A Schedulable Utilization Bound for Aperiodic Tasks

Tarek F. Abdelzaher
Department of Computer Science
University of Virginia
Charlottesville, VA 22903

ABSTRACT

In this paper, we derive a utilization bound on schedulability of *apriodic* tasks with arbitrary arrival times, execution times, and deadlines. To the author's knowledge, this is the first time a utilization bound is derived for the aperiodic task model. It allows constructing an $O(1)$ admission test for aperiodic tasks. Earlier admission tests are at best $O(n)$. We show that deadline-monotonic scheduling is the optimal *fixed-priority* scheduling policy for aperiodic tasks in the sense of maximizing the schedulable utilization bound. We prove that the optimal bound is 5/8. Our result is an extension of the well-known Liu and Layland's bound of $ln\ 2$ (derived for periodic tasks). The new bound is shown to be tight. We briefly generalize our results to tasks with multiple resource requirements and multiple processors. Dynamic priority scheduling (EDF) of aperiodic tasks is shown to have the same schedulability bound as for periodic tasks.

Our findings are especially useful for an emerging category of soft real-time applications, such as online trading and e-commerce, where task (request) arrival times are arbitrary, task service times are unknown, and service has to be performed within a given deadline. Our result provides theoretical grounds for guaranteeing deadlines of individual aperiodic requests by observing only the aggregate utilization conditions which simplifies achieving real-time assurances in such applications.

1 Introduction

Research on real-time scheduling has traditionally classified tasks into periodic, sporadic, and aperiodic. A fundamental problem in real-time scheduling is that of computing the schedulability of a task set. For periodic and sporadic tasks, many schedulability conditions exist that relate schedulability to aggregate utilization. No such result exists for aperiodic tasks. In this paper, we prove for the first time that aperiodic tasks with arbitrary arrival times, computation times and deadlines are schedulable if their aggregate utilization does not exceed 5/8.

Aperiodic tasks (with unknown arrival times) are handled in prior literature in one of two ways. The first approach requires creation of a high-priority periodic server task for servicing aperiodic requests. Examples include the sporadic server [33], the deferrable server [40], and their variations [21]. The approach bounds the total load imposed on the system by aperiodic tasks allowing critical periodic tasks to meet their deadlines. It usually assumes that aperiodic tasks are soft, and attempts to improve their responsiveness rather that guarantee their deadlines. The second approach typically relies on algorithms for joint scheduling of both hard periodic and aperiodic tasks. It uses a polynomial acceptance test upon the arrival of each aperiodic task to determine whether or not it can meet its deadline. Examples include, aperiodic response-time minimization [20], slack maximization [11], slack stealing [41], the reservation-based (RB) algorithm [8], and the guarantee routines introduced most notably by the Spring kernel project [37]. In addition to being of higher complexity than utilization-based tasks, such admission control algorithms must know the worst case execution times of arrived tasks. This precludes their application to many soft real-time systems running on general-purpose platforms where worst-case execution times are too pessimistic or difficult to predict. To date, no utilization-based schedulability test has been proposed for aperiodic tasks. In this paper, such a test is derived. The complexity of the test is $O(1)$. An added advantage of utilization-based admission control is that, if soft guarantees are sufficient, *average* system utilization can be used for schedulability testing. This obviates knowledge of individual execution times of incoming tasks. As long as the average utilization is below the bound, and if each individual task contributes only a small fraction to utilization, new aperiodic tasks can be admitted. Utilization-based admission control is especially attractive to large scale servers in which individual requests consume only a small fraction of server capacity, but their exact execution times are unknown.

The need for utilization-based admission control is further magnified with the recent increase in soft real-time applications such as multimedia and online trading, where guarantees are required but individual task execution times are unknown. Modern mainstream operating systems already do task accounting that requires computing utilization. Thus, no significant operating system changes are needed to implement utilization-based admission control. Quality of service adaptation capabilities (such as image resolution control, video frame rate manipulation, color depth control, and relaxation of response time requirements) inherent to many modern applications allow controlling the utilization of a task set. If the utilization climbs, quality of service can be adapted to reduce utilization to the desired value. In a previous paper, we explored techniques borrowed from automatic feedback control theory to address the problem of stabilizing utilization around a given value by adapting quality of service. Analytic results can be applied from classical feedback control to guarantee convergence of utilization to the desired value within a specified time thus bounding the duration of transient overload. If utilization is related to schedulability, the approach can be used to keep the task set schedulable (i.e., to maintain its aggregate utilization below the schedulability bound). The main hurdle in applying this technique is the lack of a theoretical understanding of what this desired utilization bound should be in order for the task set to be schedulable. In this paper, this bound is derived for the case of aperiodic tasks with arbitrary arrival times, execution times and deadlines.

The rest of the paper is organized as follows. Section 2 describes related work. The main contribution of the paper is presented in Section 3 which derives the optimal utilization bound for aperiodic fixed-priority task scheduling. Section 4 derives the corresponding bound for dynamic-priority scheduling. Section 5 describes simple extensions to tasks with pipelined stages of different resource requirements. Finally, Section 6 presents the conclusions of the paper and avenues for future work.

2 Related Work

To date, three main paradigms have been proposed for real-time scheduling. Perhaps the earliest approach to providing guarantees in performance-critical systems has been to rely on *static* allocation and scheduling algorithms that assume full *a priori* knowledge of the resource requirements of tasks and their arrival times [4, 32, 43, 44]. Rate monotonic scheduling theory [31] introduced a second paradigm in which knowledge of task arrival times is not required. As a result, sporadic tasks could be accommodated as long as their minimum inter-arrival time is known. The concept of *dynamic* real-time systems [35], pioneered by the Spring kernel project [36, 37], introduced the third major paradigm to describe applications where run-time workload parameters are unknown until admission control time. It resulted in innovative planning-based scheduling algorithms that provide online guarantees for dynamically arriving tasks [17, 25, 28, 34, 38, 46, 47]. Task execution times where assumed to be known, e.g., using pre-run-time code analysis techniques such as [14, 39, 45].

With the advent of a new category of soft real-time applications such as multimedia, real-time databases, and e-commerce, the concept of QoS adaptation was introduced into resource allocation and scheduling. Typically, the approach assumes that the application can tolerate multiple levels of service which vary in their quality and resource requirements. Given the requirements of different QoS levels, an adaptation mechanism can determine the right QoS level depending on load conditions. Such QoS-adaptive service models were presented in [2, 3, 9, 15, 16]. Resource allocation mechanisms were developed to take advantage of adaptation. For example, the Q-RAM architecture [27] introduces QoS-sensitive near-optimal resource allocation algorithms for applications with multiple resource requirements and multiple QoS dimensions. FARA [29, 30] presents a hierarchical adaptation model for complex real-time systems and algorithms for optimizing multi-dimensional adaptation cost. An end-to-end QoS model is presented in [16] in the context of a middleware approach to QoS management that requires application cooperation. The approach is extended in [6] to account for practical limitations such as inaccuracies in estimating application resource requirements. In [13] a dynamic distillation method is proposed to adapt to network and client variability via on-line compression techniques. In the multimedia community several systems were described with adaptive QoS as well [3, 7, 10, 12, 18, 19, 24, 26, 42]. A good survey of such architectures can be found in [5]. While these approaches are more flexible in that they allow adaptation, they still share in common with their predecessors the need to know the resource requirements of tasks.

We envision a fourth paradigm for real-time scheduling that concerns aperiodic tasks (such as requests on a web server) whose execution times (or more generally, resource requirements) are unknown. The uncertainty in resource requirements may be due, for example, to data-dependencies that make it impossible to predict the execution time of a task without interpreting the semantics of its application-specific inputs. One measurable quantity in such systems would be the aggregate utilization of the different resources. Theory is needed to relate such utilization to the schedulability of aperiodic tasks. With the plethora of QoS adaptation mechanisms described in earlier literature, feedback-based QoS-adaptation can be used to maintain the utilization within schedulable limits. In earlier work [1, 23], control-theoretical feedback-based mechanisms were introduced for QoS adaptation that can maintain a desired average utilization. In this paper, we investigate the problem of deriving a utilization bound for guaranteed schedulability. Future work of the author is concerned with developing probabilistic guarantees on maintaining the utilization below the bound using control-theoretical techniques.

