

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 *aperiodic* tasks with arbitrary arrival times, execution times, and deadlines. Earlier utilization bounds considered only periodic and sporadic tasks. To the author's knowledge, this is the first time a utilization bound is derived for the aperiodic task model. 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. 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.

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 $5/8$. This result extends the Liu and Layland bound of $\ln 2$ derived for periodic tasks [3].

The need for utilization-based admission control of aperiodic tasks is magnified by the recent increase in soft real-time applications such as online trading, where requests arrive aperiodically and response-time

guarantees are required. In such applications individual task execution times are typically unknown. Each task consumes only a small amount of server capacity collectively imposing a certain total utilization on the server. 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 [2], 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. 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. For space limitations we refer the reader to an expanded version of this publication [1] for a survey of related work.

2 The Generalized Bound

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 $5/8$. In [1], we also show that *dynamic* priority scheduling (EDF) achieves a schedulable utilization bound of unity, as is the case with periodic tasks.

2.1 Fixed-Priority Scheduling

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. This is akin to differentiated services where all network traffic is classified into a finite number of classes of different priority.

One fixed-priority classification of aperiodic tasks would be by their relative deadlines, D_i . We call a policy that assigns higher priority to tasks with smaller relative deadlines *aperiodic deadline monotonic scheduling*. Relative 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.

We call a fixed-priority scheduling policy optimal if it maximizes the schedulable utilization bound. In essence, the bound classifies all possible aperiodic 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.

2.2 Derivation of the Bound

We now prove the following theorem.

Theorem 1: *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: 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, to arrive at a minimum utilization bound, T_n must be the lowest priority task.

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 n th chain.) Note that we do *not* assume that all tasks in a chain have the same priority. 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, 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, \dots, 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:

$$C_n = D_n - \sum_{i=1}^{n-1} (A_i - A_n) \frac{C_i}{D_i} - \sum_{i=1}^{n-1} (C_i - v_i) \quad (1)$$

Where v_i (which stands for *overflow*) is the amount of computation time of task T_i that occurs after the

deadline of task T_n . Let $v = \sum_{1 \leq i \leq n-1} v_i$. Substituting for C_n in $U = \sum_{1 \leq i \leq n} C_i/D_i$, the utilization is given by:

$$U = 1 + \left(1 - \frac{1}{D_n}\right) \sum_{i=1}^{n-1} \frac{C_i}{D_i} - \frac{1}{D_n} \sum_{i=1}^{n-1} (A_i - A_n) \frac{C_i}{D_i} + \frac{v}{D_n} \quad (2)$$

Among tasks T_1, \dots, 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, v remains constant, and $dv/dA_k = 0$. Thus, from Equation (2), $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. 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, v remains constant, and $dv/dA_k = 0$. Thus, from Equation (2), $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. 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, \dots, 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 v . Thus, $dv/dA_k = 0$, 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, $v > 0$. 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 v by the same amount. It follows

that $dv/dA_k = 1$. Thus, from Equation (2), $dU/dA_k = \frac{1}{D_n}(1 - \frac{C_i}{D_i})$. This quantity is positive indicating that U can be decreased by decreasing A_k . (II) $E_{last} < D_n$: In this case, $v = 0$. $dU/dA_k = -\frac{1}{D_n}(\frac{C_i}{D_i})$. 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 $\lim_{E_{last} \rightarrow D_n^+} dU/dA_k \neq \lim_{E_{last} \rightarrow D_n^-} dU/dA_k$. 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, \dots, 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 1. 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.

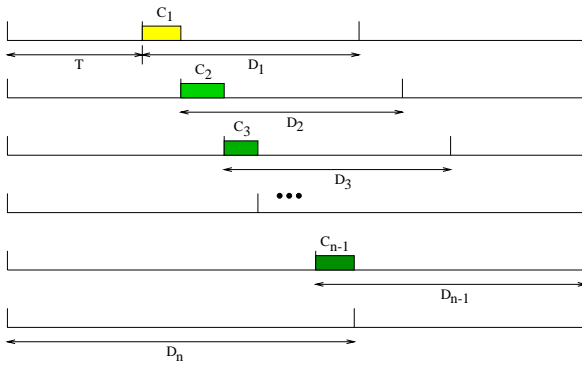


Figure 1: Schedulability of Aperiodic Tasks

Let $T = A_1 - A_n$. Since $v = 0$ at the global mini-

mum, from Equation (1), $C_n = D_n - \sum_{1 \leq i \leq n-1} \{C_i + (A_i - A_n)C_i/D_i\}$, 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:

$$C_n = T(1 - \sum_{i=1}^{n-1} \frac{C_i}{D_i}) - \sum_{i=1}^{n-2} (C_i \sum_{j=i+1}^{n-1} \frac{C_j}{D_j}) \quad (3)$$

