A Synthetic Utilization Bound for Aperiodic Tasks with Resource Requirements *

Tarek Abdelzaher, Vivek Sharma
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
e-mail:{*zaher, viveks*}*@cs.virginia.edu*

Abstract

Utilization bounds for schedulability of aperiodic tasks are new in real-time scheduling literature. All aperiodic bounds known to date apply only to independent tasks. They either assume a liquid task model (one with infinitely many infinitesimal tasks) or are limited to deadline-monotonic and earliest-deadline first scheduling. In this paper, the authors make two important contributions. First, they derive the first aperiodic utilization bound that considers a task model with resource requirements. Second, the new bound is a function of a parameter called preemptable deadline ratio *that depends on the scheduling policy. We show that many scheduling policies can be classified by this parameter allowing per-policy bounds to be derived. Simulation results demonstrating the applicability of aperiodic utilization bounds are presented.*

Keywords: Real-time scheduling, schedulability analysis, utilization bounds, aperiodic tasks.

1 Introduction

Utilization bounds are an efficient way to determine the schedulability of tasks with real-time constraints. Significant effort has gone into establishing a general theory for utilization-based admission control that extends the Liu and Layland utilization bounds for periodic tasks [11]. Most previous efforts in this area have confined themselves to variations of the periodic task model. In a significant departure from this model, the authors derived in [2] the first utilization bound for independent *aperiodic* tasks, i.e., tasks in which there are no constraints on arrival times, execution times or deadlines. The bound was derived for deadline-monotonic scheduling, which was shown to be an optimal fixed-priority policy.

A known criticism of using utilization bounds for schedulability analysis of periodic tasks is that they are generally pessimistic. For example, the bound for rate monotonic scheduling does not guarantee meeting task deadlines unless the CPU utilization is below 69%. In reality, task sets of much higher utilization can meet their deadlines. In [1] the authors have shown that admission control based on the utilization bound for aperiodic tasks does not underutilize the CPU. The reason is that the bound is defined on a utilization-like metric called *synthetic utilization*, which is numerically different from real utilization (where real utilization is defined as the percentage of time the CPU is busy processing admitted tasks). It was shown that the average real utilization of admitted tasks, in general, exceeds their synthetic utilization. Hence, while the synthetic utilization of those tasks may be kept around 58.6% (the aperiodic bound), the CPU could be more than 90% utilized.

This paper builds upon previous initial work on aperiodic utilization bounds by generalizing the aforementioned results in several important ways. First, we extend aperiodic utilization bounds to consider resource constraints. Second, we derive a parameterized bound that can be computed for arbitrary scheduling policies, as opposed to one specific to deadline monotonic scheduling or EDF. Moreover, we show how periodic tasks can be guaranteed together with aperiodics, and re-affirm by simulation the efficiency of utilization-based admission control in terms of not underutilizing the system even in the presence of resource constraints.

The work on aperiodic utilization bounds is motivated in part by the need to provide temporal guarantees in emerging real-time applications operating in unpredictable environments. These applications range from web hosting servers where aperiodic incoming service requests have deadlines but no periodicity constraints, to multifunction phase array radars where a collection of periodic and aperiodic real-time tracking, surveillance, and communication tasks must be accommodated on the same computing back-end. A constant-time utilization-based schedulability analysis and admission control test is sought for non-independent (i.e., resource constrained) real-time aperiodic tasks arriving dynamically possibly in the presence of *a priori* guaranteed periodic tasks.

The remainder of this paper is organized as follows. Section 2 describes the proposed task model. Section 3 details the proof of the generalized utilization bound. The perfor-

*The work reported in this paper was supported in part by the National Science Foundation under grants CCR-0093144, ANI-0105873, and EHS-0208769, and MURI N00014-01-1-0576.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 **IEEE**

mance of admission control based on the aforementioned bound is presented in Section 4. Section 5 presents related work. The paper concludes with Section 6 which summarizes the results and presents avenues for future work.

2 System Model

For the purposes of deriving the schedulability bound, we consider a system of purely aperiodic tasks. Observe that periodic arrivals can be thought of as a special case of aperiodic arrivals in which the arrival times happen to be equally spaced. Hence, a bound derived for aperiodic tasks guarantees that no task misses its deadline as long as the bound is not exceeded even when some tasks happen to arrive periodically.

We assume that the code of both periodic and aperiodic tasks admissible to the system is known, and hence the resource requirements of each task are known *a priori*, although the arrival times are not. We consider time-independent scheduling policies (i.e., ones where task priorities are independent of task arrival times). Thus, the priority of each task is also known in advance. Access to resources is protected by critical sections. The priority ceiling protocol is used at run-time to access critical sections. A pre-run-time analyzer determines the maximum blocking time of each task, which is the largest critical section of a lower priority task that can block the given task.

We denote each aperiodic task by T_i, which constitutes a single invocation specified by the tuple (A_i, C_i, B_i, D_i). In this tuple, $A_i \geq 0$ is the arrival time of the task, $C_i > 0$ is its computation requirement (this would be the worst case expected execution time in cases where the execution time is not known precisely), B_i is its maximum blocking time, and D_i is the relative deadline of the task, which is the time from its arrival to the point by which the task must finish execution. The absolute deadline for the task is $A_i + D_i$.

We define a *task arrival pattern* as a possibly infinite list of task arrivals, in which each task arrival T_i is defined by a tuple (A_i, C_i, B_i, D_i). The list represents the set of tasks that arrives to the system in a particular run. All arrivals are uniquely numbered. We assume that the system does not have knowledge of future arrivals. Hence, at any given time t, it is only aware of arrivals T_i for which $0 \leq A_i \leq t$.

A scheduling policy assigns a priority P_i to each task T_i. In time-independent scheduling, the priority P_i assigned to task T_i is the value returned by the function $f_P(i, C_i, B_i, D_i)$ which maps task parameters into one of a finite set of possible priority values. Note that the arrival time, A_i, is not included in the parameters of the priority function. Hence, scheduling policies such as EDF (where priority is proportional to $A_i + D_i$) are not time-independent. Task identity, i, is included in the parameter list of $f_P()$ to allow implementing prioritization policies where priority depends on arbitrary external factors such as the identity of the client who submitted the task (in a web server example) or the importance of the target that the task is tracking (in radar installations). The processor ready queue is sorted by priority. Tasks with the same priority are queued in FIFO order. Scheduling is preemptive and work-conserving.