3 The Generalized Bound for Fixed Priority Scheduling

Consider the simple model of scheduling independent aperiodic tasks on a uniprocessor. These tasks may represent web requests, database transactions, online trades, or others. The service time of each task is generally unknown. For example, it may depend on application data carried in the request, whose application-specific semantics may not be understood by the operating system.

Task arrival times and deadlines are arbitrary. Let the arrival time of task T_i be denoted A_i, its execution time (possibly unknown to the OS) be denoted C_i, and its desired maximum response time be denoted D_i. The task meets its deadline if it finishes before $A_i + D_i$. In the rest of the paper we call $A_i + D_i$ the *absolute* deadline of the task and D_i its *relative* deadline. The average processor utilization U_i contributed by this task is $U_i = C_i/D_i$ in the interval between its arrival time and deadline.

At any given instant, t, let $S(t)$ be the set of all tasks that have arrived but whose deadline has not expired, i.e., $S(t) = \{T_i | A_i \leq t < A_i + D_i\}$. We call them the *current tasks*. Let $n(t)$ be the number of current tasks at time t. The utilization contributed by these tasks, called the *current utilization*, is $U(t) = \sum_{T_i \in S(t)} C_i/D_i$. In this paper, we prove that using an optimal *fixed* priority scheduling policy, all tasks will meet their deadlines if $\forall t : U(t) \leq \frac{5}{8} + \frac{1}{8(n(t)-1)}$. When the number of current tasks, $n(t)$, increases, the bound approaches 5/8. We also show that *dynamic* priority scheduling (EDF) achieves a schedulable utilization bound of unity, as is the case with periodic tasks.

3.1 Optimality and Fixed Priority Scheduling of Aperiodic Tasks

The difference between fixed priority scheduling and dynamic priority scheduling is somewhat muddled in the case of aperiodic tasks. Since each task has exactly one invocation, the notion of maintaining the same priority across multiple invocations (as is the case with fixed-priority scheduling) is inapplicable. Thus, in the context of aperiodic tasks, we consider a scheduling algorithm to be fixed priority if it (i) classifies all tasks into a *finite* number of classes, and (ii) associates a *fixed* priority with each class. More formally, from a mathematical standpoint, we define fixed-priority scheduling as follows:

Definition: *A fixed-priority scheduling algorithm for aperiodic tasks is a function $f(\tau) \rightarrow P$, that:*
- *maps an infinite set of task invocations τ into a finite set of values P, and*
- *satisfies $f(\tau)|_{time=t} = f(\tau)|_{time=t+x}$*

This, for example, is akin to diff-serv architectures which classify all network traffic into a finite number of classes and give some classes priority over others. EDF and FIFO by this definition are not fixed-priority scheduling policies since they relate the "priority" of an aperiodic task to absolute time (which is not a finite set), i.e., $f(\tau) \rightarrow t$, where t is interpreted as the set of absolute deadlines in case of EDF, and absolute arrival times in case of FIFO. By the same token, any policy in which task priorities are a function of task arrival times is not a fixed-priority policy. Note that, such a function will either be monotonic or non-monotonic. In the former case, an infinite number of priorities may exist, thus violating both parts of the above definition.[1]. In the latter case (e.g.,

[1]This should be understood in a mathematical sense. In reality, if the number of concurrent tasks at any given

$f(\tau) = \lceil A_i \rceil mod\ 10)$, the relative priority of two tasks arriving at times t_0 and $t_0 + x$ may depend on absolute time t_0, violating the second property of the definition.

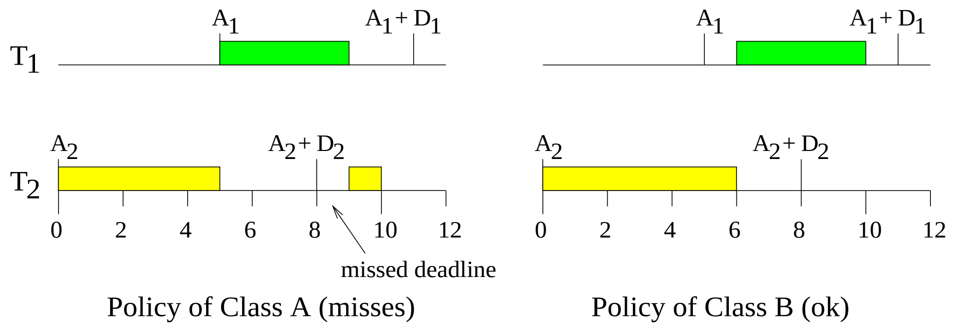
One fixed-priority classification of tasks would be by their relative deadlines, D_i. These deadlines are typically derived from a finite number of environmental constraints and determine the maximum response time within which the computing system must serve the task. It is therefore reasonable to assume that the number of distinct relative deadlines can generally be made bounded by design. For example, clients of an e-commerce server can be categorized into classes such as premium, basic, and economy, each with a different relative deadline. Aperiodic requests from clients will be served in accordance with these deadlines. We call a scheduling policy that assigns higher priority to aperiodic tasks with shorter relative deadlines, an *aperiodic deadline monotonic scheduling policy*. Unlike EDF, this policy is easily implementable on current mainstream operating systems which support fixed-priority scheduling.

The traditional sense of optimality of a fixed-priority scheduling policy is that for any set of task invocations, if the task set is schedulable by some fixed-priority policy it is schedulable by the optimal policy. In the case of aperiodic tasks, no fixed priority scheduling policy is optimal in the aforementioned sense. This lack of optimality is attributed to a semantic difference in the meaning of "schedulability". For periodic and sporadic task sets, schedulability generally refers to meeting all deadlines of the given task set *regardless* of task arrival times. Schedulability is guaranteed by considering a worst case task arrival pattern. The fact that an unschedulable task set may actually meet all deadlines for some specific task arrival patterns is irrelevant. This independence of the notion of schedulability from task arrival times makes it possible to compare prioritization policies that do not take task arrival times into account (which is true, by definition, of all fixed-priority scheduling policies). As a result, an optimal fixed-priority scheduling policy exists.

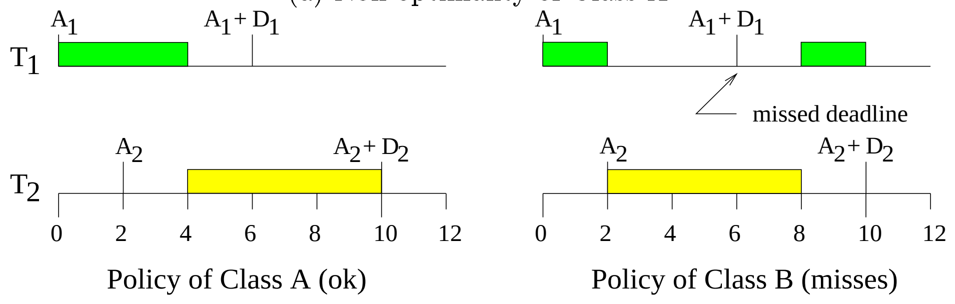
In aperiodic tasks, on the other hand, there is no notion of a fixed task set. Instead, we deal with dynamically arriving task invocations. Schedulability is meaningful only for the particular invocation arrival pattern that has occurred, which is in sharp contradiction to the case of periodic and sporadic tasks in which schedulability is analyzed for a worst-case arrival pattern. Since fixed-priority scheduling, by definition, is independent of task arrival times, there will always be a way to pick those arrival times (without affecting task priorities) such that an invocation set becomes unschedulable under a policy of choice while schedulable under another policy. This implies that no fixed-priority policy is optimal.

