Analysis of Deadline-Driven Scheduling Algorithms Using Function-Driven Techniques

W. Timothy Strayer

Computer Science Report No. TR-92-11
May 4, 1992

This work is supported in part by the U.S. Naval Ocean Systems Center and the Office of Naval Research under contract N00014-91-J-1514.

Analysis of Deadline-Driven Scheduling Algorithms Using Function-Driven Techniques

W. Timothy Strayer

University of Virginia
Charlottesville, Virginia
wts4x@virginia.edu
May 4, 1992

Abstract

Real-time systems must be responsive to task deadlines, hence deadline-driven scheduling algorithms pervade real-time scheduling. Scheduling algorithms, whether for real-time system or not, often focus one or only a few of the task attributes or system characteristics for discriminating among competing tasks. The task set is comprised of tasks that either are dependent on the particular attributes or are forced to be. Expression of scheduling algorithms using function-driven techniques, in particular the importance abstraction [STRA92] employs functions to profile task importance; the scheduler greedily chooses the most important task at every point in time. We explore the use of this abstraction for the class of task sets that have at least some deadline-driven tasks but are not limited to tasks with only deadlines as attributes or which are dependent only on deadlines for their importance.

1. Introduction

Scheduling algorithms impose an ordering on a set of tasks such that a scheduling policy is maintained. This policy specifies what properties the schedule should have. Ideally a system designer would set a policy based on the goal of the system, then construct an algorithm to implement that policy. The algorithm is then analyzed to determine if indeed the schedules produced meet the properties dictated by the scheduling policy. Other questions about the schedules are also asked. These include: Is a given task set schedulable? Under what conditions will a certain task be serviced? Under what conditions will a task fail to meet its deadline. Unfortunately, the progression from policy to algorithm to analysis is difficult since many policies, if not sufficiently restricted, require NP-Complete algorithms to implement them. Consequently, the progression is typically from an algorithm with rich analytical results to a policy that can exploit these results. The task sets are then forced to fit the profile of the algorithm. A example of such a well-studied algorithm is the rate monotonic algorithm introduced in [LIU73]; while this algorithm is designed for periodic task sets only, much work has been done to modify the algorithm to include aperiodic tasks, tasks with a secondary level of criticality, and task sets with unknown sizes. The techniques used to modify rate monotonic theory to include these cases are surveyed in [SHA90].

The *importance abstraction* [STRA92] is a framework for describing scheduling policies by focusing on how importance each task is to the system. Every system has a goal and the tasks within the system are processed with the intent of meeting the system goal. A task within the system is viewed as “important” to the system *vis-a-vis* how that task can contribute to accomplishing the system goal. As the system progresses and its state changes, various tasks become more or less important to the system. The importance abstraction is a framework for expressing those conditions under which tasks within a system become important to the system.

The importance abstraction includes within its framework a set of functions called *importance functions* which describe the tasks within the system, and a scheduler that uses the importance functions to determine which tasks should receive service. By using this abstraction to consider scheduling problems, we shift the emphasis from the analysis of the scheduling algorithm to the analysis of a set of functions. Other important efforts in function-based scheduling include [BERN71], [RUSC77], and [JENS85].

In this paper we examine the problem of meeting task deadlines. The nearest deadline first (NDF) algorithm is known to meet all deadlines if any schedule exists that meets all deadlines. Since we can emulate this algorithm within the importance abstraction, we prove this statement for tasks with arbitrary arrivals. We continue by relaxing various restrictions on the task set and including various other attributes; in all cases we use the importance abstraction as the framework within which we prove or construct results.

2. Importance Functions

If the importance of a task could be quantified at every point in time, it could be expressed as a function over time to profile a task's importance to the system. Since the importance of a task depends upon the state of the system and the attributes of that task, the pertinent components of the system state and the attributes of the task must be the parameters to the function. If we can identify each possible task in the system, and under what conditions that task will become important to the system, we can associate with each task a function that reflects the task's importance. Hence, for each task i in the task set T, an *importance function* $I_i(t)$ is constructed that reflects the conditions for task i to be important.

An importance function for a task returns a value that ranks that task among all other active tasks competing for the processor according to how important it is for that task to be processed at that moment in time. The set of importance functions are used to schedule the tasks as well as represent the tasks in analysis done on the scheduling policy imposed by the importance functions.