To perform schedulability analysis and admission control, we consider a utilization-like quantity we call *synthetic utilization*[1]. Given a task arrival pattern, ζ, synthetic utilization at time t is defined as the sum of the utilizations (C_i/D_i) of the tasks in that pattern that are *current* at time t. Current tasks are the tasks that have arrived but whose deadlines have not expired yet. Let $V^\zeta(t)$ be the set of current tasks at time t in an arrival pattern ζ, i.e., $V^\zeta(t) = \{T_i | T_i \in \zeta, A_i \leq t < A_i + D_i\}$. Synthetic utilization, $U^\zeta(t)$ is defined as:

$$U^\zeta(t) = \sum_{T_i \in V^\zeta(t)} \frac{C_i}{D_i} \quad (1)$$

Henceforth, when we say utilization, we shall mean synthetic utilization unless explicitly mentioned otherwise. Observe that the synthetic utilization is a function of time. Given a particular task attival pattern, it is useful to define a *synthetic utilization curve* as the curve that plots the synthetic utilization versus time for the pattern under consideration.

3 The Generalized Aperiodic Bound

The problem of deriving a schedulability bound for an arbitrary time-independent scheduling policy $f_P()$ can be stated as one of finding a value $Bound_{f_P}$ for the synthetic utilization, such that for any task arrival pattern, ζ, if the synthetic utilization $U^\zeta(t)$ is kept below $Bound_{f_P}$ at all times t, all tasks in that pattern are guaranteed to meet their deadlines under scheduling policy $f_P()$. Task arrival patterns whose synthetic utilization exceeds the bound may or may not contain unschedulable tasks.

3.1 Preliminaries

Given an arbitrary time-independent scheduling policy $f_P()$, the derivation of the utilization bound $Bound_{f_P}$ amounts to the following steps:

1. Consider an arbitrary task arrival pattern ζ in which one or more tasks have zero or negative slack under scheduling policy $f_P()$. Pick the first such task in that arrival pattern. Let us denote it by T_n. The arrival time of this task is A_n and its absolute deadline is $A_n + D_n$.

[1] Also known as instantaneous utilization.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

No deadlines are missed in the time interval $[0, A_n + D_n)$.

2. For the task arrival pattern ζ, compute the maximum synthetic utilization U^{ζ}_{max}, which occurs in the busy period prior to the deadline of T_n. This can be seen in Figure 1. Let B be the beginning of that busy period. Then, $U^{\zeta}_{max} = \max_{B \le t < A_n + D_n} U^{\zeta}(t)$.

3. The utilization bound is given by:

$$Bound_{f_P} = \min_{\zeta} U^{\zeta}_{max} \quad (2)$$

The minimization in Equation (2) implies that any unschedulable task pattern, ζ, necessarily reaches or exceeds $Bound_{f_P}$ at some point in time (namely, when $U^{\zeta}(t) = U^{\zeta}_{max}$) prior to the first deadline miss. Thus, any task arrival pattern where synthetic utilization is always below $Bound_{f_P}$ must be schedulable.

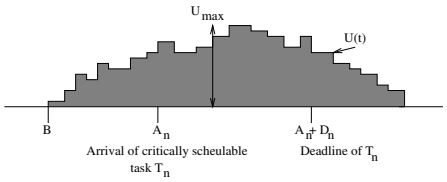


Figure 1. The maximum synthetic utilization for a critically schedulable task T_n

Unlike earlier literature on utilization bounds were the bound is typically derived for a specific scheduling policy (or for an optimal policy in its class), in this paper we seek a parameterized expression $Bound(\alpha)$ that is applicable to any time-independent policy. This expression is a function of some parameter α, whose value depends on the scheduling policy, $\alpha = g(f_P())$. Substituting in $Bound(\alpha)$ with α of a particular policy, the numeric utilization bound for that policy can be computed. We purposely postpone the definition of α until later in the paper to keep the following discussion independent of how α is defined. We call all time-independent scheduling policies that have the same value of α, a class-α scheduling family, F_α. A schedulability bound for a class-α scheduling family must satisfy:

$$Bound(\alpha) \le \min_{f_P \in F_\alpha} Bound_{f_P} \quad (3)$$

The inequality is because we are not aiming at a tight bound. We call any bound that satisfies the above inequality a *valid* bound.

The outline of the proof of the utilization bound consists of the following steps. First, we prove that in deriving $Bound(\alpha)$ we can consider, without loss of generality, only those policies in which priorities are independent of computation times (Lemma 1). Second, we prove that in the minimization given by inequality (3), we need to consider only the set of critically schedulable task patterns (Lemma 2). In Section 3.2, we prove three important properties of the critically schedulable task pattern that minimizes synthetic utilization (Lemmas 3-5). These properties narrow down the search to the extent that we can express it as a linear geometric optimization problem, which we solve in Section 3.3. This solution yields the sought bound.

We begin by showing that in deriving $Bound(\alpha)$ we can assume, without loss of generality, that task priorities are independent of task computation times. To see why this is true, let us make a distinction within any class-α scheduling family, F_α, between scheduling policies, F_{comp_α}, in which the priority assignment depends on task computation times, and scheduling policies, F_{arb_α}, where it does not. Let us define $Bound_{comp}(\alpha) = \min_{f_P \in F_{comp_\alpha}} Bound_{f_P}$, and $Bound_{arb}(\alpha) = \min_{f_P \in F_{arb_\alpha}} Bound_{f_P}$. Observe that $F_\alpha = F_{comp_\alpha} \cup F_{arb_\alpha}$. Hence:

$$Bound(\alpha) = \min\{Bound_{comp}(\alpha), Bound_{arb}(\alpha)\}. \quad (4)$$

Lemma 1: In any class-α scheduling family: $Bound_{comp}(\alpha) \ge Bound_{arb}(\alpha)$.