To prove that a policy is not optimal, it is enough to show a single invocation set that is not schedulable by this policy while schedulable by another. Thus, to prove that no fixed-priority policy is optimal, we consider a specific example of a task set composed of two tasks, T_1 and T_2,

time is bounded, dynamic scheduling policies such as EDF can be implemented using a finite number of priority levels of execution times 2 and 3, and relative deadlines 3 and 4, respectively. A fixed-priority policy can prioritize these tasks in only one of two ways, (a) $T_1 > T_2$ (class A policy), or (ii) $T_2 > T_1$ (class B policy). Figure 1-a shows that no class A policy is optimal because an arrival pattern exists that makes the set unschedulable under class A while schedulable under class B. Similarly, Figure 1-b shows that no class B policy is optimal because an arrival pattern exists that makes the set unschedulable under class B while schedulable under class A. Thus, no fixed-priority policy is optimal. An optimal policy, by necessity, would have to be a function of task arrival times (such as EDF).



(a) Non-optimality of Class A



(b) Non-optimality of Class B

Figure 1: Schedulability of Aperiodic Tasks

In the context of aperiodic fixed-priority task scheduling, we therefore suggest a different sense of optimality of a scheduling policy. We call a policy optimal if it maximizes the schedulable utilization bound. A task set is schedulable by the scheduling policy if the utilization bound is not exceeded. Note that in order to guarantee schedulability of task sets that do not exceed the bound, this bound must be computed for a worst case task arrival pattern. Hence, different fixed-priority policies can be compared in the worst case, and an optimal policy can be established. In essence, the bound classifies all possible invocation arrival patterns into those that meet the bound (guaranteed to be schedulable) and those that do not (which may be unschedulable). A policy with a higher bound will guarantee more task arrival patterns to be schedulable. The optimal policy will guarantee schedulability for the largest set of arrival patterns. In the next section such a policy and bound are derived.

3.2 The Optimal Utilization Bound

In this section, we derive the utilization bound for schedulability of aperiodic tasks under an optimal fixed-priority scheduling policy, and show that aperiodic deadline monotonic scheduling achieves the optimal bound. To provide more intuition into the construction of this derivation, we shall first consider the simple special case of two current tasks.

Theorem 1: *A set of aperiodic tasks in which at most two are current at any given time is schedulable using an optimal fixed-priority scheduling policy if* $\forall t : U(t) \leq \frac{3}{4}$, *where U(t) is the current utilization at time* t.

Proof: Let the term *critically schedulable task pattern* denote a task pattern in which adding an arbitrarily small amount of execution time to some task(s) causes a task to miss its deadline. Note that the top priority task will not miss its deadline unless its computation time is larger than its relative deadline. Thus, to compute a utilization bound we consider the schedulability of the lower priority task, say T_2. In the rest of this derivation we shall analyze the interval $[A_2, A_2 + D_2]$ during which T_2 is current. In this interval, T_2 may be preempted by a chain of higher priority tasks. No two tasks in that chain may be current at the same time, since this would bring the total number of current tasks to 3. Let us call the chain *sparse* if there is a gap between the deadline of some high priority task in the chain and the arrival time of the next. Otherwise, the chain is called *packed* (in which case the deadline of each task in the chain is the arrival time of the next). Let the last task in the chain that preempts T_2 be called T_1. Let, $U(t_{hi})$ be the maximum current task utilization in the interval $A_2 \leq t < A_2 + D_2$, i.e., $U(t_{hi}) = max_{A_2 \leq t < A_2 + D_2} U(t)$. We shall search all the critically schedulable task patterns for one that minimizes $U(t_{hi})$.

First, we shall prove that the minimum value of $U(t_{hi})$ is achieved when the utilization $U(t)$ is constant for all t in $A_2 \leq t < A_2 + D_2$. This statement is proved by contradiction. Assume that in the critically schedulable task pattern that minimizes $U(t_{hi})$, the utilization $U(t)$ for some $t = t_{lo}$ was lower than $U(t_{hi})$. In such a case, one can always find another critically schedulable task pattern with lower $U(t_{hi})$ as follows:

- **Case 1, Sparse Chain:** Consider the case where the chain of higher priority tasks that preempt T_2 is sparse, i.e., there is a gap between the deadline of one such task and the arrival time of another. Let the beginning and end of this gap be denoted by t_0 and t_1 respectively. Obviously, the maximum current utilization $U(t_{hi})$ cannot occur inside the gap. We can reduce $U(t_{hi})$ by reducing the execution time of the high priority task that is current at t_{hi} by an arbitrarily small amount δ, and creating a new high priority task with an arrival time t_0, deadline t_1 and execution time δ. The transformation does not change the total time that T_2 is preempted. Thus, T_2 remains critically schedulable. The resulting critically schedulable task pattern has a lower maximum utilization since the execution time of a task that contributes to $U(t_{hi})$ was reduced. An example of this transformation is shown in Figure 2-a.

- **Case 2, Packed Chain:** Consider the case where the chain of higher priority tasks is packed. Thus, the number of current tasks is exactly 2 everywhere in the interval $A_2 \leq t < A_2 + D_2$. Since $U(t)$ is not the same everywhere in this interval, it must be that at least two high priority tasks are present that differ in their C_i/D_i. Let t_{lo} be a time instant at which $U(t_{lo}) < U(t_{hi})$. We can reduce $U(t_{hi})$ by reducing the execution time of the high priority task that is current at t_{hi} by an arbitrarily small amount δ, and adding δ to the execution time of the high priority task that is current at t_{lo}. An example of this transformation is shown in Figure 2-b. As before, the transformation does not change the total time that T_2 is preempted. T_2 remains critically schedulable. The resulting critically schedulable task pattern has a lower maximum utilization since the execution time of a task that contributes to $U(t_{hi})$ was reduced.

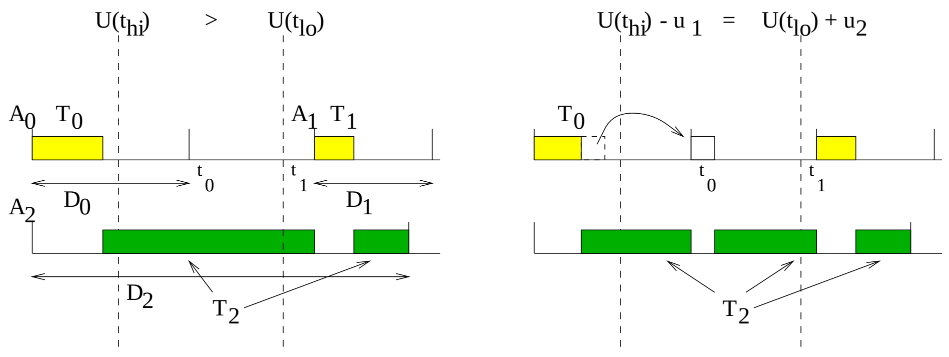
From case 1, we conclude that the maximum utilization $U(t_{hi})$ of a sparse chain can always be reduced. From case 2, we conclude that the maximum utilization of a packed chain can always be reduced unless $U(t)$ is constant. It follows that:

Property 1: The minimum lower bound on utilization that makes task T_2 critically schedulable occurs for a task set in which the high priority task chain is packed in the interval where T_2 is current, $A_2 \leq t < A_2 + D_2$.

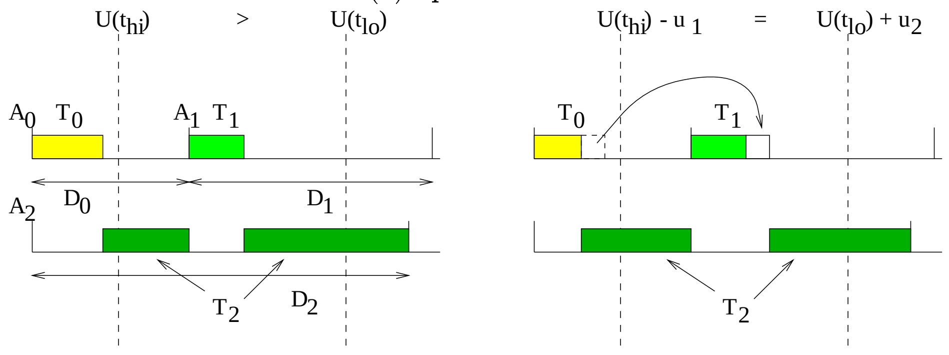
Property 2: The minimum lower bound on utilization that makes task T_2 critically schedulable occurs for a task set where $U(t)$ remains constant in the interval where T_2 in current, $A_2 \leq t < A_2 + D_2$.

