Period and Deadline Selection for Schedulability in Real-Time Systems

Thidapat Chantem[†], Xiaofeng Wang[‡], M.D. Lemmon[‡], and X. Sharon Hu[†]
[†]Department of Computer Science and Engineering, [‡]Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
{tchantem, xwang13, lemmon, shu}@nd.edu

Abstract

Task period adaptations are often used to alleviate temporal overload conditions in real-time systems. Existing frameworks assume that only task periods are adjustable and that task deadlines remain unchanged at all times. This paper formally introduces a more general real-time task model where task deadlines, which are less than or equal to task periods, are functions of task periods. This tight coupling between task deadlines and task periods has been discussed in a recent work in control systems and presents a novel real-time scheduling challenge.

To solve the period and deadline selection problem, this article identifies a feasible period-deadline combination and proposes a heuristic, which iteratively adjusts task periods and deadlines in such a way that the task set becomes schedulable. Experimental results show that the heuristic finds a solution to the period and deadline selection problem over 73% of the time, using less than three search iterations. When it is unable to find a solution to the problem, the heuristic requires less than 0.02s to run in the worst-case (with at most 100 search iterations).

1 Introduction

Task scheduling has long been an important research topic in real-time systems, where the main requirement consists of performing some functions correctly and on time (i.e., by some specific deadline). Missing a deadline in a hard real-time system may lead to catastrophic consequences, such as failure to stop an automatically controlled train on time [24].

Despite having been traditionally treated as hard real-time systems, many control systems are quite robust in the presence of certain timing perturbations. Generally speaking, depending on the system state, the sampling rate of a control system can vary within some interval without causing significant performance degradation. This observation is very useful when temporal overload situations occur. A real-time system is said to experience an overload when it cannot finish executing one or more tasks on time due to resource constraints. Although robust, if too many deadlines have been missed or if such misses occur in a highly unpredictable manner, a control system may no longer stabilize, even if all system resources are now dedicated to it.

There are two main approaches to dealing with overloads in real-time systems: (i) dropping some instances of tasks (i.e., jobs) in a controlled manner, and (ii) increasing task periods, equivalently decreasing the sampling rates, in such a way that no deadlines are missed and the performance of the system remains acceptable.

Many algorithms have been proposed to control job dropping patterns. Some examples are the (m, k) scheduling algorithms [19], the Dynamic Window-Constrained Scheduling (DWCS) algorithm [30], the skip-over algorithms [20], and the algorithms for the weakly hard real-time systems [6]. In other works such as the imprecise computation model [15] and reward-based model [2], the aim is to maximize system workload, which is assumed to be proportional to the quality of service (QoS).

Since it is sometimes more suitable to execute jobs less often instead of dropping them or allocating fewer cycles [3], we focus on such an approach in this paper. Many previous works can be found on the management of overloads in real-time systems based on task period adjustments (e.g., [22]). The works in [26], [27] and [7] solve the period selection problem for the earliest-deadline first (EDF), rate-monotone (RM), and fixed-priority scheduling algorithms, respectively. Cervin et al. propose an online period adjustment mechanism, while varying task computation times is handled in [21]. In [13], Caccamo et al. consider scenarios where the worst-case task execution times can be large but the normal task execution times tend to be very small. To efficiently use system resources while avoiding overruns, the method of task rate adaptation is combined with the use of a constant bandwidth server to guarantee hard real-time deadlines. Buttazzo et al. propose an opti-

1068-3070/08 $25.00 © 2008 IEEE
DOI 10.1109/ECRTS.2008.35

mal period selection algorithm in [10] based on the elastic task model. Many extensions to the elastic task model can be found in [9, 11, 8, 12].

In terms of schedulability tests for task sets with deadlines less than periods, Baruah et al. proposed an exact test with pseudo-polynomial running time [5]. For efficiency, we will use the sufficient test provided in [14]. However, there exists other sufficient conditions for schedulability when task deadlines are less than task periods. For instance, Devi proposed a set of sufficient schedulability tests in [16]. The main difference between this set of tests and the one in [14] is that the former requires N checks while the latter requires only one check. Some extensions to Devi's work include, but are not limited to, an approximate schedulability test [1], an adaptation to fixed-priority systems [17], and novel feasibility tests that are shown to outperform Devi's schedulability conditions [25].

Most previous works on overload management assume that only task periods can change. In [28], task deadlines vary with time, but the tasks do not have periods (i.e., tasks are non-periodic). There has also been work on determining the lower bound on task deadlines using sensitivity analysis in a periodic task model [4]. However, to the best of our knowledge, there has been no work that allows task periods and deadlines to change simultaneously.

Our first main contribution is the introduction of a more general and realistic task model where both task deadlines and task periods can vary within some intervals. The deadline in the real-time system sense really denotes the maximum allowable delay that a given task (a control task, for instance) can tolerate. As shown by the authors in [29], different sampling rates for a control system lead to different acceptable maximum delays (deadlines). Specifically, a higher sampling rate means that the corresponding control task executes more often, which, in turns, allows the system to be more tolerant to a larger delay. Conversely, a larger sampling period could make the system more susceptible to delays and thus a smaller deadline may be required. In other words, the deadline of a task is a function of its period.