Proof: While the complete proof is omitted for space limitations, the intuition behind it is that (i) adding constraints on priority assignment (such as dependencies on computation times) can only reduce the set of all schedules that satisfy these constraints, and (ii) minimization of a quantity (such as the unilization bound) over a subset yields a higher (or equal) value compared to minimization over the entire set. Consequently, the bound for the family of scheduling policies where priority assignment depends on (i.e., is constrained by) execution times cannot be lower than the bound for the family of policies where no such dependencies exist. ■

To derive a valid synthetic utilization bound, $Bound(\alpha)$, it is therefore enough to consider only those policies where task priority does not depend on task execution time. It is useful to conceptually view the derivation of the utilization bound as a search through all possible unschedulable and critically schedulable task arrival patterns for one which minimizes the bound. To reduce the number of task sets we need to consider in our search, we define a *domination* relation on task arrival patterns. We say that a task pattern ζ_1 is *dominated by* another task pattern ζ_2 with respect to $U^{\zeta}(t)$ if and only if for all t, $U^{\zeta_1}(t) \ge U^{\zeta_2}(t)$, i.e., the former has the same or higher synthetic utilization compared to the latter at any point in time. With this definition in mind, given a scheduling policy $f_P()$, if task arrival pattern ζ_1 is dominated by task arrival pattern ζ_2, and if the latter pattern contains a task of zero or negative slack, then the former pattern need not be considered in the search for the utilization bound. We call this relation the *pruning rule*. To prove the validity of this rule, observe that since ζ_2 contains

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

a task of zero or negative slack, it is considered in the minimization in Equation (2). Since ζ_1 is dominated by ζ_2, the minimization applied to ζ_1 and ζ_2 can always safely choose ζ_2 over ζ_1. Hence, ζ_1 need not be considered.

The pruning rule will be applied extensively in the rest of the paper to narrow the search space for the utilization bound. We begin by using the pruning rule to show that all unschedulable task patterns may be removed from consideration as they are dominated by critically schedulable task patterns. This is stated in the following lemma.

Lemma 2: Given a priority assignment $f_P()$ that is independent of task computation times, any unschedulable task arrival pattern ζ_{un} is dominated by some critically schedulable pattern ζ_{cr}.

Proof: The proof of the lemma lies in the observation that given any unschedulable task arrival pattern, ζ_{un}, in which T_n is the first unschedulable task (under some priority assignment $f_P()$), we can generate a different arrival pattern, ζ_{cr}, by removing from ζ_{un} any tasks or parts thereof that execute after $A_n + D_n$. The schedule up to $A_n + D_n$ remains the same because the priorities of the remaining tasks are unaffected by this transformation (they are independent of computation times). Task T_n in the resulting ζ_{cr} is critically schedulable since we removed the part of its computation time after $A_n + D_n$. The critically schedulable task pattern ζ_{cr} satisfies $U^{\zeta_{un}}(t) \geq U^{\zeta_{cr}}(t)$ for all t, since the latter pattern was obtained from the former by removing tasks or reducing their computation times. Hence, ζ_{cr} dominates ζ_{un} and the lemma follows. ■

From Lemma 2, to compute a valid bound, it is enough to consider only critically schedulable task patterns. Thus, Equation (2) becomes:

$$Bound_{f_P} = \min_{\zeta_{cr}} U^{\zeta_{cr}}_{max} \tag{5}$$

where the minimization is carried out over all critically schedulable task patterns ζ_{cr}. Let us define the *worst case* critically schedulable task arrival pattern ζ_w as one containing a critically schedulable task T_n, for which $U^{\zeta_w}_{max} \leq U^{\zeta_{cr}}_{max}$ for all ζ_{cr}, which is the utilization bound. To derive the utilization bound it suffices to find one worst case pattern.

3.2 Properties of the Worst Case Pattern

Consider a critically-schedulable task pattern, ζ, in which T_n is the first critically-schedulable task. Tasks not in the busy period of T_n (i.e., the period of continuous CPU utilization that contains T_n) do not affect the schedulability of T_n and hence need not be considered. The synthetic utilization $U^{\zeta}(t)$ in the busy period of T_n can be expressed as:

$$U^{\zeta}(t) = \sum_{T_i|P_i \geq P_n} C_i/D_i + \sum_{T_i|P_i < P_n} C_i/D_i \tag{6}$$

where $\sum_{T_i|P_i \geq P_n} C_i/D_i$ is the utilization of tasks of priority P_n or higher, and $\sum_{T_i|P_i < P_n} C_i/D_i$ is the utilization of the lower priority tasks current at time t. Let us define $u^{\zeta}(t)$ as the synthetic utilization contributed by tasks of priority P_n or higher, i.e., $u^{\zeta}(t) = U^{\zeta}(t) - \sum_{T_i|P_i < P_n} C_i/D_i$. Note that $u^{\zeta}(t) \leq U^{\zeta}(t)$ for all t. Consider the bound $bound_{f_P} = \min_{\zeta_{cr}} u^{\zeta_{cr}}_{max}$ on the quantity $u^{\zeta}(t)$. Note that $bound_{f_P} \leq Bound_{f_P} = \min_{\zeta_{cr}} U^{\zeta_{cr}}_{max}$ because $u^{\zeta}(t) \leq U^{\zeta}(t)$. Hence, if a task arrival pattern ζ satisfies $U^{\zeta}(t) < bound_{f_P}$ for all t, it also satisfies $U^{\zeta}(t) < Bound_{f_P}$ for all t, and is therefore schedulable. In other words, $bound_{f_P}$ is a valid bound. Thus, in the rest of the paper, it is enough to derive $bound_{f_P}$. In other words, we subtract the contribution of lower priority tasks to synthetic utilization. To quantify this contribution, we first compute the total computation time attributed to tasks of priority lower than P_n in the busy period of a worst case pattern.