Let the constant utilization referred to in Property 2 be denoted by U. It is desired to find the minimum U over all possible task patterns subject to the conditions of Property 1 and Property 2. Consider a task in the high priority chain that preempts T_2. Since all tasks in the chain have the same utilization, we arbitrarily choose the last task, called T_1. Let that task arrive T time units after the arrival time of T_2, i.e., $A_1 - A_2 = T$. Because it is the last task, its absolute deadline is larger than that of T_2, i.e., $A_1 + D_1 \geq A_2 + D_2$, or equivalently, $T + D_1 \geq D_2$. The utilization U must be minimized with respect to the three attributes of task T_1, namely, its execution time C_1, its relative deadline D_1, and its arrival time, T, relative to that of T_2. Thus, the derivation of the least upper bound undergoes three steps:

- **Step 1:** Minimize U with respect to the execution time C_1.



(a) Sparse Chain



(b) Packed Chain

Figure 2: Reducing the Maximum Utilization

- **Step 2:** Minimize U with respect to the deadline D_1

- **Step 3:** Minimize U with respect to T.

The result is a globally minimum expression for U. These steps are described below.

Step 1, minimizing U w.r.t. C_1: We prove that U is minimized when $C_1 = A_2 + D_2 - A_1$ (or equivalently $C_1 = D_2 - T$). This result is proved by considering the following two cases:

- **Case 1:** $C_1 \geq D_2 - T$. This case is depicted in Figure 3-a (left). The utilization is $U = C_1/D_1 + C_2/D_2$ and is constant in the interval $A_2 \leq t < A_2 + D_2$. Note that while we do not know how many high priority tasks may have preceded T_1 (i.e., executed in the interval T shown in Figure 3-a), we know that their utilization must have been equal to $U_1 = C_1/D_1$ for Property 2 to hold. In the worst case, the first of these tasks arrives together with T_2. Their combined execution time that preempts T_2 is therefore TC_1/D_1. For T_2 to be critically schedulable, it must execute for a duration $C_2 = T - TC_1/D_1$ (note that it cannot execute after the arrival time of T_1). Substituting for C_2 in U, we get:

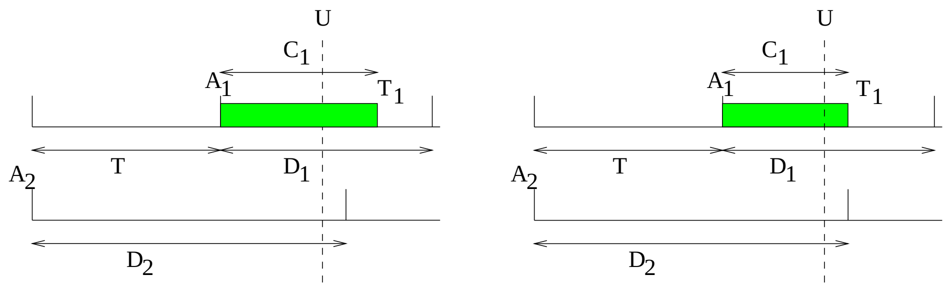
$$U = \frac{T}{D_2} + \frac{C_1}{D_1}(1 - \frac{T}{D_2}) \qquad (3.1)$$

From Equation (3.1), since $T < D_2$, the quantity $(1 - \frac{T}{D_2})$ is positive. Thus, U is minimum when C_1 is minimum, i.e., when $C_1 = D_2 - T$. This is shown in Figure 3-a (right).

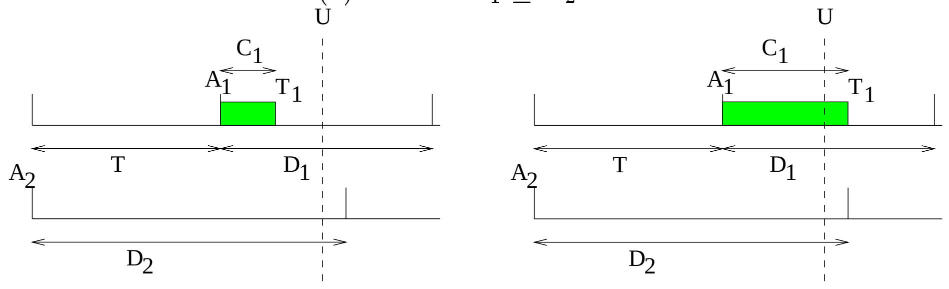
- **Case 2:** $C_1 < D_2 - T$. This case is depicted in Figure 3-b (left). In this case, for task T_2 to be critically schedulable, $C_2 = D_2 - C_1 - TC_1/D_1$. Substituting for C_2 in U, we get:

$$U = 1 - \frac{C_1}{D_1}(\frac{D_1 + T}{D_2} - 1) \qquad (3.2)$$

Note that by definition of T_1, its absolution deadline is after that of T_2. Thus, $\frac{D_1+T}{D_2} > 1$, and $(\frac{D_1+T}{D_2} - 1)$ is positive. It follows that U is minimized when C_1 is maximum, i.e., in this case, when $C_1 = D_2 - T$. This is depicted in Figure 3-b (right).



(a) Case 1: $C_1 \geq D_2 - T$



(b) Case 2: $C_1 < D_2 - T$

Figure 3: Computing C_1 for Minimum U

From Case 1 and Case 2 above, U is minimized (with respect to C_1) when $C_1 = D_2 - T$.[2] Note that for a given task set, one can always choose task arrival times such that the condition $C_1 = D_2 - T$ is satisfied. Since fixed-priority scheduling is independent of task arrival times, no matter how task priorities are computed, an adversary can always choose task arrival times to satisfy the minimum

[2]Liu and Layland [22] proved a similar result for the special case of periodic tasks.

utilization condition. Thus, this condition applies to all fixed-priority scheduling policies including the optimal policy.

Step 2, minimizing U w.r.t. D_1: From Equation (3.1) or Equation (3.2) which can be shown to be equivalent when $C_1 = D_2 - T$, it is easy to see that U decreases when D_1 increases. In particular, if $D_1 = \infty$, the utilization bound is T/D_2, which reduces to 0 when T is 0. Thus, it is possible to miss deadlines even at an arbitrary small utilization. To obtain a meaningful bound, the deadline D_1 must be upper-bounded. Note that it is impossible to achieve an upper bound on D_1 that is lower than D_2 since T_1 and T_2 may have the same relative deadline. Thus, the *minimum* upper bound achievable on D_1 is D_2. It corresponds to the best achievable utilization bound. Incidentally, deadline monotonic scheduling sorts tasks in the order of their relative deadlines such that deadlines of higher priority tasks are upper-bounded by those of lower priority tasks. Substituting with $D_1 = D_2 = T + C_1$ in either Equation (3.1) or Equation (3.2), the best utilization bound becomes:

$$U = 1 - \frac{TC_1}{(T + C_1)^2} \tag{3.3}$$

Step 3, minimizing U w.r.t. T: We minimize U with respect to T by setting $dU/dT = 0$. Note that one can choose task arrival times (without affecting task priorities under fixed-priority scheduling) to produce an arbitrary T regardless of the scheduling policy. Setting the derivative of Equation (3.3) to zero results in the condition $T = C_1$. Therefore, from Equation (3.3), $U = 0.75$. An example critical task set with a constant 0.75 utilization is shown in Figure 4. Theorem 1 is thus proved. □

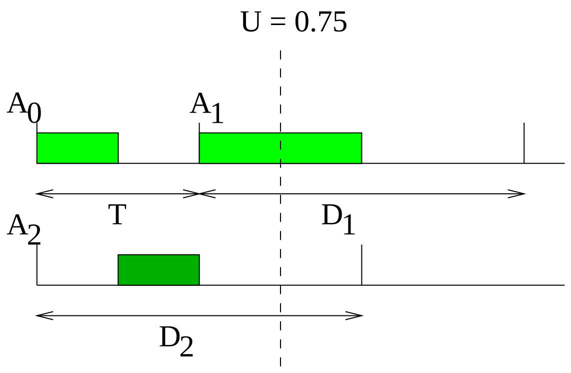


Figure 4: A Minimum Utilization Task Set

Note that the derivation of the optimal bound suggests the optimal fixed-priority scheduling policy. We have shown that achieving the optimal bound requires only that the deadline of the higher priority task be upper-bounded by that of the lower priority task. This is a property of aperiodic deadline monotonic scheduling. Thus, this policy is optimal. Next, we generalize Theorem 1 to an arbitrary number of current tasks.