The relationship between task periods and task deadlines poses an interesting scheduling problem, as one can no longer assume that increasing task periods will always improve schedulability. Although it is possible to set task deadlines to be the smallest deadlines (specified by the applications) and only vary task periods, doing so may significantly worsen schedulability. As our second main contribution, we study some interesting relationships between task periods and task deadlines that will help to solve the period and deadline selection problem. We then propose an efficient heuristic that can be used to find a set of feasible task periods and deadlines and alleviate an overload situation. Our heuristic can be applied to any real-time task set where task deadlines are less than or equal to task periods and where task deadlines are piecewise first-order differentiable functions of their respective periods. Based on some experimental results, our heuristic finds a solution to the period and deadline selection problem over 73% of the time using less than 0.02s in the worst-case.

We introduce the system model and some important assumptions in Section 2. Section 3 provides a motivating example to highlight the importance and usefulness of our work. We present our formal analysis and heuristic in Sections 4. Section 5 summarizes some experimental results and the paper concludes with Section 6.

2 Preliminaries

In this section, we describe the system model and important assumptions, as well as review some relevant schedulability tests. We also give a formal definition of the period and deadline selection problem.

2.1 System Model and Assumptions

Our system consists of a set of N periodic, synchronous tasks specified by the following 5-tuple: $(C_i, T_i, T_{i_{min}}, T_{i_{max}}, D_i)$, $i = 1, \ldots, N$, where C_i is the worst-case execution time of task τ_i, and T_i is τ_i's actual period, which must lie somewhere between $T_{i_{min}}$ and $T_{i_{max}}$. The parameter $T_{i_{min}}$ denotes the most desirable period of τ_i, as specified by the application, whereas $T_{i_{max}}$ represents the maximum period beyond which the system performance is no longer acceptable. The parameter D_i is the deadline of τ_i, and is dependent on the actual task period T_i. That is, the deadline of a task is a function of its period. Specifically, $D_i \leq T_i$, $T_i \in [T_{i_{min}}, T_{i_{max}}]$ and D_i is some function that is piecewise first-order differentiable.

The utilization of each task τ_i is defined as $U_i = C_i/T_i$ and denotes system resources dedicated to τ_i. Since the period of τ_i, $i = 1, \ldots, N$, can vary between $T_{i_{min}}$ and $T_{i_{max}}$, the minimum utilization of τ_i, $U_{i_{min}} = C_i/T_{i_{max}}$, and its maximum (desired) utilization, $U_{i_{max}} = C_i/T_{i_{min}}$, are also defined, for $i = 1, \ldots, N$.

2.2 Schedulability Test

Throughout this paper, we will assume that the Earliest Deadline First (EDF) scheduling algorithm [23] is used. When one or more tasks need to decrease their period and/or deadline in response to either internal (e.g., change in sampling rate of one or more tasks in the system) or external (e.g., network traffic) factors, a schedulability test must be used to assess whether the task set is still schedulable. A schedulability test may also provide some guidance on how to adjust task parameters in such a way that a feasible task set can be obtained. Based on the assumption that the

EDF scheduling algorithm is used, there exist some useful schedulability conditions that are briefly reviewed here.

A necessary condition for schedulability of any given task set is stated in the following lemma.

Lemma 1 [14] *Given a task set Γ, let C_i and D_i be the execution time and the deadline of task τ_i, $i = 1, \ldots, N$, respectively. In addition, let all tasks start at time 0 and let the tasks in Γ be ordered in a non-decreasing order of deadlines. Regardless of the choices of periods, any task set that is schedulable must satisfy the following property*

$$\sum_{i=1}^{j} C_i \leq D_j, j = 1, \ldots, N. \tag{1}$$

Since task deadlines can be less than or equal to periods, there exist an exact, albeit complex, schedulability test for EDF as specified by Baruah et al [5]. Said test is restated in the following theorem.

Theorem 1 [5] *Given a periodic task set with C_i, D_i, and T_i as the execution time, deadline, and period of task τ_i, $i = 1, \ldots, N$, respectively. Let $D_i \leq T_i$, $i = 1, \ldots, N$, the task set is schedulable if and only if the following constraint is satisfied $\forall L \in \{kT_i + D_i \leq \min(B_p, H)\}$ and $k \in \mathcal{N}$ (the set of natural numbers including 0), where B_p and H denote the busy period and hyperperiod, respectively,*

$$L \geq \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i. \tag{2}$$

Verifying that (2) is satisfied for all L is the main source of complexity in the above schedulability test. To reduce the complexity of the test in Theorem 1, the authors in [14] proposed the following sufficient condition for schedulability.

Theorem 2 [14] *Given a set Γ of N tasks that satisfy Lemma 1. Let C_i, D_i, and T_i be the execution time, deadline, and period of task τ_i, $i = 1, \ldots, N$, respectively. In addition, let the tasks in Γ be sorted in a non-decreasing order of deadlines. The task set Γ is schedulable if*

$$L^* \geq \sum_{i=1}^{N} \left(\frac{L^* - D_i}{T_i} + 1 \right) C_i \tag{3}$$

where

$$L^* = \begin{cases} D_2 & : \quad D_1 + T_1 \leq D_2 \\ \min_{i=1}^{N} (T_i + D_i) & : \quad \text{otherwise.} \end{cases}$$

For completeness, we include another existing sufficient condition for EDF schedulability.

Theorem 3 [24] *Consider a set Γ of N tasks where C_i and D_i are the execution time and deadline of task τ_i, $i = 1, \ldots, N$, respectively. The task set Γ is schedulable by the EDF policy if*

$$\sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1. \tag{4}$$

We will use some of these schedulability conditions in Section 4.

