Scheduling Using Dynamic Priority in Real-Time Database Systems

Prasad Wagle and Sang H. Son

Computer Science Report No. TR-90-03
March 2, 1990

Scheduling Using Dynamic Priority in Real-Time Database Systems*

Prasad Wagle
Sang H. Son

Department of Computer Science
University of Virginia
Charlottesville, Virginia 22903

Abstract

Real-time database systems have 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 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. In this paper we describe an intelligent scheduling strategy for scheduling transactions in real-time database systems. The scheduling strategy uses additional timing information about transactions and the database to enhance the system's ability to meet transaction deadlines.

In real-time database systems, if a data object is not updated for a long time, transactions that access it may consider it to be out-of-date. We introduce the concept of *validity* to quantify this notion of the age of a data object. We also show how to incorporate validity information in the scheduling strategy.

---

* This work was supported in part by ONR under contract N 0014-88-K-0245 and by IBM FSD under University Agreement WG-249153.

Table of Contents

1. Introduction 1
2. Information required for intelligent scheduling 3
3. Scheduling issues 5
4. Real-time Database Scheduler 8
    4.1 Determining eligibility 8
    4.2 Assigning dynamic priorities 9
    4.3 Making the final scheduling decision 13
5. Periodic Transactions and Multiversion Databases 14
6. Conclusion 16
References 16

1. Introduction

It has been recognized that database systems are becoming increasingly important in real-time systems. The distinguishing feature of real-time database systems is that transactions in such a system have timing constraints. An important requirement of these systems is to complete as many transactions as possible without violating their timing constraints [Son88]. 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 [Stankovic88]. 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 and predictable. 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 [Bernstein87]. 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.

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.

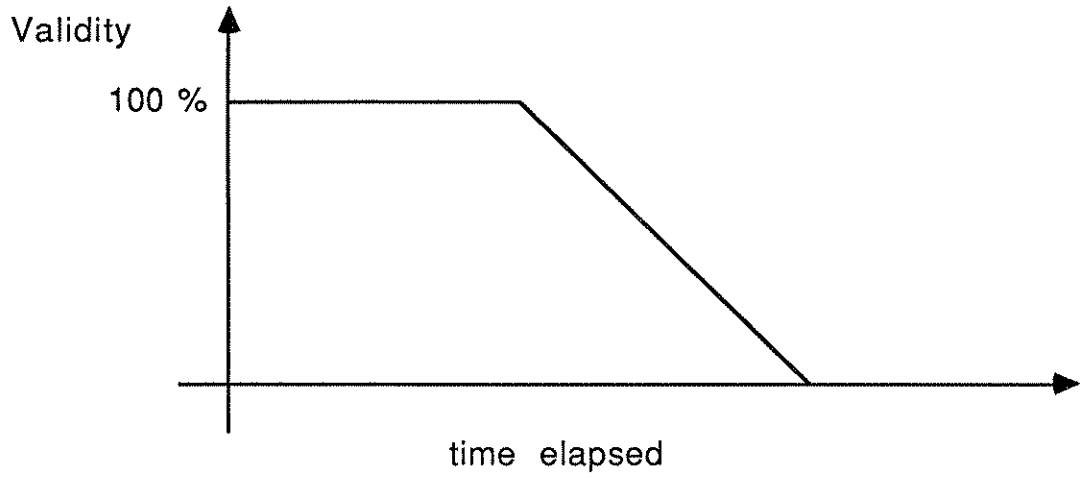

Fig. 1. Validity curve

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.

In this paper, we describe an intelligent scheduling strategy for scheduling transactions in real-time database systems. The strategy uses additional timing information about transactions and the database to enhance the system's ability to meet transaction deadlines. We also introduce the concept of *validity* to quantify the notion of the *age* of a data object. We show how to incorporate this validity information in the scheduling strategy. The remainder of this paper is organized as follows. Section 2 describes the information about transactions and the database which can be used to make intelligent scheduling decisions. Section 3 discusses the issues involved in designing scheduling strategies. Section 4 presents a dynamic scheduling strategy incorporating the extra information discussed in Section 2. Section 5 shows how the scheduling strategy can be extended to periodic transactions and multiversion databases. Section 6 is the conclusion.