Theorem 2: *A set of aperiodic tasks is schedulable using an optimal fixed-priority scheduling policy if* $\forall t : U(t) \leq \frac{5}{8} + \frac{1}{8(n-1)}$, *where* n *is the maximum number of current tasks in the system, and* $U(t)$ *is the current utilization at time* t.

Proof: To prove the theorem, we shall first extend Property 1 and Property 2 to the case of an arbitrary number of current tasks, then follow Step 1, Step 2, and Step 3 of the proof of Theorem 1. Let us consider a critically schedulable task pattern. By definition, some task in this pattern must have zero slack. Let us call this task T_n. Consider the interval of time $A_n \leq t < A_n + D_n$ during which T_n is current. At any time t within that interval, $U(t) = C_n/D_n + \sum_{T_i > T_n} C_i/D_i + \sum_{T_i < T_n} C_i/D_i$, where C_n/D_n is the utilization of task T_n, $\sum_{T_i > T_n} C_i/D_i$ is the utilization of higher priority tasks that are current at time t, and $\sum_{T_i < T_n} C_i/D_i$ is the utilization of lower priority tasks that are current at time t. Since lower priority tasks do not affect the schedulability of T_n, $U(t)$ is minimized when $\sum_{T_i < T_n} C_i/D_i = 0$. In other words, one can always reduce the utilization of a critically schedulable task pattern (in which task T_n has zero slack) by removing all tasks of priority lower than T_n. Thus, in the remainder of this proof, to arrive at a minimum utilization bound, T_n must be the lowest priority task. This is an intuitive result, since one would expect it to be "easier" to cause the *lowest* priority task miss its deadline.

As before, let us define a chain of tasks as a task sequence in which no two tasks are current at the same time. A chain is sparse if it contains a gap between the deadline of one task and the arrival time of another. Since at any given time in the interval $A_n \leq t < A_n + D_n$ the number of current tasks is at most n, we can string all tasks of higher priority than T_n into $n-1$ chains. (Task T_n constitutes the nth chain.) To make the treatment of the problem easier, we logically regard gaps in sparse chains as tasks of an infinitesimally small execution time. Thus, we can claim that all chains are packed. Let $U(t_{hi}) = \max_{A_n \leq t \leq A_n + D_n} U(t)$. If $U(t)$ is not the same everywhere in the interval $A_n \leq t < A_n + D_n$, it must be that at least two high priority tasks are present in some chain j that differ in their C_i/D_i, such that the task with the higher utilization is current at t_{hi}. In this case, we can reduce $U(t_{hi})$ by reducing the execution time of the high priority task which is current at t_{hi} in chain j by an arbitrarily small amount δ, and adding δ to the execution time of a task with a lower utilization in the same chain. The transformation does not change the total time that T_2 is preempted. Thus, T_2 remains critically schedulable. The resulting task pattern has a lower maximum utilization because the execution time of a task that contributes to $U(t_{hi})$ has been reduced. We have shown above that $U(t_{hi})$ can be reduced whenever $U(t)$ is not constant. Thus:

Property 3: The minimum lower bound on utilization that makes T_n critically schedulable occurs when the utilization $U(t)$ remains constant in the interval where T_n is current, $A_n \leq t < A_n + D_n$.

We now proceed with minimizing the utilization U with respect to the attributes of higher priority tasks, namely, their execution times C_i, their deadlines D_i, and their relative arrival times.

The minimization undergoes three steps:

Step 1, minimizing U w.r.t. C_i: For each higher priority chain i, $1 \leq i \leq n-1$, consider the task T_i that arrives last within the interval $A_n \leq t < A_n + D_n$. Let the utilization of this task be $U_i = C_i/D_i$. Since T_i is the last task, its deadline is outside this interval (note that C_i may be zero if T_i is a gap). The lowest priority task T_n is preempted by all tasks T_1, ..., T_{n-1}, and all tasks that precede them in their chains since the arrival time of T_n. Let the sum of execution times of all tasks that precede T_i in chain i, and preempt T_n, be C_{P_i}. Since the utilization $U(t)$ is the same in the interval $A_n \leq t < A_n + D_n$, it must be that $\sum_{1 \leq i \leq n-1} C_{P_i} = \sum_{1 \leq i \leq n-1} (A_i - A_n) C_i/D_i$. In the worst case, task T_n arrives simultaneously with the first task in each chain, and is therefore preempted by the entire $\sum_{1 \leq i \leq n-1} C_{P_i}$. Since T_n has no slack, its execution time is then given by:

$$C_n = D_n - \sum_{1 \leq i \leq n-1} (A_i - A_n) C_i/D_i - \sum_{1 \leq i \leq n-1} (C_i - overflow_i) \tag{3.4}$$

Where $overflow_i$ is the amount of computation time of task T_i that occurs after the deadline of task T_n. Let $overflow = \sum_{1 \leq i \leq n-1} overflow_i$. Substituting for C_n in $U = \sum_{1 \leq i \leq n} C_i/D_i$, the utilization is given by:

$$U = 1 + (1 - \frac{1}{D_n}) \sum_{1 \leq i \leq n-1} C_i/D_i - \frac{1}{D_n} \sum_{1 \leq i \leq n-1} (A_i - A_n) C_i/D_i + \frac{overflow}{D_n} \tag{3.5}$$

Among tasks $T_1, ..., T_{n-1}$, where task T_k is the last task in chain k, let the latest task completion time be E_{last}. Let S_k be the start time of T_k. To minimize U with respect to the computation times of these tasks, we shall inspect the derivative dU/dA_k. Three cases arise:

1. *T_k arrives while a task of higher priority is running:* In this case, T_k is blocked upon arrival. Advancing the arrival time A_k by an arbitrarily small amount does not change its start time (and therefore does not change the start or finish time of any other task). Consequently, $overflow$ remains constant, and $doverflow/dA_k = 0$. Thus, from Equation (3.5), $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. This quantity is negative indicating that U can be decreased by increasing the arrival time A_k.

2. *T_k arrives while a task of lower priority is running:* In this case T_k preempts the executing task upon arrival. Advancing the arrival time A_k by an arbitrarily small amount reorders execution fragments of the two tasks without changing their combined completion time. Consequently, $overflow$ remains constant, and $doverflow/dA_k = 0$. Thus, from Equation (3.5), $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. This quantity is negative indicating that U can be decreased by increasing the arrival time A_k.

3. *T_k arrives while no task is running:* In other words, it arrives at or after the completion time of the previously running task. Let us define a *busy period* as a period of contiguous CPU execution of tasks $T_1, ..., T_{n-1}$. The execution of these tasks forms one or more such busy periods. Two cases arise:

   - A. T_k is not in the busy period that ends at E_{last}: Advancing A_k will not change $overflow$. Thus, $doverflow/dA_k = 0$, and $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. This quantity is negative indicating that U can be decreased by increasing the arrival time A_k.
   - B. T_k is in the busy period that ends at E_{last}. Three cases arise:
     - (I) $E_{last} > D_n$: In this case, $overflow > 0$. Since no other task was running when T_k arrived, advancing the arrival time of T_k will shift the last busy period and increase $overflow$ by the same amount. It follows that $doverflow/dA_k = 1$. Thus, from Equation (3.5), $dU/dA_k = \frac{1}{D_n}(1 - \frac{C_i}{D_i})$. This quantity is positive indicating that U can be decreased by decreasing A_k.
     - (II) $E_{last} < D_n$: In this case, $overflow = 0$. $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. This quantity is negative indicating that U can be decreased by increasing the arrival time A_k.
     - (III) $E_{last} = D_n$: From (I) and (II) above, it can be seen that $lim_{E_{last} \to D_n^+} dU/dA_k \neq lim_{E_{last} \to D_n^-} dU/dA_k$. Thus, the derivative dU/dA_k is not defined at $E_{last} = D_n$. From the signs of the derivative in (I) and (II), it can be seen that U has a minimum at $E_{last} = D_n$.