2.3 Problem Definition

We consider the following problem: Given an initially infeasible set Γ of N real-time tasks where the period T_i of task τ_i must lie somewhere between $[T_{i_{min}}, T_{i_{max}}]$, and the deadline D_i of τ_i is some function of its period, determine a period-deadline combination (T_i, D_i), $i = 1, \ldots, N$, such that the task set Γ becomes schedulable. In other words, we wish to find (T_i, D_i), $i = 1, \ldots, N$, such that

$$\sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq L \tag{5}$$

$$T_i \geq T_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{6}$$

$$T_i \leq T_{i_{max}} \quad \text{for } i = 1, 2, \cdots, N \tag{7}$$

where L is defined as in Theorem 1, C_i is the worst-case execution time of τ_i, and both $T_{i_{min}}$ and $T_{i_{max}}$ are specified by the applications under consideration.

The constraint in (5) ensures the schedulability of the task set. The constraints in (6) and (7) bound the period of τ_i, $i = 1, \ldots, N$, to ensure performance.

3 Motivations

In control systems, an advantage in using the traditional periodic task model where task deadlines are fixed is that these systems can be treated as discrete-time systems for which there exists a variety of mature controller synthesis methods. However, when the periodic task model is used, task periods and deadlines are often chosen conservatively to guarantee stability. This leads to wasted resources and perhaps even system over-provisioning. For these reasons, there has been a recent movement in the control system community to investigate alternative approaches to the periodic task model.

The work in [29] is such an example. Each task determines its next release time based on the current system state as sampled by the current job. This type of control systems is known as state-based self-triggering systems. Self-triggering can be viewed as a closed-loop form of releasing tasks for execution, whereas the traditional periodic task

Table 1. Task set for motivating example

Task	C_i	$T_{i_{min}}$	$T_{i_{max}}$	D_i
τ_1	0.18	0.5	3.5	$T_1 e^{-T_1}, T_1 \in [0.5, 3.5]$
τ_2	0.18	0.5	3.5	$T_2 e^{-T_2}, T_2 \in [0.5, 3.5]$

model is considered open-loop. Since each control task is aware of its system state, it can adjust its period and deadline in such a way that only the required system resource is requested. More precisely, with a small period, a task is executed relatively often and the system is thus more tolerant to delays, permitting the task deadline to be larger (e.g., perhaps almost as large as the task period itself). On the other hand, when the task period is large, the system is more susceptible to disturbances, requiring that the task deadline be smaller (compared to the task period) to reduce jitters.

To understand how the deadline as a function of the period affects schedulability, let us consider a simple task set, which consists of two identical tasks whose attributes are shown in Table 1. The deadline of each task can be computed as shown in the last column of Table 1 (all units are in milliseconds). Figure 1 plots the task deadlines as a function of task periods where the vertical dotted lines limit the acceptable period range for the example tasks. Initially, the task set is not schedulable with $T_1 = T_2 = 0.5$ms, since the initial deadlines $D_1 = D_2 = 0.303$ms and the aggregate execution time required is 0.36ms. If we simply set $T_1 = T_2 = 3.5$ms, which is the maximum allowable periods, then the corresponding deadlines will be $D_1 = D_2 =$ 0.106ms. The task set is, again, not schedulable and one may wrongly conclude that the task set cannot be made feasible. However, there exists many feasible period-deadline combinations. For example, when $T_1 = T_2 =$ 1ms and $D_1 = D_2 = 0.368$ms, the task set is schedulable.

In the traditional periodic task model, since task deadlines are considered fixed, system designers must use the smallest possible deadlines to ensure that, given a specific range of task periods, the system will always meet the minimum performance requirements. For the above example, the smallest deadline for both tasks is 0.106ms, which means that the task set can never be made schedulable using existing techniques. It is not difficult to see in this example that the task deadlines can be set to 0.36ms for the task set to be feasible, regardless of the resultant periods. In general, however, both task periods and task deadlines must be considered simultaneously, since different tasks may have different timing requirements.

4 Period and Deadline Selection

As shown in the previous section, since a task's deadline is a function of its period, adjusting the period affects both the corresponding deadline and the schedulability of the entire task set. Due to the condition in (5), the problem defined in Section 2.3 is nonlinear, non-convex, and non-continuous since L, T_i, and D_i, $i = 1, \ldots, N$, are variables, and because of the floor function. Solving the above problem directly using a nonlinear solver is inefficient and it cannot be guaranteed that a solution will be found, even if one exists. For these reasons, we propose using a heuristic which uses a number of fast sufficient conditions to find a solution. In a nutshell, the heuristic starts by performing some simple schedulability tests to determine a feasible period-deadline combination. Such tests also serve to eliminate some infeasible period and deadline values should they fail to identify a feasible task set. The heuristic then uses this knowledge to conduct an efficient search process.

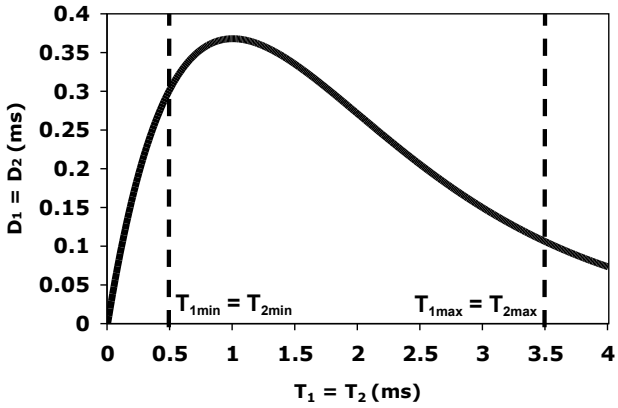

Figure 1. Deadline as a function of period

4.1 Identifying Infeasible Regions Using Simple Tests

We now describe our idea of using the simple tests in more detail. We first determine the minimum and maximum deadlines, $D_{i_{min}}$ and $D_{i_{max}}$, respectively, for each task τ_i, $i = 1, \ldots, N$. The maximum deadline of τ_i, $D_{i_{max}}$ can directly be solved by finding the maximum of D_i. (Recall that the maximum of a function can be obtained by taking its derivative and subsequently finding the root(s) of said derivative.) The corresponding period value is denoted T_i^{Dmax}, $i = 1, \ldots, N$.