Lemma 3: To find a worst case pattern in which T_n is critically schedulable, it is sufficient to consider only those patterns where tasks of priority lower than P_n either do not exist or satisfy the following: (i) they block some task of priority P_n or higher, and (ii) they lock a semaphore of priority ceiling P_n or higher for the entire duration of their execution.

Proof: Consider an arbitrary critically schedulable task arrival pattern, ζ in which T_n is the first critically schedulable task. Let us remove from this pattern any tasks of priority lower than P_n which do not directly or transitively delay any tasks of priority P_n or higher. Observe that removal of such tasks does not affect the schedulability of T_n. Consider the busy period of T_n. Let us generate an arrival pattern ζ_2 from ζ in which each task of priority lower then P_n in ζ (which contains critical sections) is "cut up" at the boundaries of its critical sections into tasks that don't lock any resources and tasks that represent a critical section. Figure 2-a and Figure 2-b illustrate before and after snapshots of this transformation. The set of resulting tasks in ζ_2 has the same arrival time and relative deadline as the original task in ζ. Note that this transformation does not affect the schedule since task priorities are unaffected. Hence, ζ_2 is critically-schedulable. Observe that the transformation does not change synthetic utilization. Hence, $u^{\zeta_2}(t) = u^{\zeta}(t)$ for all t.

Since the scheduling policy is arrival time independent, we can now advance the arrival time of the tasks in ζ_2 representing critical sections to their start times without affecting their priority and hence without affecting the schedule. Let

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

the resulting arrival pattern be ζ_3. It is shown in Figure 2-c. Since this transformation moves low priority tasks only and since their contribution to synthetic utilization is not included in $u(t)$, we have $u^{\zeta_3}(t) = u^{\zeta_2}(t)$ for all t.

Finally, observe that in the resulting task arrival pattern, tasks of priority lower than P_n that do not contain critical sections do not affect the schedule of other tasks and hence can be removed. These tasks do not delay any higher priority tasks (since they don't lock any resources) and do not delay lower priority tasks with critical sections (since those latter tasks execute immediately upon arrival in ζ_3). The removal of these tasks from the arrival pattern yields an arrival pattern ζ_4, shown in Figure 2-d. The transformation may fragment the busy period into multiple sections interspersed by gaps. Only the section in which T_n executes is retained. Tasks comprising the other sections can be dropped without affecting the schedulability of T_n. It follows that $u^{\zeta_4}(t) \leq u^{\zeta_3}(t)$ for all t. By transitivity, $u^{\zeta_4}(t) \leq u^{\zeta}(t)$. Hence, the resulting arrival pattern dominates the original pattern, ζ. The resulting pattern is also critically schedulable since none of the transformations affected the schedulability of T_n. Hence, by the pruning rule, the original pattern need not be considered. Observe that the new pattern satisfies the properties mentioned in the lemma. The lemma is therefore proved. ■

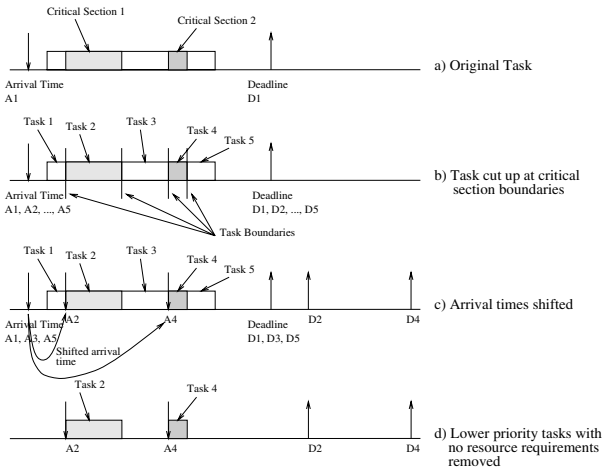


Figure 2. Transformations for Lemma 3

Note that, without loss of generality, we may assume that no tasks arrive at or after $A_n + D_n$ in the critically schedulable pattern, because such tasks only increase utilization without affecting the schedulability of T_n [1].

Lemma 4: The worst case pattern includes at most one blocking time from a lower priority task when a priority ceiling protocol is used.

Proof: Consider the busy period $[B, A_n + D_n)$ which satisfies Lemma 3 and in which the critically schedulable task T_n has priority P_n. Such a busy period consists only of:

- T_n and tasks of same or higher priority. Let us call them the high-priority task set.
- Tasks of priority lower than P_n which block a task in the high-priority task set. Let us call them the low priority task set.

By contradiction, assume that such blocking occurs twice or more in the aforementioned busy period. Consider the last time it occurs. Let T_L be the low priority task which causes the blocking. Since the low priority blocking task T_L contends for the CPU at its original priority (before the critical section is accessed), it must be that no high priority task was running at the time it acquired the CPU. Also, by Lemma 3, all lower priority tasks in the busy period enter their critical section immediately, i.e., they execute at the priority ceiling of the resource they lock, which is at least of priority P_n. Such tasks therefore cannot be preempted by the low priority task P_L. Hence, when P_L acquired the CPU, no lower priority task was running either. It follows that P_L is the first task in the busy period, which is a contradiction. ■

From Lemma 4, tasks of priority P_n or higher execute for at least $L + D_n - B^n_{max}$, in the busy period of T_n, where B^n_{max} is the length of the worst case critical section blocking a task of priority P_n or higher. In [1], we show that the sum of execution times in the busy period is equal to the area under the utilization curve. Thus, finding the utilization bound, $bound_{f_P}$, reduces to finding a geometric shape of area $L + D_n - B^n_{max}$ that has a minimum height. Ideally, the minimum height geometric shape for a fixed area and base is the rectangle. However, since no tasks arrive after $A_n + D_n$, the synthetic utilization must decrease after $A_n + D_n$ as tasks that were current at $A_n + D_n$ reach their respective deadlines. The height of the utilization curve, U, is therefore defined by the height of the flat rectangle in the interval that ends at $A_n + D_n$, as shown in Figure 3. Looking at the shape of the area in Figure 3, note that the total area under the curve can be divided into a rectangular area $U(L+D_n)$ and the additional area A after the absolute deadline of the lowest priority task. Hence:

$$U(L + D_n) + A = D_n + L - B^n_{max} \tag{7}$$

from which:

$$U = 1 - A/(L + D_n) - B^n_{max}/(L + D_n) \tag{8}$$

Thus, given B^n_{max}, L and D_n, we can see that minimum U is achieved at the maximum value of A.