2. Information required for intelligent scheduling

Conventional scheduling algorithms rarely use any information about transactions, except priority, while making scheduling decisions. It is possible to use additional information about transactions and the data objects to intelligently schedule transactions. In this section, we discuss the nature of the information required by the scheduling strategy and how 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.

SP System priority: This is the static component of the dynamic priority associated with a transaction. It is a measure of the criticality 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.

RS Read set: This is the set of data objects which the transaction reads

WS Write set: This is the set of data objects which the transaction writes.

A Arrival time: This is the time at which the transaction arrives in the system.

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

E Runtime estimate: 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.

MV Minimum Validity: 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.

RSV Read set validity: 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.

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

T Current time: This is the time at which the scheduling decision is made.

3. Scheduling 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 (T1 and T2) have the characteristics given below. This situation can arise if both T1 and T2

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.

Trans..	Arrival time	Estimate	Deadline (hard)	Operations
T1	0	12	30	read(1)
T2	5	12	20	read(1)

According to an elementary FCFS scheduling strategy, T1 is scheduled first and it completes at time 12. T2 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)

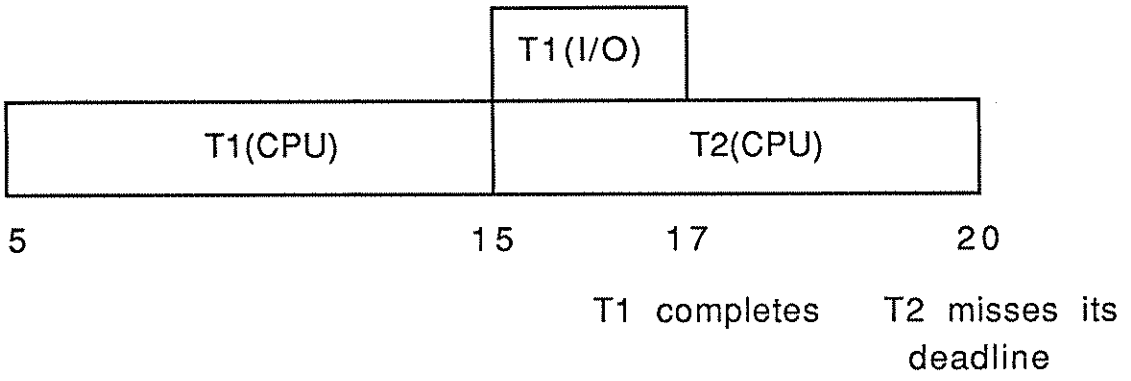

Fig. 2. FCFS Scheduling

If the system is intelligent enough to follow the elaborate scheduling strategy to be discussed in Section 4, T2 would be scheduled first. (According to the least slack method of assigning priorities, T2 has a higher priority than T1, because the slack of T2 is less than the slack of T1.) In that case both transactions would meet their deadlines as shown in Fig. 3.