To determine the lower bound on the deadline of a task τ_i, $i = 1, \ldots, N$, we would ideally use Lemma 1. However, Lemma 1 requires that tasks be sorted in a non-decreasing order of deadlines. Since a task deadline is a variable to be determined, we cannot directly use Lemma 1 to compute the minimum deadline. Instead, let $\widetilde{D}_i$ be the smallest deadline of task τ_i, i.e., $\widetilde{D}_i \leq D(T_i)$, $T_i \in [T_{i_{min}}, T_{i_{max}}]$, $i = 1, \ldots, N$. We say that task τ_i dominates task τ_j (denoted by $\tau_i \succeq \tau_j$) if $\widetilde{D}_i > D_{j_{max}}$. Otherwise, we say

that τ_i and τ_j are non-comparable. Using the above dominance definition, a partial order can be built for a given set of tasks. It is easy to see that Lemma 1 holds true for tasks with deadlines as variables if we sort the tasks using the partial order established above. For example, consider a simple task set consisting of task τ_j and τ_k. If $\tau_j \succeq \tau_k$ then $D_{k_{min}} = C_k$ and $D_{j_{min}} = C_k + C_j$. In general, for a task τ_i, $D_{i_{min}} = \max\{DS\} + C_k$, where DS is the set of deadlines of tasks that are dominated by τ_i. Since $D_{i_{min}}$ set in this way is a lower bound on the minimum task deadline for task τ_i, $i = 1, \ldots, N$, we can eliminate some infeasible period-deadline combinations (shown by the right-slanted pattern in Figure 2). The task period that corresponds to when the task deadline is $D_{i_{min}}$ is referred to as T_i^{Dmin}, $i = 1, \ldots, N$.

Once we have found the minimum and maximum deadlines for each task in the task set, we can apply a series of efficient schedulability tests to avoid searching for a solution, if possible. We start with the sufficient condition from Theorem 3 using $D_{i_{max}}$, $i = 1, \ldots, N$, as the task deadlines. The following lemma helps to explain why only $D_{i_{max}}$, $i = 1, \ldots, N$, need to be considered when applying Theorem 3 on the current task set.

Lemma 2 *Given a set Γ of N tasks. Let C_i and D_i be the execution time and deadline of task τ_i, $i = 1, \ldots, N$, respectively. If the schedulability condition from Theorem 3 is not satisfied for $D_{i_{max}}$, $i = 1, \ldots, N$, then it is not satisfied for any $D_i < D_{i_{max}}$, $i = 1, \ldots, N$.*

Proof: If the task set Γ fails the schedulability test in Theorem 3, then

$$\sum_{i=1}^{N} \frac{C_i}{D_{i_{max}}} > 1. \tag{8}$$

Using any $D_i < D_{i_{max}}$, $i = 1, \ldots, N$, would yield

$$\sum_{i=1}^{N} \frac{C_i}{D_i} > \sum_{i=1}^{N} \frac{C_i}{D_{i_{max}}} > 1. \tag{9}$$

Therefore, the lemma holds. □

Note that if the condition from Theorem 3 is satisfied for $D_{i_{max}}$, $i = 1, \ldots, N$, then we have identified a feasible solution. Otherwise, we apply the schedulability test from Theorem 2 for a special point $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$ (see Figure 2). (To use Theorem 2, we order the tasks in a non-decreasing order of deadlines using $D_i = D_{i_{max}}$ and $T_i = T_i^{Dmax}$, $i = 1, \ldots, N$, whenever L needs to be determined). We choose to test the point $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$, because if the task set is not schedulable at this point according to Theorem 2, then it is not schedulable for any $(\overline{T}_i, \overline{D}_i)$, $\overline{T}_i \leq T_i^{Dmax}$ and $\overline{D}_i \leq D_{i_{max}}$, $i = 1, \ldots, N$. The following theorem proves this claim and explains why the left-slanted region in Figure 2 can be eliminated from further consideration if the point $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$, are found not to be schedulable according to Theorem 2.

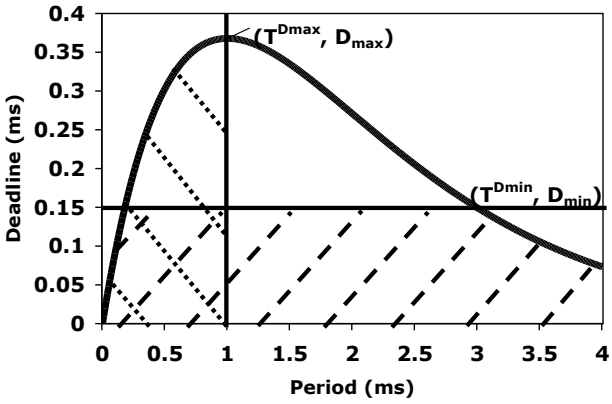

Figure 2. Infeasible schedulability regions

Lemma 3 *Given a set Γ of N tasks. Let C_i and D_i be the execution time and deadline of task τ_i, $i = 1, \ldots, N$, respectively. Let T_i be the period obtained when $D_i = D_{i_{max}}$, $i = 1, \ldots, N$. If the condition in Theorem 2 is not satisfied for (T_i, D_i), $i = 1, \ldots, N$, then it is not satisfied for any $(\overline{T}_i, \overline{D}_i)$, $\overline{D}_i \leq D_i$, $\overline{T}_i \leq T_i$, $i = 1, \ldots, N$.*