When a set of importance functions has been associated with a task set, the tasks within that task set are scheduled according to the values of the importance functions. Let I_T be the set of all importance functions for the task set T. At every point in time the scheduler must evaluate the function $M{:}I_T \rightarrow T$, where M takes the set of importance functions and returns a task. Without loss of generality we assume that the tasks in T are indexed, in no particular order, so that a task is identified by its index. The function M evaluates each importance function in the set I_T and returns the task $i \in T$ whose importance function has the maximum value at that point in time. If, at some point in time, the scheduler finds that two or more tasks are most important simultaneously, the scheduler will arbitrarily choose one of those tasks as the task to receive service, and will continue to allow that task to receive service until some other task becomes most important. This model assumes that all of the tasks in the task set are preemptable.

We can express the actions of the scheduler with some mathematical constructs. The boolean relation $(M(I_T) = i)$ returns the value 1 if the most important task at the time of evaluation is the task i, and the value 0 otherwise.[1] By using this boolean relation as a function of time, we can determine how long a specific task has been most important over a certain period. Let the value $w_i\big|_{t_1}^{t_2}$ represent the amount of work applied to the task i from time t_1 to time t_2. The equation

$$w_i\big|_{t_1}^{t_2} = \int_{t_1}^{t_2} (M(I_T) = i)\, dt \qquad \text{Eq 1}$$

shows the relationship between the importance functions and the amount of work done to a particular task. This equation states that the amount of work received by task i over the period from t_1 to t_2 is equal to the amount of time that the function M returns task i from time t_1 to t_2.

An interesting and important aspect of the importance abstraction is the ability to *emulate* traditional scheduling policies within its framework. The importance abstraction is said to emulate an arbitrary scheduling policy in that it makes exactly the same scheduling decisions at exactly the same time. In the importance abstraction the act of scheduling always remains the same: choose the most important task to perform at each decision point;

[1]. The convention of using a boolean expression within a set of parentheses to denote a function that returns 1 if the boolean expression is true and 0 if it is false is used in Graham, Knuth, and Patashnik's book **Concrete Mathematics** (1988); they attribute the convention to Iverson in the programming language APL.

the various scheduling policies are actually implemented by defining appropriate importance functions. The importance functions must be defined in such a way that a task becomes most important at precisely the same instant as the conventional scheduling policy would have chosen it.

3. Meeting Deadlines

An important question that we ask about task sets within real-time systems is: Does there exist a schedule that will ensure that all deadlines of all tasks are met? We know from [LIU73] that the nearest deadline first (NDF) algorithm will meet all deadlines for a periodic task set whose fixed size is known *a priori*. It is widely held that NDF will meet all deadlines of any fixed size task set, periodic or aperiodic, if any schedule exists that can meet all deadlines. Readily, NDF is sufficient for meeting all deadlines for a feasible task set, but it is not necessary; given any feasible task set a permutation of the tasks, as long as no task requires work past its deadline, is also a feasible schedule. Since we assume preemptable tasks, there are an infinite number of such permutations, and hence an infinite number of schedules that meet all deadlines.

In this section we prove that the nearest deadline first algorithm will meet all deadlines for a fixed size aperiodic task set if any schedule exists that meets all deadlines. We then consider the same question for task sets with arbitrary sizes. Next we consider task sets where the tasks have both a deadline and a criticality. Again we consider deadlines and criticality, but this time where the task set contains tasks dependent on either one or the other but not both. Finally, we consider the heterogeneous task set where tasks may have a deadline, a criticality, or both, and find that we can construct an importance function for a task within this task set such that all deadlines of the most critical tasks are met if any schedule exists that will meet them.

3.1. Nearest Deadline First

The nearest deadline first scheduling algorithm ranks all tasks by the nearness of their deadlines: a task i with a deadline d_i that is nearer than the deadline d_j of task j is more important. We construct importance functions to emulate this algorithm such that:

$$d_i < d_j \Rightarrow I_i(t) > I_j(t),\ \forall\,(i, j \in T) \qquad \text{Eq 2}$$

There are many functions that satisfy this property—one such function is given by:

$$\forall\,(i \in T)\,, \qquad I_i(t) \;=\; \begin{cases} 0,\ \text{if}\ (t < a_i) \\ (d_i - t)^{-1},\ \text{if}\ (d_i > t \geq a_i) \\ 0,\ \text{if}\ (d_i \leq t) \end{cases} \qquad \text{Eq 3}$$

Note that there are two discontinuities, one at the moment that the task becomes active and one at the moment it misses its deadline. Also note that the task becomes infinitely important just as the deadline is reached.