Let T_{max} be a task with the maximum deadline among the tasks of priority equal to or higher than P_n in the task arrival pattern. Note that more than one task can have the

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

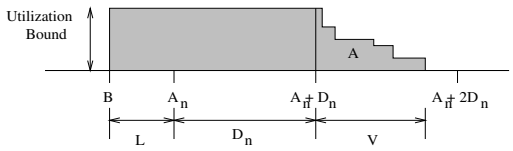


Figure 3. An example of a utilization curve for a real pattern

same deadline. Let the deadline of T_{max} be D_{max}. Let E_i, $1 \leq i \leq K$, be the tasks in the task pattern, of priority equal to or higher than that of T_n whose absolute deadlines are after the absolute deadline of the critically-schedulable task T_n. Without loss of generality, let tasks E_i be numbered in increasing order of their arrival times; E_1 is the first to arrive and E_K is the last.

Lemma 5: In the worst case pattern (when the height of the utilization curve is minimum), all tasks E_i, $1 \leq i \leq K$ satisfy the following properties

1. $A_{i+1} = A_i + C_i$, where A_i is the arrival time and C_i is the computation time for task E_i.

2. All tasks E_i are tasks of type T_{max}

Proof: Let us consider the tasks E_i (an example scenario is shown in Figure 4). For the tasks to be executable, we have the following constraints on the task arrival and computation times:

$$A_n + D_n - A_K \geq C_K \quad (9)$$
$$A_n + D_n - A_{K-1} \geq C_{K-1} + C_K \quad (10)$$
$$A_n + D_n - A_{K-2} \geq C_{K-2} + C_{K-1} + C_K \quad (11)$$
$$\ldots$$

These equations represent the constraint that the time between an arrival of task E_i and the instant $A_n + D_n$ should be more than the sum of computation times of the tasks that have arrived (Figure 4). (Remember that tasks E_i are arranged in increasing order of their arrival times.) Otherwise, some portion of the task will execute after $A_n + D_n$. That portion increases utilization without affecting the schedulability of T_n which contradicts the definition of a worst-case task pattern. We can write the above equations as:

$$A_n + D_n - A_i \geq \Sigma_{j \geq i}^{K} C_j \quad (12)$$

The area A of the utilization curve can be written down as follows:

$$A = \Sigma_{E_i}(A_i + D_i - (A_n + D_n))\frac{C_i}{D_i} \quad (13)$$

Rewriting the equation above, we get:

$$A = \Sigma_{E_i} C_i - \Sigma_{E_i}(A_n + D_n - A_i)\frac{C_i}{D_i} \quad (14)$$

As we have argued before, the minimum utilization is achieved when the Area A is the maximum. In Equation (14), the first term is fixed and hence the second term needs to be minimized. The second term can be minimized by substituting the minimum values for $A_n + D_n - A_i$ from Equation (12). On doing so, we see that we have made:

$$A_n + D_n - A_i = \Sigma_{j \geq i}^{K} C_j \quad (15)$$

If we take the difference of any two consecutive equations from Equations (15), we get:

$$A_{i+1} = A_i + C_i \quad (16)$$

which is Property 1. On substituting the values from Equations (15) into Equation (14), we get:

$$A = \Sigma_{E_i} C_i - \Sigma_{E_i}((\Sigma_{j \geq i}^{K} C_j)\frac{C_i}{D_i}) \quad (17)$$

The second term can be further reduced by taking D_i to be the maximum value it can take, which would be D_{max}. This is Property 2. Hence, we see that the worst case pattern satisfies Properties 1 and 2 (as shown in Figure 4). ■

Note that we have not considered the case that task T_n might also execute in between the tasks E_i. However, area A would be greater in the case that T_n does not execute between any E_i; so, since we want the maximum area for A, we exclude that possibility.

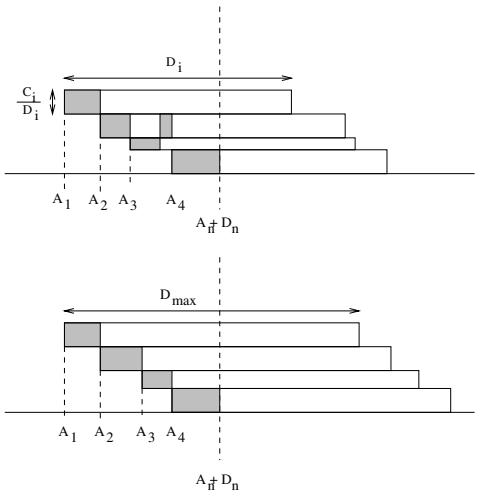


Figure 4. Properties of the set of tasks E_i

3.3 Utilization Bound

We now derive the expression for the bound in terms of D_n and D_{max}. In Figure 5, let A_n and D_n be the arrival

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

time and relative deadline, respectively, of task T_n, which is the critically schedulable task under consideration. Observe that each task E_i that is current at A_n+D_n contributes to the area under the utilization curve a rectangle of height equal to C_i/D_i. Thus, each step on line ED in Figure 5 is of height C_i/D_i for some i. By Lemma 5, these tasks have the same deadline, D_{max}, hence, $C_i/D_i = C_i/D_{max}$. Also by Lemma 5 (see Equation (16)), these tasks arrive separated by exactly their computation time. Hence, their absolute deadlines are separated by the same. Each step on line ED is therefore of width C_i. The slope of the line ED (and GC) can thus be computed from $\frac{height\,of\,step}{computation\,time} = \frac{C_i/D_{max}}{C_i} = 1/D_{max}$. Since this value is independent of the computation times of the tasks, we get GC and ED as straight lines across all the tasks E_i. As the slope of the line GC = $1/D_{max}$, it follows that distance GF = UD_{max}. Observe that area A is upper bounded by the area of the trapezoid $CDEF$. From Equation (8), to find the minimum bound area A should be maximized. Thus, in the following we assume that area A is equal to $CDEF$.