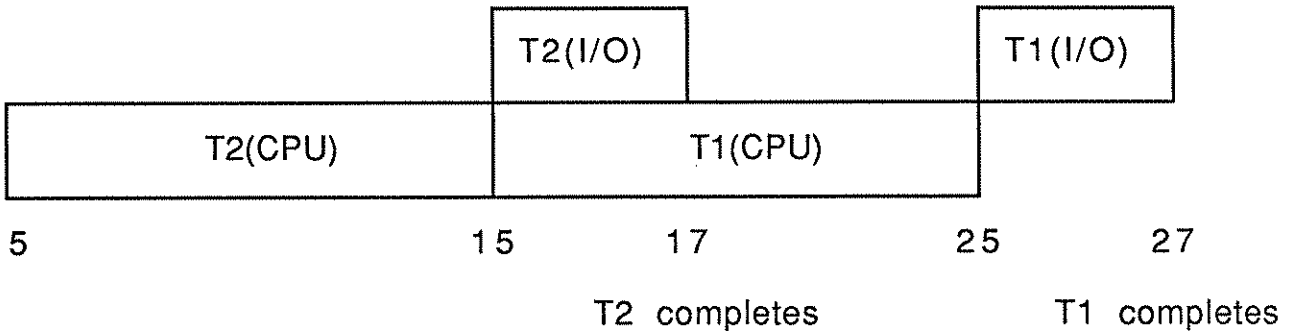

Fig. 3. Intelligent Scheduling

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 transaction 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.

4. Real-time Database Scheduler

The scheduling strategy for transactions in real-time database systems can be decomposed into three sub-parts [Abbott88], [Abbott89]:

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.

4.1 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) $\geq$ (Estimate - Processing time received)

i.e. $(D - T) \geq (E - P)$

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.

4.2 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 [Strayer89]. 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 := DP1 + DP2 + DP3
where,

DP1 := W1 * SP
DP2 := W2 * SDL
DP3 := W3 * SV

The factors involved in determining the dynamic priority of a transaction have constraints closely related to the characteristics of real-time transactions. First, W1 > 0, since if SP increases, DP should increase. Also, if SDL > 0 then W2 < 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, W2 < 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, W2 > 0. Similar discussion applies to W3 and SV.

The relative values of W1, W2, W3 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 W1 would not be very high. Some systems might require that absolutely none of the higher priority transactions be aborted, in which case W1 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 (T1 and T2) have the characteristics given below.

Trans..	Arrival time	Estimate	Deadline (hard)	Min.Valid. (hard)	Operations
1	0	12	30	100%	read(1)
2	0	12	25	50%	read(1)

Let the validity curve for object 1 be as shown in Fig. 4., and the time it was last modified be 0. Let the weights W2 and W3 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.

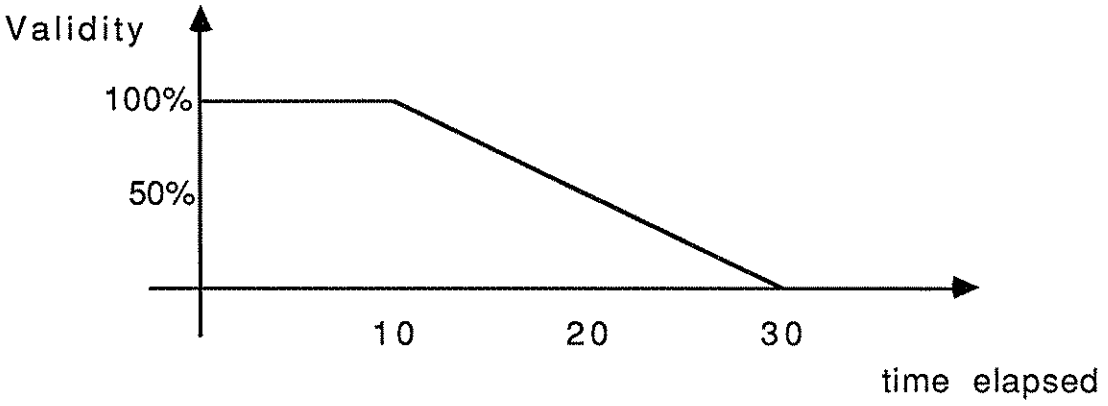

Fig. 4. Validity Curve

If validity constraints are not considered:

In this case, DP := DP1 + DP2. The slack of T1 with respect to deadline is 18. The slack of T2 with respect to deadline is 13. Therefore,

DP2(T1) = -18 and DP2(T2) = -13.

i.e. DP2(T2) > DP2(T1).

Assuming equal system priorities, DP(T2) > DP(T1), implying that T2 would be scheduled first The execution would proceed as shown in Fig. 5. T2 would finish its execution at time 12. Then T1 would start. But, at time 20 the validity of object 1 would be 50%. This would violate the validity constraint of T1, which would have to be aborted.