Liu and Layland [LIU73] have shown that, if deadlines can be met for a given task set, they will be met using the nearest deadline first policy. However, Liu and Layland show this for a set of periodic tasks by proving that nearest deadline first will schedule tasks to meet deadlines if the *utilization factor* (the sum over all tasks of the ratios of work required to length of period) for the task set is 1 or less. Unfortunately, the utilization factor proof in [LIU73] only holds for periodic task (a counterexample: task 1 has arrival time $a_1 = 5$, work required $w_1 = 10$, deadline $d_1 = 15$, and task 2 has $a_2 = 15$, $w_2 = 10$, $d_2 = 25$; the utilization factor is 2, yet the task set is feasible).

To show that the nearest deadline first algorithm will meet all deadlines for an aperiodic task set if there exists any schedule which can meet all deadlines, we must show that the completion time c for each task is always less than or equal to the task's deadline *d*; that is, for each task *i* with deadline d_i, $c_i \le d_i$. We prove this by constructing an expression for the conditions under which any schedule of *n* tasks will meet all deadlines. For a schedule to meet every deadline in the task set the schedule must ensure that the following is true for all points in time:

$$\underset{1 \le j \le n}{\text{AND}} \left(\sum_{i=1}^{j} w_i \Big|_t^{\infty} \le \max\,(d_j - t, 0) \right) \qquad \text{Eq 4}$$

where $n = |T|$ is the size of the task set. This is actually a set of conditions, all of which must be true. Consider $t = 0$. For $j = 1$, the amount of work done on task 1 over all time must not exceed its deadline. For $j = 2$, the amount of work done on task 2 in addition to the work done on task 1 must not exceed the deadline d_2. For $j = n$, where $n = |T|$, the amount of work done on all *n* tasks must not exceed the deadline of task *n*.

Theorem 1

Given a task set *T* for which there exists some schedule that meets all deadlines, then a schedule imposed by the nearest deadline first algorithm will also meet all deadlines.

Proof

Assume the tasks of task set *T* are scheduled by a set of importance functions for which Eq 5 is a property. Let *T* be ordered such that $d_1 \le d_2 \le \ldots \le d_n$. Let task *k* be the lowest indexed task for which $c_k > d_k$, where c_k is the completion time and d_k is the deadline for task *k*.

There are two cases. First, if there is no idle time between time 0 and time c_k, then the sum of all of the work done on all tasks over that interval is the length of the interval and equals c_k. Therefore:

$$\sum_{i=1}^{n} w_i \Big|_0^{c_k} = c_k \qquad \text{Eq 5}$$

Since the property given in Eq 5 holds for this task set, only the tasks whose deadlines are d_k or prior are serviced over the interval 0 to c_k; we may rewrite Eq 7 as:

$$\sum_{i=1}^{k} w_i\Big|_0^{c_k} = c_k \qquad \text{Eq 6}$$

Also, these tasks are run to completion before task k is completed, so we can replace $w_i\big|_0^{c_k}$ with w_i:

$$\sum_{i=1}^{k} w_i = c_k \qquad \text{Eq 7}$$

But the k^{th} condition of Eq 6, for $t = 0$, states:

$$\sum_{i=1}^{k} w_i\Big|_0^{\infty} \le \max\,(d_k - 0, 0) = d_k \qquad \text{Eq 8}$$

Since $w_i\big|_0^{\infty}$ equals all of the work required by the task, this expression in Eq 10 can be replace by the value w_i:

$$\sum_{i=1}^{k} w_i \le d_k \qquad \text{Eq 9}$$

By substitution of Eq 9 into Eq 11, we arrive at $c_k \le d_k$, a contradiction of our initial assumption that $c_k > d_k$.