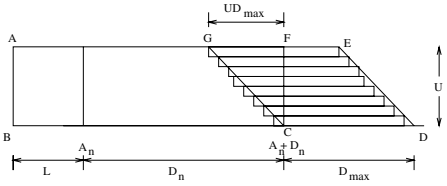


Figure 5. The Worst Case Pattern

In the following we denote areas by the names, in parenthesis, of the corresponding geometric shapes, e.g., $(ABDE)$ denotes the area contained by the mentioned vertices. Let U be the utilization bound, which we want to derive. We have the following relations:

$$\begin{aligned}(ABDE) &= Total\,Computation\,Time \\ &= D_n + L - B^n_{max} \quad (18)\\ (CDEG) &= UD_{max} \quad (19)\\ (BCGA) &= \tfrac{1}{2}U(D_n + L + (D_n + L - UD_{max})) \quad (20)\\ (ABDE) &= (CDEG) + (BCGA) \quad (21)\end{aligned}$$

Substituting from Equations (13-15) into Equation (16):

$$D_n + L - B^n_{max} = UD_{max} + \tfrac{U}{2}(D_n + L + (D_n + L - UD_{max}))$$

Simplifying, and dividing by D_{max} we get:

$$U^2 - 2(\frac{1 + D_n + L}{D_{max}})U + 2(\frac{D_n + L - B^n_{max}}{D_{max}}) = 0$$

We can now rewrite the above expression by setting $\frac{D_n}{D_{max}} = \alpha$, $\frac{L}{D_{max}} = \beta$, and $\frac{B^n_{max}}{D_n} = \gamma$, which yields:

$$U^2 - 2(1 + \alpha + \beta)U + 2(\alpha + \beta - \alpha\gamma) = 0$$

The above quadratic equation has the roots:

$$U = (1 + \alpha + \beta) \pm \sqrt{1 + \alpha^2 + \beta^2 + 2\alpha\beta + 2\alpha\gamma}$$

For $U < 1$, we should take the negative sign which gives us:

$$U = (1 + \alpha + \beta) - \sqrt{1 + \alpha^2 + \beta^2 + 2\alpha\beta + 2\alpha\gamma} \quad (22)$$

Differentiating both sides with respect to β, we get $\frac{dU}{d\beta} = 1 - \frac{\alpha+\beta}{\sqrt{1+\alpha^2+\beta^2+2\alpha\beta+2\alpha\gamma}}$. We can see that $\frac{dU}{d\beta}$ is always positive, hence U is an increasing function with respect to L and so we get the minimum U at $L = 0$ which is the minimum value that L can take. Hence:

$$u_{bound} = (1 + \alpha) - \sqrt{1 + 2\alpha\gamma + \alpha^2} \quad (23)$$

Also, if there are no resource constraints (blocking) across tasks, $\gamma = 0$ and Equation 23 reduces to

$$u_{bound} = (1 + \alpha) - \sqrt{1 + \alpha^2} \quad (24)$$

Note that the smaller the α the lower the utilization bound. Thus, to obtain the minimum bound, we must substitute with the minimum possible α, i.e., with the minimum D_n/D_{max}. We call it, the preemptable deadline ratio, i.e., the minimum ratio of the deadline of some task D_{lo} to the deadline of a task of equal or higher priority, D_{hi}. This ratio is usually a function of both the scheduling policy and the parameters of the task set. For example, for deadline-monotonic scheduling, a high priority task cannot have a larger deadline. Hence, $\alpha = 1$. The generalized utilization bound is therefore given by:

$$U_{bound} = 1 + \min\frac{D_{lo}}{D_{hi}} - \sqrt{1 + \max\frac{B^n_{max}}{D_n} + (\min\frac{D_{lo}}{D_{hi}})^2} \quad (25)$$

Observe that the above general formula reduces to the optimal tight bound derived in [2] for deadline monotonic scheduling. This tight bound was found in [2] to be equal to $\frac{1}{1+\frac{1}{\sqrt{2}}}$. For $\alpha = 1$ (deadline-monotonic) and no resource blocking $\gamma = 0$, Equation (23) gives us $2 - \sqrt{2}$ which is the same as $\frac{1}{1+\frac{1}{\sqrt{2}}}$ derived in [2].

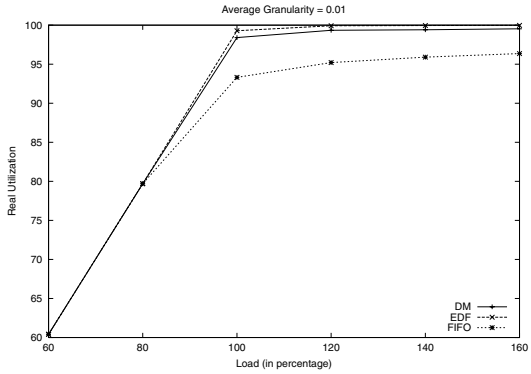
To appreciate the generality of this bound, we give one example of its use for an arbitrary scheduling policy. Consider an aperiodic task system where incoming tasks have relative deadlines that can take *any* value, say between 1 and

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 **IEEE**

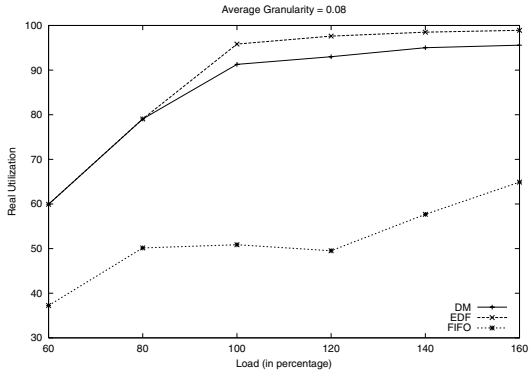
2^N. The operating system has a limited number of priority levels. Hence, the scheduler classifies all incoming tasks such that if the relative deadline of the task is in the interval $[2^k, 2^{k+1})$, the task is assigned priority k (where lower-numbered priorities are considered "higher"). Using the results derived in this paper, it is possible to derive a schedulability bound for this system. Simply observe that under the stated priority assignment, $\min \frac{D_{lo}}{D_{hi}} = 0.5$. Hence, from Equation (25), $U_{bound} = 0.382$.