From the above, U can be decreased in all cases except case 3.B.(III) where a minimum occurs. Since the above cases exhaust all possibilities and 3.B.(III) is the only minimum, it must be a global minimum. In this case, each task T_k arrives while no task is running (by definition of case 3) and contributes to a contiguous busy period that ends at E_{last} (by definition of subcase B) where $E_{last} = D_n$ (by definition of subcase III). In other words, each task arrives exactly at the completion time of the previous task (for the busy period to be contiguous), with the completion of the last task being T_n. For simplicity, let us re-number tasks $T_1, ..., T_{n-1}$ in order of their arrival times. It follows that U is minimized when $C_i = A_{i+1} - A_i$, $1 \leq i \leq n-2$ and $C_{n-1} = D_n - A_{n-1}$ as depicted in Figure 5. Since fixed-priority scheduling is independent of task arrival-times, no fixed-priority policy can prevent the aforementioned sequence of arrivals from occurring. Regardless of how tasks are prioritized, an adversary can always choose their arrival times to satisfy the minimum utilization condition derived above.

Let $T = A_1 - A_n$. Since $overflow = 0$ at the global minimum, from Equation (3.4), $C_n = D_n - \sum_{1 \leq i \leq n-1} \{C_i + (A_i - A_n) C_i / D_i\}$, where $A_i - A_n = T + \sum_{1 \leq j \leq i-1} C_j$. Substituting for $A_i - A_n$, we get the expression:

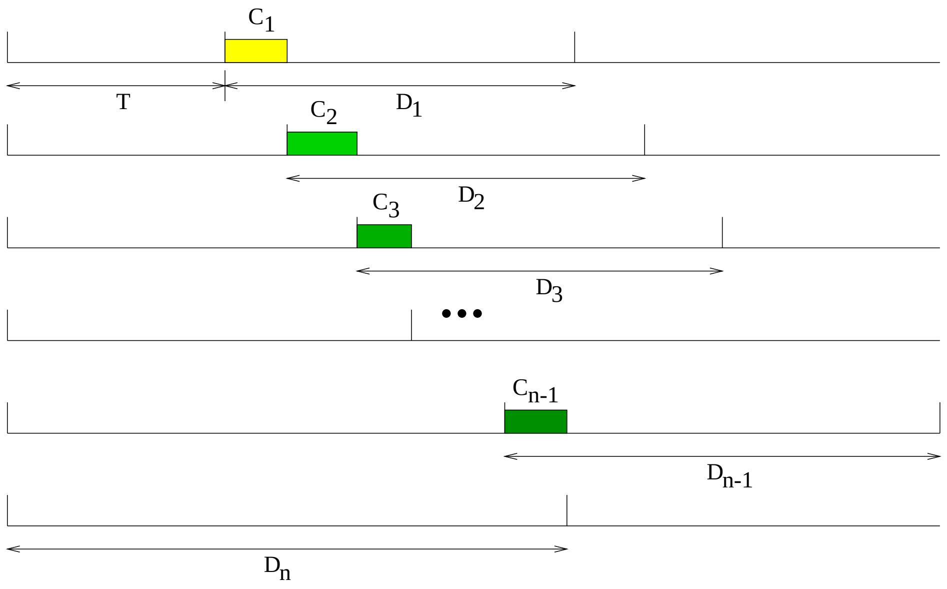


Figure 5: Schedulability of Aperiodic Tasks

$$C_n = T(1 - \sum_{1 \le i \le n-1} C_i/D_i) - \sum_{1 \le i \le n-2} (C_i \sum_{i+1 \le j \le n-1} C_j/D_j) \tag{3.6}$$

Substituting in the utilization expression $U(t) = \sum_i C_i/D_i$, we get:

$$U = T/D_n + (1 - T/D_n) \sum_{1 \le i \le n-1} C_i/D_i + \sum_{1 \le i \le n-2} (C_i/D_n \sum_{i+1 \le j \le n-1} C_j/D_j) \tag{3.7}$$

Step 2, minimizing U w.r.t. D_i: The utilization in Equation (3.7) decreases when D_1, ..., D_{n-1} increase. To achieve a meaningful utilization bound, these deadlines must be upper-bounded by the scheduling policy. The minimum achievable upper bound on these deadlines is D_n since it is possible for all relative deadlines to be equal. Subject to this observation, in the best scheduling policy the schedulable utilization bound is minimum when $D_1 = D_2 = ... = D_{n-1} = D_n$. Equation (3.7) can be significantly simplified in this case. The resulting utilization is given by:

$$U = 1 - \frac{T \sum_{1 \le i \le n-1} C_i + \sum_{1 \le i \le n-2} (C_i \sum_{i+1 \le j \le n-1} C_j)}{(T + \sum_{1 \le i \le n-1} C_i)^2} \tag{3.8}$$

Step 3, minimizing U w.r.t. T: Since arrival times of tasks $T_1, ..., T_{n-1}$ are spaced by the respective task computation times, as found in Step 1, to obtain the condition for minimum utilization, it is enough to minimize U with respect to T. We set $dU/dT = 0$. Setting the derivative of Equation (3.8) to zero, we get $T = \sum_{1 \le i \le n-1} C_i$. Note that the value of T depends on task arrival times. Thus, regardless of the used fixed-priority scheduling policy, an adversary can choose task arrival times (without affecting task priorities) to produce T that satisfies the above minimum schedulable utilization condition. Substituting with $T = \sum_{1 \le i \le n-1} C_i$ in Equation (3.8) and simplifying, we get:

$$U = \frac{5}{8} + \frac{1}{8} \frac{\sum_{1 \le i \le n-1} C_i^2}{(\sum_{1 \le i \le n-1} C_i)^2} \tag{3.9}$$

The quantity $\frac{\sum_{1 \le i \le n-1} C_i^2}{(\sum_{1 \le i \le n-1} C_i)^2}$ in Equation (3.9) is lower bounded by $1/(n-1)$ which corresponds to the case where task computation times C_1, ..., C_{n-1}, are equal. Consequently, the optimal lower bound on the utilization of a critically schedulable aperiodic task set is:

$$U = \frac{5}{8} + \frac{1}{8(n-1)} \tag{3.10}$$

The theorem is thus proved. $\square$

The bound presented in Theorem 2 is tight. From the construction of the proof it can be seen that the bound is achieved if task T_n is preempted by $n-1$ packed chains of higher priority tasks. Each chain may consist of two tasks of equal utilization, the first of which arrives together with T_n. The second task in each chain j, $1 \le j \le n-1$, has an arrival time $A_j = (j-1)T/(n-1)$, a computation time $C = T/(n-1)$, and a relative deadline $D = 2T$. Note that this implies that the first task in chain j has a relative deadline $T + (j-1)C$ and a computation time $\{T + (j-1)C\}C/D$. Task T_n, which is critically schedulable, has a computation time $T - \sum_{1 \le j \le n-1} \{T + (j-1)C\}C/D$ and a deadline of $2T$. This worst case task pattern is shown in Figure 6.