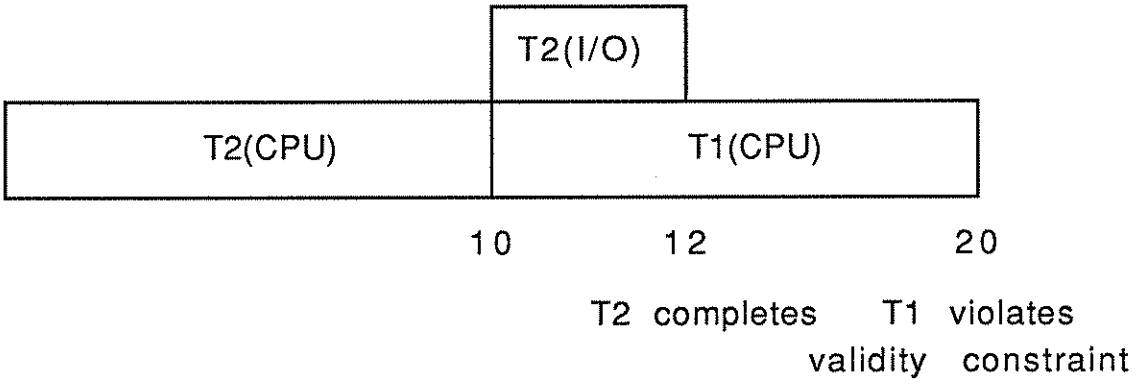

Fig.5. Validity constraints ignored

If validity constraints are considered:

In this case, DP := DP1 + DP2 + DP3. The slack of T1 with respect to *validity* constraints is 10. The slack of T2 with respect to validity constraints is 20. Therefore,

DP3(T1) = -10 and DP3(T2) = -20.

i.e. DP2(T1) + DP3(T1) = -28 and DP2(T2) + DP3(T2) = -33

i.e. DP2(T1) + DP3(T1) > DP2(T2) + DP3(T2).

Assuming equal system priorities, DP(T2) > DP(T1), implying that T1 would be scheduled first. The execution would proceed as shown in Fig. 6. At time 10 the validity of object 1 would be 100%, satisfying T1's validity constraints. Thus T1 would finish its execution at time 12. Then T2 would start. At time 20, the validity of object 1 would be 50%, satisfying T2's validity constraints. Thus T2 would finish its execution at time 22.