For the second case, if there is at least one gap of idle time between time 0 and time c_k, let t_g be the time when the last gap ends so that on the interval t_g to c_k there is no idle time. The work done over that interval must sum to the length of the interval:

$$\sum_{i=1}^{n} w_i\Big|_{t_g}^{c_k} = c_k - t_g \qquad \text{Eq 10}$$

Since Eq 5 holds, no tasks of index greater than k will be serviced during this interval, so we can change the upper limit of the summation. Also, since each task with index k or less will finish before time c_k, we can replace the expression $w_i\big|_{t_g}^{c_k}$ with $w_i\big|_{t_g}^{\infty}$:

$$\sum_{i=1}^{k} w_i\Big|_{t_g}^{\infty} = c_k - t_g \qquad \text{Eq 11}$$

The k^{th} condition of Eq 6, for $t = t_g$, yields:

$$\sum_{i=1}^{k} w_i\Big|_{t_g}^{\infty} \leq \max\,(d_k - t_g, 0) \qquad \text{Eq 12}$$

By substitution of Eq 13 into Eq 14, we arrive at:

$$\begin{gathered} c_k - t_g \leq d_k - t_g \\ c_k \leq d_k \end{gathered} \qquad \text{Eq 13}$$

Again, we find the contradiction.

Therefore, if there exists a schedule which can meet all deadlines in a task set, then the schedule imposed by the importance functions which emulate the nearest deadline first algorithm will also meet all deadlines. Since the importance functions and the algorithm impose the same schedule, then the result holds for the nearest deadline first algorithm as well.

∎

For rate monotonic, each task is instantiated exactly once during the task's period. This instantiation must complete before its period expires and the new instantiation is created. We can therefore think of each instantiation of a task as a separate task, and the end of the period as that task's deadline. In this sense rate monotonic is similar to the nearest deadline first algorithm where the deadline d_i is given by $d_i = a_i + T_i$, where T_i is the period for task *i*.

3.2. Deadline-Driven Tasks with Arbitrary Task Set Size

Theorem 1 assumes that the task set *T* has a constant cardinality *n* and is known *a priori*. In a system where the task set *T* can not be known *a priori*, and where the cardinality is not known to be a constant (i.e., there may be arbitrary future arrivals), we can not prove that all deadlines will be met. We can, however, create a test which will identify as early as possible when a task will miss its deadline.

Let tasks be requested at arbitrary times such that the request time for task *i* is less than or equal to the arrival time for task *i*; that is, $req_i \le a_i$. Index the tasks such that, for all tasks $i, j \in T$

$$i > j \Rightarrow (req_i < req_j) \vee ((req_i = req_j) \wedge (a_i < a_j)) \qquad \text{Eq 14}$$

Thus the tasks are indexed primarily by when they are requested, and secondarily be when they arrive.

We need to define a few functions for convenience. Let $D{:}T \rightarrow N$ be a function that takes a task and returns a natural number representing the task's current order with respect to deadline nearness. If task *i* has the j^{th} nearest deadline, then $D(i) = j$. Let $D'{:}N \rightarrow T$ be the inverse function which, given a natural number *j*, returns the task index whose deadline is currently the j^{th} nearest. Let $n(t)$ be a function that returns the cardinality of the set *T* at time *t*. The condition for meeting all deadlines for the task set *T* at time *t* is:

$$\underset{1 \le j \le n(t)}{\text{AND}} \left(\sum_{i=1}^{D(j)} w_{D'(i)}\Big|_t^{\infty} \le \max(d_j - t, 0) \right) \qquad \text{Eq 15}$$

This condition is similar to the condition given in Eq 6. This condition states that, at some time *t* and for all tasks *j* from 1 to the current cardinality of the task set *T*, the sum of the work required by all tasks whose deadlines are priori to task *j* must be less than or equal to the amount of time between the current time and task *j*'s deadline. We define the term *overload* to be the state of the task set at time *t* such that Eq 17 is not true.

Theorem 2

Let *T* be an arbitrarily large task set containing tasks with arbitrary request times. The nearest deadline first algorithm will meet all deadlines if any algorithm can meet all deadlines.

Proof

Assume that a system requests work on tasks at arbitrary time such that the size of the task set is not known *a priori*. Assume that task *k* is requested at time req_k, and at that time an overload occurs such that some task m can not meet its deadline using the nearest deadline first algorithm. At time req_k we can construct a task set T_k which includes all tasks requested from time 0 to time req_k. Let these tasks be indexed according to the nearness of their deadlines such that $i < j \Rightarrow d_i < d_j$. By application of Theorem 1 we know that no algorithm can meet all deadlines if the nearest deadline first algorithm can not meet all deadlines.