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

$$U = \frac{T}{D_n} + (1 - \frac{T}{D_n}) \sum_{i=1}^{n-1} \frac{C_i}{D_i} + \sum_{i=1}^{n-2} (\frac{C_i}{D_n} \sum_{j=i+1}^{n-1} \frac{C_j}{D_j}) \quad (4)$$

Step 2, minimizing U w.r.t. D_i : The utilization in Equation (4) decreases when D_1, \dots, D_{n-1} increase. To achieve a meaningful utilization bound, these deadlines must be upper-bounded by the scheduling policy. It is easy to see that no scheduling policy can assign task priorities such that the upper bound on D_1, \dots, D_{n-1} is smaller than D_n . By contradiction, if such a policy existed, it would have been unable to schedule a task set where all deadlines are equal ($\forall i : D_i = D_n$). Hence, the minimum achievable upper bound on deadlines D_1, \dots, D_{n-1} (by any scheduling policy) is D_n . Subject to this observation, in the best scheduling policy the schedulable utilization bound is minimum when $D_1 = D_2 = \dots = D_{n-1} = D_n$. Equation (4) can be significantly simplified in this case. The resulting utilization is given by:

$$U = 1 - \frac{T \sum_{i=1}^{n-1} C_i + \sum_{i=1}^{n-2} (C_i \sum_{j=i+1}^{n-1} C_j)}{(T + \sum_{i=1}^{n-1} C_i)^2} \quad (5)$$

Step 3, minimizing U w.r.t. T : Since arrival times of tasks T_1, \dots, 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 (5) to zero, we get $T = \sum_{1 \leq i \leq n-1} C_i$. 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 $T = \sum_{1 \leq i \leq n-1} C_i$ in Equation (5) and simplifying, we get:

$$U = \frac{5}{8} + \frac{1}{8} \frac{\sum_{1 \leq i \leq n-1} C_i^2}{(\sum_{1 \leq i \leq n-1} C_i)^2} \quad (6)$$

The quantity $\frac{\sum_{1 \leq i \leq n-1} C_i^2}{(\sum_{1 \leq i \leq n-1} C_i)^2}$ in Equation (6) is lower bounded by $1/(n-1)$ which corresponds to the case where task computation times are equal. Consequently, the optimal lower bound on the utilization of a critically schedulable aperiodic task set is:

$$U = \frac{5}{8} + \frac{1}{8(n-1)} \quad (7)$$

The theorem is thus proved.

The presented bound 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 , $1 \leq j \leq n-1$, 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 $T - \sum_{1 \leq j \leq n-1} \{T + (j-1)C\}C/D$ and a deadline of $2T$. This worst case task pattern is shown in Figure 2.

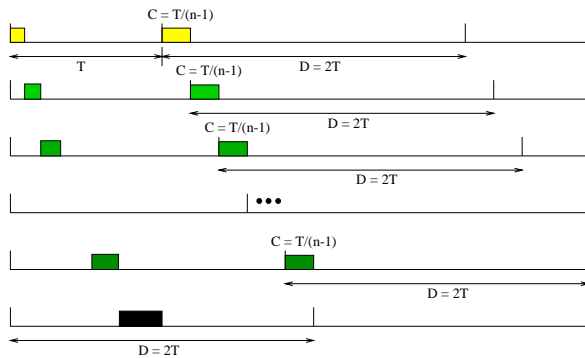


Figure 2: The Worst Cast Task Pattern

In the above proof, we showed that optimal utilization bound is achieved when the relative deadlines of the $n-1$ higher priority tasks can be upper-bounded by that of the lowest priority task. We observed that no policy can do better. We now observe 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.*

3 Conclusions

In this paper, we derived, for the first time, the optimal utilization bound for the schedulability of aperiodic tasks under fixed-priority scheduling. A more detailed proof and related work survey are presented in [1]. 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. Note that the bound may be pessimistic if the arrivals are extremely bursty, in the same sense that a bound derived for periodic tasks may be pessimistic when applied to sporadic tasks. An important future step is to refine the bound by using information on the burst. Other extensions include considering 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.

References

- [1] T. F. Abdelzaher. A schedulable utilization bound for aperiodic tasks. Technical Report CS-2000-21, University of Virginia, Charlottesville, VA, August 2000.
- [2] T. F. Abdelzaher and N. Bhatti. Web server QoS management by adaptive content delivery. In *International Workshop on Quality of Service*, London, UK, June 1999.
- [3] 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.