4 Simulations and Evaluation

Simulations were performed to study the applicability and efficiency of the utilization bound. The simulator consists of a workload generator which can generate task sets with inter-arrival times and computation requirements having a specified distribution. For space limitations, in this paper, we shall be concerned only with Poisson arrivals. We study the utilization at a single processor which is fed with aperiodic tasks from the workload generator. The tasks undergo a schedulability check based on the utilization bound and if the synthetic utilization of the task added to the current synthetic utilization at the processor is less than the utilization bound, the task is admitted and placed into the ready queue. The processor executes tasks from the queue in an order which depends on the scheduling policy specified. When a deadline of a task is reached the synthetic utilization is decremented by the contribution of this task. When the processor becomes idle, the synthetic utilization is reset to zero, and all past task arrivals are forgotten.



a) Small tasks



b) Large tasks

Figure 6. Real Utilization for DM, EDF and FIFO (Poisson distribution)

4.1 Performance Different Scheduling Policies

In the following we evaluate the real utilization achievable with synthetic utilization based admission control under different scheduling policies. Figure 6 shows the real utilization observed for different scheduling policies as the input load is varied. In Figure 6-a the average task granularity, C_i/D_i, was small (0.01), whereas in Figure 6-b it was large (0.08). The aperiodic tasks were generated in this experiment with deadlines ranging from 2000 to 18000. The computation times were randomly selected from a Poisson distribution such that the desired average granularity is achieved. Different loads were generated by varying the Poisson rate parameter of the inter-arrival distribution. Given the chosen range of deadlines, the preemptable deadline ratio α for FIFO scheduling is 2000/18000. Accordingly, the Utilization bound for FIFO was 0.105. The utilization bound for Deadline Monotonic (DM) is 0.586 (preemptible deadline ratio = 1). The utilization bound for EDF is 1.0 [2].

As can be observed, although the synthetic utilization is limited to the bound of the respective policy, the real utilization of the resource ranges around 90%-100% for input load equal to and above 100%. This implies that even though the utilization bound might be low, the processor is not underutilized. As expected, EDF does better than Deadline Monotonic which in turn does better than FIFO. An interesting observation is that FIFO does very well when tasks are small, but significantly worse when task granularity increases.

4.2 Aperiodic and Periodic Tasks

Since periodic tasks can be broken down into aperiodic tasks, the aperiodic utilization bound is applicable to the case of a mixed workload with periodic and aperiodic tasks. In this mixed workload, a fraction of synthetic utilization is reserved for periodic tasks (equal to the utilization factor of the periodic task set). Admission control is applies to aperiodic tasks based on the leftover utilization.

Figure 7 shows the real utilization at the processor when different percentages of the workload are composed of periodic tasks. Deadline Monotonic is used in these simulations. The distributions used for arrival and computation times are Poisson. The utilization of periodic tasks is gradually increased. The aperiodic workload is kept the same.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 **IEEE**

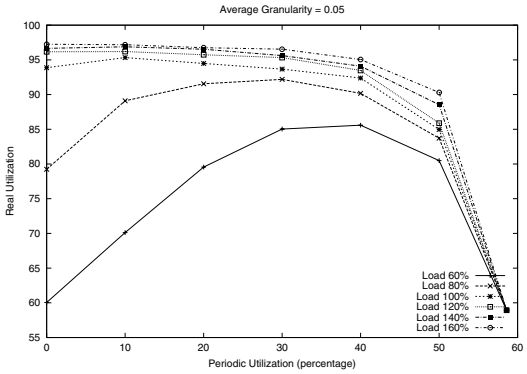


Figure 7. Real utilization for Deadline Monotonic with varying periodic tasks load

It can be seen that initially real utilization remains high as aperiodic tasks are replaced by periodic tasks. However, as the percentage of periodic tasks increases to values close to the utilization bound, fewer aperiodic tasks can get through and the real utilization approaches 58.6%. This is expected since the real-utilization becomes equal to synthetic utilization when all tasks are periodic, and thus becomes equal to the bound.

4.3 Tasks with critical sections

Figure 8 shows the variation in real utilization as the blocking factor B_{max} is varied from 0 to the total computation time of the task (100%) for different task granularities. The plots are for 100% load on the system. The priority ceiling protocol was used and the distributions for inter-arrival times and computation times was the Poisson distribution. The size of the critical sections was chosen uniformly from 0 to B_{max}.

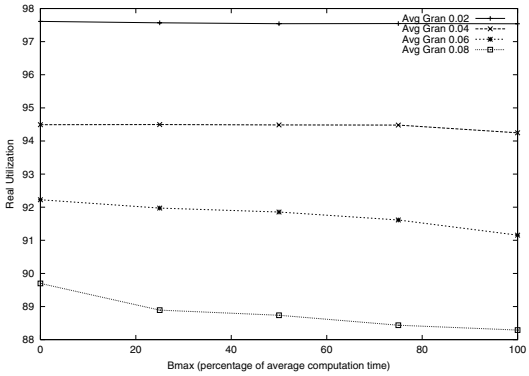


Figure 8. Variation in real utilization as Bmax is varied

Note that for smaller average computation time, B_{max} is also small and the change in the utilization bound is not significant leading to little change in real utilization. However, for larger values of average computation time and larger percentages of blocking time, the drop in real utilization is more significant. However, we could say that for realistic values of average computation times and blocking factors, the change in system performance is not much and hence workloads with resource constraints can also lead to high utilization.