∎

3.3. Meeting Critical Deadlines, with Arbitrary Task Set Size

One of the problems with a pure nearest deadline first algorithm is that the tasks are not otherwise ranked in the presence of missed deadlines such that the most critical tasks are given preference at the expense of the least critical. The importance abstraction can easily express this bilevel ranking, where the nearest deadline first policy is augmented by considering criticality measures associated with each task. Let us call this new for of nearest deadline first the nearest critical deadline first (NCDF). From the representation of the NCDF policy within the importance abstraction we seek the conditions under which a given task *k* will meet its deadline, and from that prove that NCDF maximizes a quantity based on the criticality of the tasks serviced.

Let each task *i* have two attributes: the deadline d_i and a criticality p_i. Assume that the criticality p_i is an element of *L*, where *L* is the set of natural numbers in the range MINCRIT to MAXCRIT. To construct a set of importance functions which will implement the NCDF policy we define a few auxiliary functions for convenience. Define the function $Over: \{T\} \times time \rightarrow Boolean$ as:

$$Over\,(T, t) \;=\; \underset{1 \le j \le |T|}{\text{AND}} \left(\sum_{i=1}^{D(j)} w_{D'(i)} \Big|_t^{\infty} > \max(d_j - t, 0) \right) \qquad \text{Eq 16}$$

The function *Over* returns one if the task set *T* will not meet all deadlines at time *t*, zero otherwise. Note that this is a functional representation of the conditions from Eq 17. Define

Crit:$L \rightarrow \wp(T)$ as a function that takes the criticality level from the set *L* and returns the subset of *T* that share this criticality level. Finally, we define a function *InMostCrit*:$T \times time \rightarrow Boolean$ that takes a task and a time and returns one if the task is in the set of tasks whose deadlines will be met because they are among the most critical at that time, and returns zero otherwise. Now for the importance function:

$$\forall (i \in T), \qquad I_i(t) = \begin{cases} 0, \text{ if } (t < a_i) \\ (d_i - t)^{-1}, \text{ if } ((d_i > t \geq a_i) \wedge InMostCrit(i, t)) \\ 0, \text{ otherwise} \end{cases} \qquad \text{Eq 17}$$

Given a task *k* with an importance function defined as above, we seek the conditions under which this task *k* will meet its deadline. Since we are now considering a task set with arbitrary future arrivals, we can not predict *a priori* that task *k* will meet its deadline; rather, we can show the conditions necessary at certain points in time for task *k* to meet its deadline. At time req_k, task *k* is schedulable if the following is true:

$$\left(\sum_{i=1}^{D(k)} (w_{D'(i)} \Big|_{req_k}^{\infty}) (p_{D'(i)} \geq p_k) \right) \leq d_k - req_k \qquad \text{Eq 18}$$

We must check this condition not at time req_k but every time a request for service is made, hence:

$$\underset{t = req_j | \ (req_k \leq req_j \leq d_k)}{\text{AND}} \left(\left(\sum_{i=1}^{D(k)} (w_{D'(i)} \Big|_{t}^{\infty}) (p_{D'(i)} \geq p_k) \right) \leq (d_k - t) \right) \qquad \text{Eq 19}$$

This expression states that, for each time *t* that a new task arrives between the request of task *k* and task *k*'s deadline, the following must be true: the sum of the work remaining for tasks whose deadlines are nearer than task *k*'s and whose criticality is at least as great as task *k*'s must be less than or equal to the difference between task *k*'s deadline and the time we are considering.

Biyabani *et al*. explore this kind of bilevel ranking in [BIYA88]. Most notably they offer a new sematic for the term *guarantee* that reflects the uncertainty of the future task set composition. They state that at request time a task is guaranteed to meet its deadline if (1) it is among the most critical tasks in the current task set, and (2) the arrivals of subsequent tasks do not cause this task to leave the set of the most critical tasks. The system guarantees that the most critical tasks will meet their deadlines; however, we can not predict which tasks will be in the set of most critical tasks.