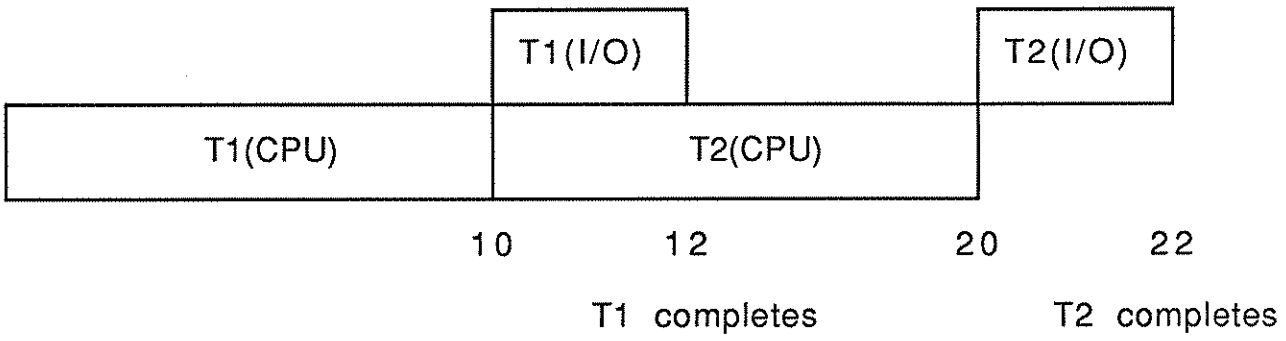

Fig. 6. Validity constraints considered

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

4.3 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(Tr) > MAX( P(Th) , P(Th if preempted) ) THEN
        IF RemainingTime(Th) > Slack(Tr) THEN
                Preempt Th;
        END;
END;
```

where RemainingTime(T_h) = Runtime estimate
- Processing time received by T_h.

5. Periodic Transactions and Multiversion Databases

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 [Haleen89]. 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. 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.

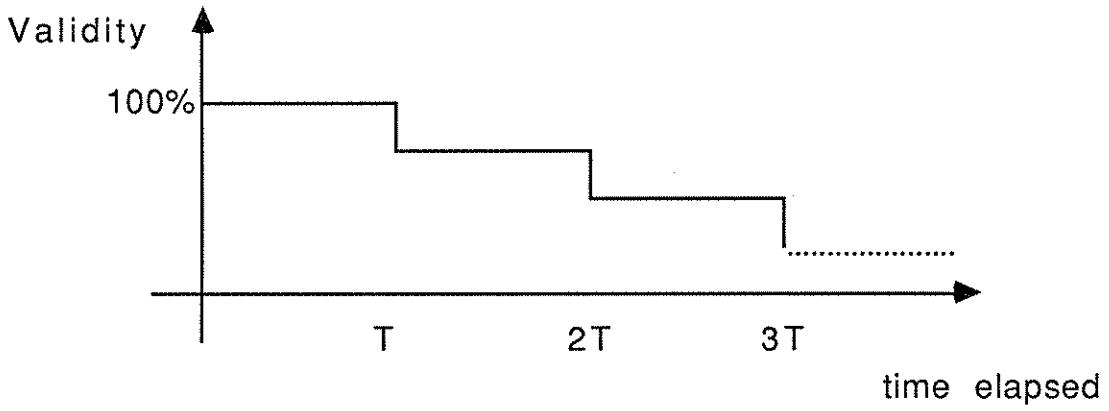

Fig 6. Example Validity curve

If the database supports multiversion concurrency control, then each write operation on a data object produces a new version of that data object. For a read operation, the scheduler has to choose the correct version to be read from the pool of currently existing versions. This reduces the conflict rate between the operations issued by transactions. For example, instead of rejecting a read operation which arrives too late, the scheduler can make it read an older version. The particular version chosen depends on the multiversion concurrency control algorithm being followed. Once a version is chosen, the rest of the scheduling strategy remains the same. For example, if there are more than one transactions waiting to access a version of a data object, the scheduler has to go through the same process of determining eligibility, assigning priorities and making the final decision as

described in Section 4. Overall, the multiversion scheduler at the database level has to reject or delay fewer operations issued by transaction managers than the single-version counterpart.

6. 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 use timing information associated with transactions and the database. This makes the task of scheduling transactions extremely complex. In this paper, we described a scheduling strategy for transactions in real-time database systems. The scheduling strategy uses timing information about transactions and the 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. The scheduling strategy can be extended to incorporate periodic transactions and multiversion databases.

References

[Abbott88] Abbott, A., Garcia-Molina, H., "Scheduling Real-time Transactions: a Performance Evaluation", Proceedings of the 14th VLDB Conference, 1988.

[Abbott89] Abbott, A., Garcia-Molina, H., "Scheduling Real-time Transactions with Disk Resident Data", Proceedings of the 15th VLDB Conference, 1989.

[Bernstein87] Bernstein, P.A., Hadzilakos, V., Goodman, N., "Concurrency control and recovery in database systems", Addison-Wesley, 1987.

[Haleen89] Haleen, B. A., "SDEX/20 and 43RSS: Navy Standard Operating Systems", Proceedings of the 1989 Workshop on Operating Systems for Mission Critical Computing, 1989.

[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.

[Stankovic88] Stankovic, J. A., Zhao, W., "On Real-Time Transactions", ACM SIGMOD Record, 17, 1, March 1988.

[Strayer89] 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.