Proof: Since the task set is not schedulable at (T_i, D_i), $i = 1, \ldots, N$, we have

$$L < \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) C_i. \tag{10}$$

In addition, since $\overline{D}_i \leq D_i$ and $\overline{T}_i \leq T_i$, $i = 1, \ldots, N$,

$$\begin{aligned} L &< \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) C_i \\ &< \sum_{i=1}^{N} \left(\frac{L - \overline{D}_i}{\overline{T}_i} + 1 \right) C_i, \end{aligned} \tag{11}$$

since $L - \overline{D}_i > L - D_i$ and $\overline{T}_i < T_i$, $i = 1, \ldots, N$. Finally, as $L > \overline{L}$, where $\overline{L} = \overline{D}_2$ if $\overline{D}_1 + \overline{T}_1 \leq \overline{D}_2$ and $\overline{L} = \min_{i=1}^{N}(\overline{T}_i + \overline{D}_i)$ otherwise,

$$\overline{L} < \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) C_i. \tag{12}$$

□

Observe that we use the schedulability conditions in Theorems 2 and 3 in conjunction to one another. This is because a task set that is feasible according to one of the aforementioned schedulability conditions is not necessarily feasible according to the other (and vice versa).

The left-slanted region in Figure 2 is a result of Lemma 3 and can be eliminated from further consideration. (Note that since we test the point $(T_i^{Dmax}, D_{i_{max}})$ using the schedulability test from Theorem 2, such a point may in fact be feasible. However, to exactly determine schedulability, the condition in Theorem 1 needs to be satisfied and is often too time consuming to be used during an overload situation.) The area with no pattern, however, indicates the remaining search region. Note that for a specific period T_i, $i = 1, \ldots, N$, any deadline $0 < \overline{D}_i < D_i$ is also acceptable from the system performance point of view. However, since using $\overline{D}_i$ will worsen schedulability, we only consider D_i. All the simple tests described thus far appear as part of our heuristic and are shown in Algorithm 1. Lines 1–4 show the first simple test discussed in Lemma 2. The second simple test from Lemma 3 is shown in Lines 5–9. Finally, Lines 10–14 show that we perform an additional schedulability test for $(T_i^{Dmin}, D_{i_{min}})$, $i = 1, \ldots, N$, since this point has already been computed and such an additional test does not incur a significant amount of additional overhead.

Algorithm 1 SimpleTests(Γ)

1: $result \leftarrow \sum_{i=1}^{N} \frac{C_i}{D_{i_{max}}}$
2: **if** $result \leq 1$ **then**
3: **return** $\left[D_{i_{max}}, T_i^{Dmax}\right]$, for $i = 1, \ldots, N$
4: **end if**
5: compute L as in (14) using $D_{i_{max}}$ and T_i^{Dmax}, $1 \leq i \leq N$
6: $result \leftarrow \sum_{i=1}^{N} \left(\frac{L - D_{i_{max}}}{T_i^{Dmax}} + 1\right) \cdot C_i$
7: **if** $result \leq L$ **then**
8: **return** $\left[D_{i_{max}}, T_i^{Dmax}\right]$, for $i = 1, \ldots, N$
9: **end if**
10: compute L as in (14) using $D_{i_{min}}$ and T_i^{Dmin}, $1 \leq i \leq N$
11: $result \leftarrow \sum_{i=1}^{N} \left(\frac{L - D_{i_{min}}}{T_i^{Dmin}} + 1\right) \cdot C_i$
12: **if** $result \leq L$ **then**
13: **return** $\left[D_{i_{min}}, T_i^{Dmin}\right]$, for $i = 1, \ldots, N$
14: **end if**
15: **return** $\emptyset$

4.2 Efficiently Conducting the Search Process

If all the aforementioned simple tests fail, we will have to search along the unpatterned region of Figure 2 to find a feasible period-deadline combination, (T_i, D_i), $i = 1, \ldots, N$ (Algorithm 2). Since the main source of complexity of the problem defined in Section 2.3 is that (5) must be satisfied for all possible values of L, the search process will instead use the schedulability test from Theorem 2. In other words, the problem in Section 2.3 is modified to

$$\sum_{i=1}^{N} \left(\frac{L - D_i}{T_i}\right) C_i \leq L - \sum_{i=1}^{N} C_i \tag{13}$$

$$L = \begin{cases} D_2 & : D_1 + T_1 \leq D_2 \\ \min\left(T_i + D_i\right) & : \text{otherwise} \end{cases} \tag{14}$$

$$T_i \leq \min\left\{T_{i_{max}}, T_i^{Dmin}\right\},\ i = 1, 2, \cdots, N \tag{15}$$

$$T_i \geq \max\left\{T_{i_{min}}, T_i^{Dmax}\right\},\ i = 1, 2, \cdots, N \tag{16}$$

where T_i^{Dmin} and T_i^{Dmax}, $i = 1, \ldots, N$ are as defined previously.