We constructed the importance functions so that only the most critical tasks are serviced to completion. When the system presents more tasks than can be serviced without missing a deadline, some tasks must be pruned. The condition *InMostCrit* is used within the importance functions of Eq 19 to do this pruning. We can quantify how well the goal of meeting most critical deadlines is being met by summing the criticality values for all tasks whose deadlines have been met by time *t*. Define the quantity *CritCount* as:

$$CritCount = \sum_{i=1}^{n(t)} \left(w_i \big|_{a_i}^{d_i} \geq w_i \right) p_i \qquad \text{Eq 20}$$

When the work done on a task *i* is greater than or equal to the work required, the criticality value of task *i* is added to the criticality count *CritCount*. Because the NCDF policy is greedy, we expect the *CritCount* for the schedule produced to be optimal among all policies. The following lemma supports a theorem that proves that NCDF is optimal with respect to maximizing this quantity.

Lemma 1

Any task set that is schedulable by the nearest deadline first (NDF) policy is also schedulable by the NCDF policy

Proof

Let *T* be a task set that is schedulable by NDF. Thus, by Eq 17 we know that, for all time *t*, the following is true:

$$\underset{1 \le j \le n(t)}{\text{AND}} \left(\sum_{i=1}^{D(j)} w_{D'(i)} \Big|_t^{\infty} \le \max(d_j - t, 0) \right) \qquad \text{Eq 21}$$

Since the only difference in NDF and NCDF is the presence of the condition *InMostCrit*, as long as *InMostCrit* is true for some task *i* over all time *t*, then task *i* will be scheduled by both algorithms at exactly the same time, for exactly the same duration, and having exactly the same completion time. Let task *k* be a task schedulable by NDF but not by NCDF. Thus, *InMostCrit*(*k*,*t*) must not be true for some time *t*. This implies by Eq 18 that

$$\sum_{i=1}^{D(k)} w_{D'(i)} \Big|_t^{\infty} > d_k - t \qquad \text{Eq 22}$$

But from Eq 23 we know that

$$\sum_{i=1}^{D(k)} w_{D'(i)} \Big|_t^{\infty} \le d_k - t \qquad \text{Eq 23}$$

This is a contradiction.

∎

Theorem 3

The NCDF policy maximizes the criticality count *CritCount* among all scheduling policies.

Proof

Assume that there exists some scheduling policy *A* that, at some time t, produces a schedule that has a higher value for *CritCount* than NCDF. Let T_A be the set of tasks scheduled by policy *A* by time *t*, and T_{NCDF} be the set of tasks scheduled by NCDF. If these tasks are equal then their *CritCount*s must also be equal and thus we have a contradiction.

If the task sets are not equal, then the set of tasks chosen by policy *A* must contain some tasks not chosen by NCDF. For the quantity *CritCount* of T_A to be higher than that for T_{NCDF}, policy *A* either scheduled more tasks or instead scheduled tasks of greater criticality. By Theorem 1 we know that the task set T_A can be scheduled by NDF. By Lemma 1 we know that any task set schedulable by NDF is also schedulable by NCDF. Therefore, policy *A* could not have scheduled more tasks than NCDF; instead, to have a higher value for *CritCount*, policy *A* must have scheduled different, more critical tasks.

Since, at every point in time, NCDF chooses the most critical task with the nearest deadline, any more critical tasks chosen by policy *A*, and therefore schedulable by both NDF and NCDF, would have also been chosen by NCDF. Thus policy *A* could not have scheduled more critical tasks than NCDF, and a contradiction results.

∎

3.4. Meeting Deadlines for Heterogeneous Task Sets

Consider a task set that contains some tasks that are only deadline-driven and some tasks that are only priority-driven. Because the priority-driven tasks do not have a time constraint, most policies schedule the deadline-driven tasks first and use any remaining processor cycles to service the priority-driven tasks. Policies of this type are easily constructed within the importance abstraction by using the following importance functions: Let T_d be the subset of T that are deadline-driven tasks and T_p be the subset of T that are priority-driven tasks. Let p be equal to MAXCRIT. The importance functions for both types of tasks are given by:

$$\forall (i \in T) \qquad I_i(t) = \begin{cases} (d_i - t)^{-1} + p, & \text{if } (i \in T_d \wedge a_i \le t < d_i) \\ p_i, & \text{if } (i \in T_p \wedge a_i \le t) \\ 0, & \text{otherwise} \end{cases} \qquad \text{Eq 24}$$

Since the importance of a deadline-driven task is always higher than the importance of any priority-driven task, Theorems 1 and 2 from the previous sections still hold with respect to T_d. A characteristic of schedules produced using this set of importance function is that priority-driven tasks are always deferred until there are no deadline-driven tasks in the set to be serviced. Thus, as a consequence of trying to meet the deadlines of the tasks of subset T_d the priority-driven tasks must wait until there are no active deadline-driven tasks.

