern87] Bernstein, P.A., Hadzilakos, V., Goodman, N., "Concurrency control and recovery in database systems", Addison- Wesley, 1987.

[Hale89] Haleen, B. A., "SDEX/20 and 43RSS: Navy Standard Operating Systems", Proceedings of the 1989 Workshop on Operating Systems for Mission Critical Computing, 1989.

[Huan89] Huang, J., Stankovik, J. A., "Experimental Evaluation of Real-Time Transaction Processing", To appear in Real-Time Systems Symposium, December 1989.

[Sha8] Sha, L., Rajkumar, R., Lehoczky, J., "Concurrency Control for Distributed Real-Time Databases", ACM SIGMOD Record, 17, 1, March 1988.

[Son88] Son, S. H., "Real-Time Database Systems: Issues and Approaches", ACM SIGMOD Record, 17, 1, March 1988.

[Song89] Song, X., Liu, J., "Performance of a Multiversion Concurrency Control Algorithm in Maintaining Temporal Consistency", Working paper, University of Illinois, 1989.

[Stan88A] Stankovic, J. A., "Real-Time Computing Systems: The Next Generation", COINS Technical Report 88-06, Department of Computer and Information Science, Univ. of Massachusetts, Amherst, Ma.

[Stan88B] Stankovic, J. A., Zhao, W., "On Real-Time Transactions", ACM SIGMOD Record, 17, 1, March 1988.

[Stra89] Strayer, W., T., Dempsey, B.,J., Weaver, A., C., "Making XTP Responsive to Real-Time Needs", The University of Virginia, Department of Computer Science, Technical Report, TR-89-18, November 1989.