Given an initially infeasible task set, one can compute the corresponding value of L as in (14). Let us first assume that the value of L is fixed once it has been computed. To satisfy the condition in (13), observe that since the right-hand side of (13) can be treated as a constant, one way to solve the above problem is to adjust task periods and deadlines such that the left-hand side becomes as small as possible. We can express this idea mathematically as the following constrained optimization problem.

$$\min: \quad \sum_{i=1}^{N} \left(L - \widehat{D}_i\right) \cdot U_i \tag{17}$$

$$\text{s.t.}: \quad U_i \leq U_i^{max} \quad \text{for } i = 1, 2, \cdots, N \tag{18}$$

$$U_i \geq U_i^{min} \quad \text{for } i = 1, 2, \cdots, N \tag{19}$$

where $\widehat{D}_i$ is the task deadline function that depends on U_i, i.e., $\widehat{D}_i \equiv \widehat{D}_i(U_i) = D_i(C_i/U_i)$. (For notational simplicity, D_i always refers to $D_i(T_i)$ and $\widehat{D}_i$ to $D_i(U_i)$.) $U_i = C_i/T_i$, $U_i^{max} = \frac{C_i}{\max\left\{T_{i_{min}}, T_i^{Dmax}\right\}}$, and $U_{i_{min}} = \frac{C_i}{\min\left\{T_{i_{max}}, T_i^{Dmin}\right\}}$.

Solving the above constrained optimization problem is attractive because if a solution to the problem in (13)–(16) exists for a fixed value of L, then we will find it by solving the above constrained optimization problem. This claim is formally stated in the following lemma.

Lemma 4 *Given an initially infeasible task set Γ where C_i, U_i, and $\widehat{D}_i$ denote the execution time, utilization, and deadline (as a function of the utilization) of task τ_i, $i = 1, \ldots, N$, respectively. For a fixed value of L, if there exists a solution to the problem in (13)–(16), it will be found by solving the problem defined in (17)–(19).*

Proof: The lemma can be trivially proved by observing that (13) can be rewritten as

$$\sum_{i=1}^{N} \left(L - \widehat{D}_i\right) \cdot U_i \leq L - \sum_{i=1}^{N} C_i. \tag{20}$$

The constrained optimization problem in (13)–(16) minimizes the left-hand side of the above equation. Thus, if we

can adjust task periods and deadlines such that (20) is true, then the solution to the optimization problem in (17)–(19) will also be a solution to the problem in (13)–(16). □

The following theorem presents a globally optimal solution to the problem in (17)–(19) and hence a solution to the problem in (13)–(16), for a fixed value of L.

Theorem 4 *Given the constrained optimization problem as specified in (17)–(19). Let U_i be the utilization of task τ_i, $i = 1, \ldots, N$. Let $\widehat{D}_i$ be the deadline of τ_i where $\widehat{D}_i$ is a function of the U_i, i.e., $\widehat{D}_i \equiv \widehat{D}_i(U_i) = D_i(C_i/U_i)$, and let*

$$G_i(U_i) = (L - \widehat{D}_i) \cdot U_i. \tag{21}$$

For a fixed value of L, the solution, U_i, is optimal if and only if

$$U_i = \text{argmin}_{U_i \in \left\{ \overline{\mathbf{U}}_i \cup U_i^{min} \cup U_i^{max} \right\}} \{G_i(U_i)\} \tag{22}$$

where $\overline{\mathbf{U}}_i$ is a set of $\overline{U}_i$ such that $L - \widehat{D}_i - \widehat{D}_i' \overline{U}_i = 0$.

Proof: We prove that if U_i, $i = 1, \ldots, N$, is an optimal solution to the constrained optimization problem in (17)–(19), then (22) must be true by utilizing the Kuhn-Tucker (KKT) necessary conditions for optimality for constrained optimization problem, which can be written in terms of the Lagrangian function for the problem as

$$\begin{aligned} J_a(\mathbf{U}, \mu) &= \sum_{i=1}^{N} (L - \widehat{D}_i) \cdot U_i + \sum_{i=1}^{N} \mu_i (U_i^{min} - U_i) \\ &+ \sum_{i=1}^{N} \lambda_i (U_i - U_i^{max}) \end{aligned} \tag{23}$$

where μ_i's and λ_i's are Lagrange multipliers, $\mu_i \geq 0$ and $\lambda_i \geq 0$, $i = 1, \ldots, N$. The necessary conditions for the existence of a relative minimum at U_i are, for $i = 1, \ldots, N$,

$$0 = L - \widehat{D}_i' U_i - \widehat{D}_i - \mu_i + \lambda_i \tag{24}$$

$$0 = \mu_i (U_i^{min} - U_i) \tag{25}$$

$$0 = \lambda_i (U_i - U_i^{max}) \tag{26}$$

From (24)

$$L - \widehat{D}_i' U_i - \widehat{D}_i = \mu_i - \lambda_i \tag{27}$$

If $L - \widehat{D}_i' U_i - \widehat{D}_i < 0$ for $U_i \in \left[U_i^{min}, U_i^{max}\right]$, then μ_i must be 0 and $\lambda_i > 0$. Hence, $U_i = U_i^{max}$. If $L - \widehat{D}_i' U_i - \widehat{D}_i > 0$ for $U_i \in \left[U_i^{min}, U_i^{max}\right]$ then $\lambda_i = 0$ and $\mu_i > 0$. Therefore, $U_i = U_i^{min}$. Otherwise, $L - \widehat{D}_i' U_i - \widehat{D}_i = 0$ at least once when $U_i \in \left[U_i^{min}, U_i^{max}\right]$. In such a case, we can find the value(s) of U_i by finding all the extreme points in the interval $\left[U_i^{min}, U_i^{max}\right]$, which is equivalent to solving the equation $L - \widehat{D}_i - \widehat{D}_i' U_i = 0$ for U_i. Note that since the KKT conditions are necessary for optimality, we have completed the proof for this part.