Consider a system that must meet all deadlines as well as attempt to minimize the average response time to the priority-driven tasks. If there is no stated benefit from servicing the deadline-driven tasks sooner rather than later, as long as the deadline is met if it can be met, then we want a schedule that defers deadline-driven tasks to the last possible moment. Unfortunately, deferring deadline-driven tasks without *a priori* knowledge of future task arrivals may indeed cause some deadlines to be missed where not deferring the tasks (as with NDF and NCDF) would have met the deadlines. Consequently there must be restrictions on the task set in order to explore a policy that uses procrastination of deadline-driven tasks to reduce the response time for priority-driven tasks.

Clearly, the most conservative restriction is to require a fixed size task set that is known *a priori*. Let each of the n tasks in T be indexed thus: tasks 1 through *m* are elements of T_p and are ordered by increasing arrival times, and tasks *m*+1 through *n* are the elements of T_d and are ordered by increasing deadlines. To keep the procrastination of deadline-driven tasks from causing some task's deadline to be missed, the latest possible starting time for a given task *i* such that it can still meet its deadline must be determined. Define s_i to be this latest possible starting time:

$$s_i = \min_{i \le j \le n} \left(d_j - \sum_{k=i}^{j} w_k\right) \qquad \text{Eq 25}$$

The restriction of a fixed size task set known *a priori* can be relaxed to allow arbitrary arrivals with conditions placed on when the request for service for each task is made. Assume that the tasks are now indexed by their request times such that $i < j \Rightarrow req_i < req_j$. The restriction must ensure that, if any two tasks' deadlines are sufficiently close together, then the tasks must be requested appropriately. Recall that $D'(i)$ returns the index of the task whose deadline is the ith nearest. If the difference between the deadlines of tasks $D'(i+1)$ and $D'(i)$ is less than the quantity $w_{D'(i+1)}$, then it is possible for task $D'(i)$ to be deferred in such a way as to interfere with the meeting of task $D'(i+1)$'s deadline. Both tasks can be taken into account if the task whose deadline is later is known about at the same time as or before the task whose deadline is nearer. Specifically, the request times for tasks $D'(i)$ and $D'(i+1)$ must be ordered such that:

$$d_{D'(i+1)} - w_{D'(i+1)} < d_{D'(i)} \Rightarrow req_{D'(i+1)} \le req_{D'(i)} \qquad \text{Eq 26}$$

Rewriting Eq 27 to reflect indexing tasks by request time order, the latest starting time for some task i is given by:

$$s_i = \min_{D(i) \le j \le |T_d|} \left(d_{D'(j)} - \sum_{k=i}^{j} w_{D'(k)}\right) \qquad \text{Eq 27}$$

In either case, a set of importance functions for a procrastination policy is:

$$\forall (i \in T) \qquad I_i(t) = \begin{cases} (d_i - t)^{-1} + p, & \text{if } (i \in T_d \wedge s_i \le t < d_i) \\ p_i, & \text{if } (i \in T_p \wedge a_i \le t) \\ 0, & \text{otherwise} \end{cases} \qquad \text{Eq 28}$$

3.5. Meeting Critical Deadlines for Heterogeneous Task Sets with Arbitrary Sizes

We can combine the conditions from the importance functions of Eq 19 and Eq 30 to form a set of importance functions that provide guaranteed service to the most critical deadline-driven tasks while minimizing the average response time for tasks that are priority-driven. Assuming the restrictions on the request times for the task set as given in Eq 28, the importance functions are:

$$\forall (i \in T) \qquad I_i(t) = \begin{cases} (d_i - t)^{-1}, \text{ if } (i \in T_d) \\ \qquad \wedge \ (s_i \le t < d_i) \\ \qquad \wedge \ InMostCrit(i, t) \\ p_i, \text{ if } (i \in T_p \wedge a_i \le t) \\ 0, \text{ otherwise} \end{cases} \qquad \text{Eq 29}$$