5 Related Work

The basic utilization bound for periodic tasks [11] presents a sufficient (but not necessary) schedulability condition that considers all possible periodic task patterns. Several optimizations have been developed which improve the value of the bound by considering more information about the task system. For example, in [8], it is shown that if the values of task periods form K harmonic chains, where $K < n$, then the bound can be increased to $K(2^{1/K} - 1)$. In [15, 6], the bound is further improved by considering actual values of task periods. A utilization bound for a modified rate-monotonic algorithm which allows deferred deadlines is considered in [18]. A bound for large periodic task sets under the rate monotonic scheduling policy was proposed in [3]. The bound is less pessimistic than the Liu and Layland bound and accounts for resource constraints and aperiodic servers as well. The bound has also been extended to multiprocessor scheduling. In [14] a bound is derived for rate-monotonic scheduling on a partitioned multiprocessor. In [13] and [12] multiprocessor bound were derived for EDF and Rate Monotonic scheduling policies.

In their prior work, the authors derived a bound for the first time for the aperiodic task model. They derive the optimal uniprocessor bound for aperiodic tasks in [2]. In [1] they extend it to non-partitioned multiprocessor scheduling and present an approximate pessimistic generalized bound that is a function of the preemptable deadline ratio of an arbitrary time-independent scheduling policy in the special case of liquid tasks. This paper presents the first generalized aperiodic bound that considers a general, non-liquid task model.

Slack stealing [21] and aperiodic servers, such as the sporadic server [19] and the deferrable server [20], allow aperiodic tasks to be handled within a periodic task framework. Our approach does the opposite by allowing periodic tasks to be handled with an aperiodic utilization-based admission control framework. Since, individual instances of periodic tasks can be trivially broken down into aperiodic tasks, the latter approach is much simpler to implement.

Extensive work has focussed on schedulability analysis for aperiodic servers in hard real time systems in the presence of resource constraints. The priority inversion problem due to blocking was first mentioned in [9]. In [17], Sha et.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 IEEE

al. proposed modifications to schedulability tests for using utilization bounds derived by Liu and Layland [11] to guarantee schedulability of periodic tasks using the priority ceiling protocols by the rate-monotonic algorithm. Work by Ghazalie and Baker [7] take into account resource constraints in developing a schedulability test using Liu and Layland's utilization bound. Other approaches [4], [10] and [5] extend the work for aperiodic servers. The approach in this paper incorporates the effects of blocking during derivation of the aperiodic utilization bound, thus enabling derivation of bounds parameterized by a term dependent on the blocking effects. Interestingly, the resulting term is under a square root in contrast to the blocking factor in periodic task scheduling which decreases the bound linearly. Hence, our expression appears to reduce the bound less pessimistically.

In addition to being more general and less pessimistic in its accounting for blocking, the utilization bound presented in this paper significantly improves the generalized bound for liquid tasks presented in [1]. The new bound reduces to the optimal tight bound for special case of deadline-monotonic scheduling of independent aperiodic tasks.

6 Conclusions

In this paper, we presented the first synthetic utilization bound for an arbitrary scheduling policy and a non-liquid aperiodic task model. We show that the bound can be easily applied to a mix of periodic/aperiodic tasks. Further, the bound also accounts for blocking on critical sections assuming a priority ceiling protocol.

An important observation is that admission control based on synthetic utilization leads to high server utilization despite the use of a fixed-priority scheduling policy. Based on the observations, it could be concluded that aperiodic synthetic utilization based approaches can be used for providing admission control for high-performance real-time servers.

References

[1] T. Abdelzaher, B. Andersson, J. Jonsson, V. Sharma, and M. Nguyen. The aperiodic multiprocessor utilization bound for liquid tasks. In *Real-time and Embedded Technology and Applications Symposium*, San Jose, California, September 2002.

[2] T. F. Abdelzaher and C. Lu. Schedulability analysis and utilization bounds for highly scalable real-time services. In *IEEE Real-Time Technology and Applications Symposium*, TaiPei, Taiwan, June 2001.

[3] E. Bini, G. Buttazzo, and G. Buttazzo. A hyperbolic bound for the rate monotonic algorithm. In *13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[4] M. Caccamo, G. Lipari, and G. Buttazzo. Sharing resources among periodic and aperiodic tasks with dynamic deadlines. In *IEEE Real-Time Systems Symposium*, December 1999.

[5] M. Caccamo and L. Sha. Aperiodic servers with resource constraints. In *IEEE Real-Time Systems Symposium*, London, England, December 2001.

[6] X. Chen and P. Mohapatra. Lifetime behavior and its impact on web caching. In *IEEE Workshop on Internet Applications*, 1999.

[7] T. M. Ghazalie and T. P. Baker. Aperiodic servers in a deadline scheduling environment. *The Journal of Real Time Systems*, 1995.

[8] T. W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, December 1991.

[9] B. W. Lampson and D. D. Redell. Experiences with processes and monitors in mesa. *Communications of the ACM*, February 1980.

[10] G. Lipari and G. C. Buttazzo. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *Journal of Systems Architecture*, 2000.

[11] 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.

[12] J. M. Lopez, J. L. Diaz, and D. F. Garcia. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling. In *13th Euromicro Conference on Real-Time Systems*, Delft, Netherlands, June 2001.

[13] J. M. Lopez, M. Garcia, J. L. Diaz, and D. F. Garcia. Worst-case utilization bound for edf scheduling on real-time m ultiprocessor systems. In *12th Euromicro Conference on Real-Time Systems*, pages 25–33, Stockholm, Sweden, June 2000.

[14] D.-I. Oh and T. P. B. TP. Utilization bounds for n-processor rate monotone scheduling wi th static processor assignment. *Real-Time Systems*, 15(2), 1998.

[15] D.-W. Park, S. Natarajan, and A. Kanevsky. Fixed-priority scheduling of real-time systems using utilization bounds. *Journal of Systems and Software*, 33(1):57–63, April 1996.

[16] 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.

[17] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, Septemeber 1990.

[18] W. K. Shih, J. Liu, and C. L. Liu. Modified rate-monotonic algorithm for scheduling periodic jobs with deferred deadlines. *IEEE Transactions on Software Engineering*, 19(12):1171–1179, December 1993.

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

[20] 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.

[21] 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.

Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03)
0-7695-1936-9/03 $17.00 © 2003 **IEEE**