Now, we prove that if U_i, $i = 1, \ldots, N$, is determined as in (22), then it is an optimal solution to the constrained optimization problem in (17)–(19). We start by observing that, given a piecewise differentiable function $G_i(U_i)$, the global minimum of $G_i(U_i)$ in the interval $\left[U_i^{min}, U_i^{max}\right]$ must either be at one of the extreme points inside $\left[U_i^{min}, U_i^{max}\right]$ or at the boundaries, i.e., at U_i^{min} or U_i^{max}. This is indeed captured by the expression in (22).

Finally, since the objective function in (17) can be rewritten as $\min_{i=1}^{N} G_i(U_i)$, minimizing each individual $G_i(U_i)$, $i = 1, \ldots, N$, is equivalent to minimizing (17). □

We use the result from the above theorem directly in the main part of our heuristic (Line 26 in Algorithm 2). Although the heuristic can optimally solve the problem in (17)–(19) for a fixed value of L, it needs to iteratively search for a feasible task set. This is because the value of L may either increase or decrease as D_i and T_i, $i = 1, \ldots, N$, change. Consider two consecutive iterations h and $h+1$. If the task set with periods $T_i^{(h)}$ and deadlines $D_i^{(h)}$, $i = 1, \ldots, N$, satisfies the constraints in (13)–(16) given some fixed value of $L^{(h)}$ and $L^{(h+1)} \geq L^{(h)}$, then the task set is guaranteed to be schedulable (as shown in the following lemma) and the search process ends.

Remark: If the left-hand side of (17) is a convex function, then the KKT necessary conditions for optimality also become sufficient conditions. In such a case, a global optimal solution to the optimization problem in (17)–(19) for a non-fixed L can be found using Theorem 4.

Lemma 5 *Given a set Γ of N tasks, and let C_i, T_i, and D_i be the execution time, period, and deadline of task τ_i, $i = 1, \ldots, N$, respectively. If the task set satisfies the condition in Theorem 2 for some L, then it also satisfies the condition in Theorem 2 for any $\overline{L} \geq L$.*

Proof: We have

$$L \geq \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) C_i \tag{28}$$

$$L - \sum_{i=1}^{N} L \frac{C_i}{T_i} \geq \sum_{i=1}^{N} C_i - \frac{D_i C_i}{T_i} \tag{29}$$

$$L \left(1 - \sum_{i=1}^{N} \frac{C_i}{T_i} \right) \geq \sum_{i=1}^{N} C_i - \frac{D_i C_i}{T_i} \tag{30}$$

which clearly holds for any $\overline{L} \geq L$. □

Now, if $L^{(h+1)} < L^{(h)}$, the schedulability condition in Theorem 2 must explicitly be checked (Lines 17- 20 in Algorithm 2). In this way, the heuristic will either return a feasible task set or continue searching until the number of

Algorithm 2 FindFeasiblePeriodsDeadlines(Γ, $maxIter$)

1: **for** each $\tau_i \in \Gamma$ **do**
2: $D_{i_{max}} \leftarrow \max_{T_i \in [T_{i_{min}}, T_{i_{min}}]} D_i$
3: $T_i^{Dmax} \leftarrow$ period when deadline is $D_{i_{max}}$
4: $D_{i_{min}} \leftarrow C_i$
5: $T_i^{Dmin} \leftarrow$ period when deadline is $D_{i_{min}}$
6: **end for**
7: $result \leftarrow$ `SimpleTests`(Γ)
8: **if** $result \neq \emptyset$ **then**
9: **return** $[D_i, T_i], i = 1, \ldots, N$
10: **end if**
11: $D_i \leftarrow D_{i_{max}}, i = 1, \ldots, N$
12: $T_i \leftarrow T_i^{Dmax}, i = 1, \ldots, N$
13: $done \leftarrow false$
14: $iterNum \leftarrow 0$
15: **while not** $done$ **do**
16: $iterNum \leftarrow iterNum + 1$
17: compute L as in (14) using D_i and T_i, $1 \leq i \leq N$
18: $result \leftarrow \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1 \right) \cdot C_i$
19: **if** $result \leq L$ **then**
20: **return** $[D_i, T_i], i = 1, \ldots, N$
21: **end if**
22: **if** $iterNum > maxIter$ **then**
23: $done \leftarrow true$
24: **end if**
25: **for** each $\tau_i \in \Gamma$ **do**
26: compute U_i as in Theorem 4
27: $T_i \leftarrow \frac{C_i}{U_i}$
28: determine D_i accordingly
29: **end for**
30: **end while**
31: **return** $\emptyset$

maximum iterations, *maxIter*, has been reached (Line 22). The value *maxIter* is a user-defined constant and, from our experiments in the next section, can be set to some small number such as 100.

The time complexity of our heuristic is dominated by the while loop on Line 15 of Algorithm 2. Inside the while loop, the most time consuming operations appear inside the for-loop on Line 25. Let $|\overline{\mathbf{U}}_i|$ be size of $\overline{\mathbf{U}}_i$ as defined in Theorem 4 over all iterations and let $O(G')$ be the worst-case time complexity required to find all solutions to the equation $L - \widehat{D}_i - \widehat{D}_i' U_i = 0$, also from Theorem 4. The running time of our heuristic is then $O(maxIter \cdot N \cdot (|\overline{\mathbf{U}}_i| + O(G')))$, where N is the number of tasks in the task set.