In the nearest deadline first policy processor idle time occurs only after the deadlines of all of the active tasks are met. With a heterogeneous task set, the idle time is used to service the priority-driven tasks. When the deadline-driven tasks are deferred until the last possible moment, the priority-driven tasks are serviced sooner, thus moving the idle time in between the servicing of tasks from T_p and tasks from T_d. The final variation on the nearest deadline first policy presented here observes that, although there may be no benefit from servicing deadline-driven tasks earlier than later, there is no benefit from waiting to serve them while idle time exists. We construct a set of importance functions that implement a policy that (1) meets the deadlines for tasks in T_d, (2) prunes the least critical deadline-driven tasks when necessary, (3) reduces the response time for tasks in T_p, and (4) eliminates processor idle time if any task is active.

Define the function $Active{:}\{T\} \to Boolean$ to take a task set and return the value one if the set has any tasks which have arrived but for whom service is not completed, and return value zero otherwise. The set of importance functions is:

$$\forall (i \in T) \qquad I_i(t) = \begin{cases} (d_i - t)^{-1}, \text{ if } (i \in T_d) \\ \qquad \wedge \ (s_i \le t < d_i) \\ \qquad \wedge \ InMostCrit(i, t) \\ p_i, \text{ if } (i \in T_d \wedge a_i \le t < s_i \wedge \neg Active(T_p)) \\ p_i, \text{ if } (i \in T_p \wedge a_i \le t) \\ 0, \text{ otherwise} \end{cases} \qquad \text{Eq 30}$$

Schedules produced using these importance functions will service deadline-driven tasks in criticality order during what would have been idle time until either some priority-driven task becomes active or the current time equals the latest possible start time for this task.

Since servicing tasks from T_d during the idle time will affect the latest possible start time, the term s_i can be made into a continuous function $s_i(t)$:

$$s_i(t) = \min_{D(i) \leq j \leq |T_d|} \left(d_{D'(j)} - \sum_{k=i}^{j} w_{D'(k)} \Big|_t^{\infty} \right) \qquad \text{Eq 31}$$

Replacing s_i with $s_i(t)$ in Eq 32 will constantly update the latest possible start time. As this time is made later, the priority-driven tasks are given longer service times before being preempted for the deadline-driven tasks. This further reduces the average response time for tasks in T_p.

4. Conclusions

The importance abstraction has been shown to provide a general framework for expressing and analyzing scheduling policies. In particular, traditional scheduling algorithms such as nearest deadline first can easily be expressed, and their resulting schedules emulated, within this framework. Since the framework places emphasis on what makes an individual task important to the system, complex scheduling policies are also easily expressed. Here we have taken the NDF scheduling algorithm and proven some widely held results; more interestingly, however, we deviate from the pure NDF and include for consideration additional attributes. We are able to consider these variations on the NDF algorithm by changing the importance functions without changing the framework.

5. References

[BERN71] Bernstein, A. J. and Sharp, J. C., “A Policy-Driven Scheduler for a Time-Sharing System,” *Communications of the ACM*, Vol. 14, No. 2, pp. 74-78 (February 1971).

[BIYA88] Biyabani, S. R., Stankovic, J. A. and Ramamritham, K., “The Integration of Deadline and Criticalness in Hard Real-Time Scheduling,” *Proceedings of the 1988 IEEE Real-Time Systems Symposium*, Huntsville, Alabama, pp. 152-160 (December 6-8, 1988).

[JENS85] Jensen, E. D., Locke, C. D. and Tokuda, H., “A Time- Driven Scheduling Model for Real-Time Operating Systems,” *Proceedings of the Real-Time Systems Symposium*, pp. 112-122 (December 3-6, 1985).

[LIU73] Liu, C. L. and Layland, J. W., “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of the ACM*, Vol. 20, No. 1, pp. 46-61 (January 1973).

[RUSC77] Ruschitzka, M. and Fabry, R. S., “A Unifying Approach to Scheduling,” *Communications of the ACM*, Vol. 20, No. 7, pp. 469-477 (July 1977).

[SHA90] Sha, L. and Goodenough, J. B., “Real-Time Scheduling Theory and Ada,” *IEEE Computer*, Vol. 23, No. 4, pp. 53-62 (April 1990).

[STRA92] Strayer, W. T., “Function-Driven Scheduling: A General Framework for Expression and Analysis of Scheduling,” Dissertation, University of Virginia, Department of Computer Science, May 1992.