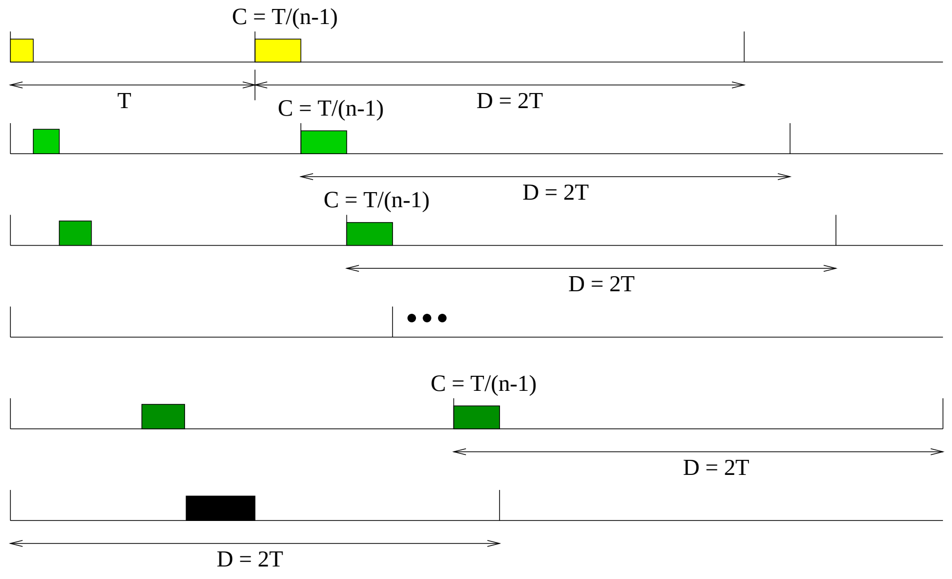


Figure 6: The Worst Cast Task Pattern

In the above proof, in order to obtain the optimal utilization bound we had to upper-bound the relative deadlines of higher priority tasks. In particular, we made use of the assumption that the relative deadlines of the $n-1$ higher priority tasks can be upper-bounded at best by that of the lowest priority task. We observed that this bound is tight. No policy can reduce it (since all tasks may have the same relative deadline). We also observed that aperiodic deadline monotonic scheduling ensures that relative deadlines of higher priority tasks are upper bounded by those of lower priority tasks. Thus, we arrive at the following corollary:

Corollary: *Aperiodic deadline monotonic scheduling is an optimal fixed priority scheduling policy in the sense of maximizing the schedulable utilization bound.*

Theorem 2 and its corollary establish a tight utilization bound on the schedulability of aperiodic tasks and an optimal fixed-priority aperiodic task scheduling policy. Tasks in different contexts can mean different things, such as service requests, packets, transactions, etc. We addressed fixed-priority scheduling because of its particular importance as it is widely supported by current operating systems, unlike more esoteric real-time scheduling algorithms.

The derived bound makes possible an $O(1)$ schedulability test based on current utilization. The test increments current utilization by C_i/D_i of the arrived aperiodic task. The task is admited if the new current utilization does not exceed the bound. Otherwise, utilization is rolled back to its previous value and the task is rejected. Upon task deadlines, current utilization is decremented by C_i/D_i of the task whose deadline expired. The derived utilization bound may be of interest in several contexts. For example, in the context of packet scheduling on network routers, the bound states that current communication link utilization due to real-time traffic should be less than 5/8 for all real-time packets to meet their deadlines (the remaining utilization can be allotted to best effort traffic). Other applications that may benefit from this bound could be web servers with guaranteed response time, databases with real-time transactions, and in general, adaptive real-time applications operating in aperiodic environments.

The bound can be used in an average sense. Instead of maintaining a running estimate of current utilization, one may rely on estimates of *average* utilization measured over finite time intervals. The accuracy of such an approximation needs to be studied and probabilistically guaranteed, which is an interesting avenue for future research. The approximation eliminates the need to know task computation times as long as the total utilization is known. It may be of great value to applications such as large servers in which computation times of individual requests are unknown, and each request consumes only a small fraction of the total capacity. Controlling only the aggregate utilization to maintain schedulability allows per-task guarantees to be attained by mechanisms that require only aggregate knowledge. The approach clearly has great scalability benefits.

4 The Bound for Dynamic Priority Scheduling

For completeness, in this section, we consider dynamic-priority scheduling. It is easy to show that EDF is the optimal dynamic priority scheduling algorithm for aperiodic tasks and that the optimal utilization bound for dynamic priority scheduling is 1. To see that, consider a hypothetical processor capable of generalized processor sharing. The processor assigns each task T_i (of execution time C_i and deadline D_i) a processor share equal to C_i/D_i. All current tasks execute concurrently. Each task will terminate exactly at its deadline. The maximum schedulable utilization of this processor is trivially $U = 1$. One can imagine that such a processor, in effect, schedules current tasks in a round-robin fashion, assigning each an infinitesimally small time slice that is proportional to its share, C_i/D_i. We shall show that this round-robin schedule can be converted into an EDF schedule without causing any task to miss its deadline.

Let us sweep the round-robin schedule from its beginning to its end. At every time t, let us choose among all current tasks the one with the earliest deadline, say T_j, and consolidate its round-robin slices scheduled in the interval $[A_j, t]$. Consolidation is effected by shifting these slices towards the task's arrival time, displacing slices of tasks that have not been consolidated. Since we are merely switching the order of execution of task slices within an interval that ends at t, no slice is displaced beyond t. Since task T_j was current at time t, it must be that $t \leq A_j + D_j$ (the absolute deadline of T_j). Thus, no slice is displaced beyond T_js deadline. However, since T_j is the task with the shortest deadline among all current tasks at time t, no slice is displaced beyond its task's deadline. The resulting schedule after the sweep is EDF. Thus, EDF has the same utilization bound as generalized processor sharing. In other words, it will cause no aperiodic deadline misses as long utilization is less than 1. This result is similar to the case of sporadic tasks.

5 Extensions to Multiple Resources

It is straightforward to extend aperiodic task schedulability analysis to a special case of multiple resource requirements in which execution on different resources is pipelined. Consider a task model in which each task T_j is represented by a vector of stages in which each stage i has a resource requirement C_j^i for resource r_i, and a relative deadline D_j^i. The task arrives at time A_j, with an end-to-end relative deadline $D_j = \sum_i D_j^i$. Such a task set is schedulable if each individual resource is schedulable. If the utilization of each resource remains below the bound defined by Theorem 2, individual task stages will meet their deadlines. Thus, the end-to-end relative deadline of each task will be met. This simple model approximates applications such as web and e-commerce servers. In a web server, processing of a user's request proceeds on multiple stages each of which requires a different resource. First, CPU is needed for protocol processing and user authentication. Next, disk bandwidth is needed to read the requested file (web page) by the disk controller. Then, CPU is needed again to prepare the file for transmission. Finally, link bandwidth is needed to transmit the file by the network adaptor. All deadlines will be met if the current utilization of each resource (CPU, disk bandwidth, and communication link) is kept below the schedulable bound.

More complicated schedulability analysis is needed in the case where tasks require several resources *together*. Critical sections, for example, belong to this category of models, since they imply that tasks require *both* the CPU and some additional serial resources at the same time. Schedulability analysis in the presence of critical sections has been studied in the case of rate monotonic scheduling. In essence, the utilization bound is decreased by the worst case blocking delay. We believe that the approach will apply to the aperiodic task case as well. However, schedulability analysis in the presence of critical sections is beyond the scope of this paper. Other interesting extensions left for future work include those to multiprocessor scheduling, non-preepmtable tasks, intertask communication, and general precedence and mutual exclusion constraints.

6 Conclusions and Future Work

In this paper, we derived, for the first time, the optimal utilization bound for the schedulability of aperiodic tasks under fixed-priority scheduling. The bound allows an $O(1)$ admission test of incoming tasks, which is faster than the polynomial tests proposed in earlier literature. We also showed that aperiodic deadline monotonic scheduling is an optimal policy in the sense of maximizing the schedulable utilization bound. This result may be the first step towards an aperiodic deadline monotonic scheduling theory — an analog of rate monotonic scheduling theory for the case aperiodic tasks. Such a theory may prove to be of significant importance to many real-time applications such as real-time database transactions, online trading servers, and guaranteed-delay packet scheduling. In such applications aperiodic arrivals have deadline requirements and their schedulability must be maintained.

While we limited this paper to the first fundamental result, our investigation is by no means complete. We will explore in subsequent publications extensions of the theory to the case of dependent tasks, multiple resource requirements, precedence and exclusion constraints, non-preemptive execution, and other task dependencies in a multi-resource environment. We shall also extend our results to multiprocessor scheduling of aperiodic tasks. While a multiprocessor can be trivially considered as a set of uniprocessors, it is interesting to investigate whether or not better bounds are possible when all processors share a single run queue.

Finally, to make the results more usable, it is important to investigate methods for aggregate utilization control that would maintain the utilization below the schedulability bound. Statistical properties of the task arrival process can be combined with mathematical analysis of feedback control loops to derive probabilistic guarantees on meeting task deadlines. This avenue is currently being pursued by the author.

Acknowledgements

The author would like to thank Jack Stankovic a discussion with whom sparked the idea for this research. The author would also like to thank Kang Shin, Sang Son, Gang Tao and Chenyang Lu for their useful advice.

References

[1] T. Abdelzaher and K. G. Shin. Qos provisioning with $q$contracts in web and multimedia servers. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[2] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin. QoS negotiation in real-time systems and its application to automated flight control. In *IEEE Real-Time Technology and Applications Symposium*, Montreal, Canada, June 1997.

[3] T. F. Abdelzaher and K. G. Shin. End-host architecture for qos-adaptive communication. In *IEEE Real-Time Technology and Applications Symposium*, Denver, Colorado, June 1998.

[4] T. F. Abdelzaher and K. G. Shin. Combined task and message scheduling in distributed real-time systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(11), November 1999.

[5] C. Aurrecoechea, A. Cambell, and L. Hauw. A survey of QoS architectures. In *4th IFIP International Conference on Quality of Service*, Paris, France, March 1996.

[6] S. Brandt and G. Nutt. A dynamic quality of service middleware agent for mediating application resource usage. In *Real-Time Systems Symposium*, pages 307–317, Madrid, Spain, December 1998.

[7] A. Cambell, G. Coulson, and D. Hutchison. A quality of service architecture. *ACM Computer Communications Review*, April 1994.

[8] Y.-C. Chang and K. G. Shin. A reservation-based algorithm for scheduling both periodic and aperiodic real-time tasks. *IEEE Transactions on Computers*, 44(12):1405–1419, December 1995.

[9] S. Chatterjee, J. Sydir, B. Sabata, and T. Lawrence. Modeling applications for adaptive qos-based resource management. In *Proceedings of the 2nd IEEE High-Assurance System Engineering Workshop*, Bethesda, Maryland, August 1997.

[10] D. Chen, R. Colwell, H. Gelman, P. K. Chrysanthis, and D. Mosse. A framework for experimenting with QoS for multimedia services. In *International Conference on Multimedia Computing and Networking*, 1996.

[11] R. Davis and A. Burns. Optimal priority assignment for aperiodic tasks with firm deadlines in fixed priority pre-emptive systems. *Information Processing Letters*, 53(5):249–254, March 1995.

[12] B. Field, T. Znati, and D. Mosse. V-net: A framework for a versatile network architecture to support real-time communication performance guarantees. In *InfoComm*, 1995.

[13] A. Fox, S. Gribble, E. A. Brewer, and E. Amir. Adapting to network and client variability via on-demand dynamic distillation. In *Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, Massachusetts, October 1996.

[14] M. G. Harmon, T. P. Baker, and D. B. Whalley. A retargetable technique for predicting execution time of code segments. *Journal of Real-Time Systems*, 7(2):159–182, September 1994.

[15] D. Hull, A. Shankar, K. Nahrstedt, and J. W. S. Liu. An end-to-end qos model and management architecture. In *Proceedings of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, pages 82–89, San Francisco, California, December 1997.

[16] M. Humphrey, S. Brandt, G. Nutt, and T. Berk. The DQM architecure: middleware for application-centered qos resource management. In *Proceedings of IEEE Workshop on Middleware for Distributed Real-time Systems and Services*, San Francisco, California, December 1997.

[17] G. Koren and D. Shasha. D-over: An optimal on-line scheduling algorithm for overloaded real-time systems. In *IEEE Real-Time Systems Symposium*, pages 290–299, Phoenix, Arizona, December 1992.

[18] L. Krishnamurthy. *AQUA: An Adaptive Quality of Service Architecture for Distributed Multimedia Applications.* PhD thesis, University of Kentucky, 1997.

[19] A. Lazar, S. Bhonsle, and K. Lim. A binding architecture for multimedia networks. *Journal of Parallel and Distributed Computing*, 30:204–216, Nevember 1995.

[20] J. Lehoczky and S. R. Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Real-Time Systems Symposium*, pages 110–123, Phoenix, AZ, December 1992.

[21] T.-H. Lin and W. Tarng. Scheduling periodic and aperiodic tasks in hard real-time computing systems. *Performance Evaluation Review*, 19(1), May 1991.

[22] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of ACM*, 20(1):46–61, 1973.

[23] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son. The design and evaluation of a feedback control edf scheduling algorithm. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[24] K. Nahrstedt and J. Smith. Design, imlementation, and experiences with the OMEGA end-point architecture. *IEEE JSAC*, September 1996.

[25] M. D. Natale and J. A. Stankovic. Dynamic end-to-end guarantees in distributed real-time systems. In *Proc. Real-Time Systems Symposium*, pages 216–227, December 1994.

[26] B. D. Noble and M. Satyanrayanan. Experience with adaptive mobile applications in odyssey. to appear in Mobile Networking and Applications.

[27] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. Practical solutions for qos-based resource allocation problems. In *Real-time Systems Symposium*, pages 296–306, Madrid, Spain, December 1998.

[28] K. Ramamritham, J. A. Stankovic, and W. Zhao. Distributed scheduling of computing tasks with deadlines and resource requirements. *IEEE Transactions on Computers*, 38(8):1110–1123, August 1989.

[29] D. Rosu and K. Schwan. Faracost: An adaptation cost model aware of pending constraints. In *IEEE Real-Time Systems Symposium*, Phoenix, Arizona, December 1999.

[30] D. Rosu, K. Schwan, and S. Yalamanchili. FARA - a framework for adaptive resource allocation in complex real-time systems. In *Real-time Technology and Applications Symposium*, pages 79–84, Denver, Colorado, June 1998.

[31] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate monotonic scheduling theory: A framework for developing real-time systems. *Proceedings of the IEEE*, 82(1):68–82, January 1994.

[32] T. Shepard and M. Gagne. A pre-run-time scheduling algorithm for hard real-time systems. *IEEE Transactions on Software Engineering*, 17(7):669–677, Jul 1991.

[33] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems*, 1(1):27–60, June 1989.

[34] J. Stankovic. Decentralized decision making for task reallocation in a hard real-time system. *IEEE Transactions on Computers*, 38(3):341–355, March 1989.

[35] J. Stankovic and K. Ramamritham. *Hard Real-time Systems*. IEEE Press, 1988.

[36] J. A. Stankovic and K. Ramamritham. The design of the Spring kernel. In *Proc. Real-Time Systems Symposium*, pages 146–157, December 1987.

[37] J. A. Stankovic and K. Ramamritham. The Spring Kernel: A new paradigm for real-time systems. *IEEE Software*, pages 62–72, May 1991.

[38] J. A. Stankovic, K. Ramamritham, and S. Cheng. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Transactions on Computers*, 34(12):1130–1143, December 1985.

[39] A. D. Stoyenko and T. J. Marlowe. Polynomial-time transformations and schedulability analysis of parallel real-time programs with restricted resource contention. *Journal of Real-Time Systems*, 4(4):307–329, December 1992.

[40] J. K. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.

[41] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Real-Time Systems Symposium*, pages 22–33, San Juan, Puerto Rico, December 1994.

[42] C. Volg, L. Wolf, R. Herrwich, and H. Wittig. HeiRAT – quality of service management for distibuted multimedia systems. *Multimedia Systems Journal*, 1996.

[43] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on Software Engineering*, 19(2):139–154, February 1993.

[44] J. Xu and D. L. Parnas. Scheduling processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Software Engineering*, SE-16(3):360–369, March 1990.

[45] S.-M. Yang, Y.-S. Kim, and M. H. Kim. An approach to dynamic execution time estimation for adaptive real-time task scheduling. *Journal of the Korea Information Science Society*, 23(1), January 1996.

[46] W. Zhao, K. Ramamritham, and J. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Transactions on Computers*, 36(8):949–960, August 1987.

[47] W. Zhao, K. Ramamritham, and J. Stankovic. Scheduling tasks with resource requirements in hard real-time systems. *IEEE Transactions on Software Engineering*, 13(5):564–577, May 1987.