Remark: In our approach, we assume that when a task set is infeasible, each task is equally responsible for reducing its processor demand (if possible) to alleviate the overload situation. In practice, however, some tasks may be more important than the others. As a result, a weight may be associated with each task to denote its importance. In such a case, our approach can be extended by factoring in the weight of each task when deciding the amount of processor demand reduction that each task should be responsible for. Specifically, the problem formulation in Section 2.3 can be modified to a constrained optimization problem of the form

$$\min : \quad \sum_{i=1}^{N} w_i (T_{i_{min}} - T_i)^2 \tag{31}$$

$$\text{s.t. :} \quad \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1 \right) C_i \leq L \tag{32}$$

$$T_i \geq T_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{33}$$

$$T_i \leq T_{i_{max}} \quad \text{for } i = 1, 2, \cdots, N \tag{34}$$

where w_i is the weight of the task τ_i, $i = 1, \ldots, N$, and all other parameters retain their meaning as previously defined. Clearly, the modified problem can be too time consuming (and perhaps too difficult) to solve using a nonlinear solver and thus the use of a heuristic similar to the one presented earlier is recommended.

Table 2. Main results

Method	Number of solutions found	% solutions found
Fixed deadline technique	0/80	0%
Our heuristic	59/80	73.8%

Table 3. Heuristic total running time and number of iterations

Number of task sets	Running Time (s)	Number of iterations needed
37 (solution found)	< 0.01	< 3
13 (solution not found)	< 0.02	> 100

5 Experimental Results

Since directly solving the period and deadline selection problem in Section 2.3 using a commercial non-linear solver can be very time consuming and it cannot be guaranteed that a solution will be found, even if one exists, we proposed an efficient heuristic in Section 4. In this section, we evaluate the performance of our approach.

Due to the lack of realistic benchmarks suitable for the intended experiment, we randomly generated 80 task sets consisting of 5 tasks each. In order to scrutinize the search aspect of the heuristic, each task set is chosen such that it is initially infeasible with the guarantee that all the three simple tests from Algorithm 1 will fail. In addition, given a task set, there exists at least one period-deadline combination, (T_i^*, D_i^*), for each task τ_i, $i = 1, \ldots, N$, such that the task set can be made schedulable using the schedulability test from Theorem 2 (and hence satisfies the necessary and sufficient condition from Theorem 1).

In our experiment, we use the following deadline function, whose curve is representative of the relationship between task periods and task deadlines of the type of control systems under consideration. (It is worth noting, however, that any deadline function can be used, as long as it is piecewise first-order differentiable.)

$$D_i = \frac{k1_i}{T_i - k2_i}, \tag{35}$$

$i = 1, \ldots, N$, where $k1_i$ and $k2_i$, $i = 1, \ldots, N$, are some constants that depend on the specific task under consideration. To find $k1_i$ and $k2_i$, $i = 1, \ldots, N$, we start by randomly generating the point $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$, which denotes the point where the deadline for the task τ_i is maximum. In addition, we ensure that the point $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$, is not schedulable according to Theorems 2 and 3. (Recall that the purpose of the experiment is to test the search aspect of the heuristic and therefore we have to ensure that the simple tests fail.) Note that the deadline function in (35) is defined only for $T_i \geq T_i^{Dmax}$, $i = 1, \ldots, N$, since according to Lemma 3, any task set that is not schedulable for $(T_i^{Dmax}, D_{i_{max}})$, $i = 1, \ldots, N$, will not be schedulable for any (T_i, D_i), $T_i \leq T_i^{Dmax}$, $D_i \leq D_{i_{max}}$, $i = 1, \ldots, N$. In other words, any period $T_i < T_i^{Dmax}$, $i = 1, \ldots, N$, can be ignored by the search process.

The following steps were taken to generate a task set. First of all, the following parameters were specified: utilization level, maximum hyperperiod, minimum period, maximum period, precision, and maximum number of tries. Based on these parameters, task periods are generated in such a way that the hyperperiod is no larger than the maximum hyperperiod. (This could take a number of tries.) In our experiment, we set the maximum hyperperiod, minimum period, and maximum period to 500,000, 10,000, and 40,000, respectively. The precision was specified to be 100, whereas the maximum number of tries was set to 10,000. The precision denotes the minimum increment in any task period. For example, if the precision is set to 100, a task period could be 5200, but not 5010. Finally, for the task sets used in our experiment, the range for the utilization level was between 0.5 and 0.7.

Each task is randomly assigned an execution time such that the total utilization equals that specified by the user. No task will have an utilization that is greater than half of the specified total utilization. Then, each task is randomly assigned a deadline D_i that ensures that $\sum_{i=1}^{N} \frac{C_i}{D_i} > 1$. As a final step, the random task generator tests the schedulability of the task set using the necessary and sufficient condition from Theorem 1. If the task set is unschedulable, task deadlines are randomly increased such that the new deadline is greater than the previous deadline but $\sum_{i=1}^{N} \frac{C_i}{D_i}$ is still greater than 1. This final step is repeated until either a feasible task set has been found or the maximum number of tries has been reached.

After the generation of the aforementioned random points, each task set will be associated with two points: $(T_i^{Dmax}, D_{i_{max}})$ and (T_i^*, D_i^*), $i = 1, \ldots, N$, where the former point is not schedulable according to Theorem 2, but the latter point is. Using these two points, the constants $k1_i$ and $k2_i$, $i = 1, \ldots, N$, can be found. Finally, the point $(T_i^{Dmin}, D_{i_{min}})$, $i = 1, \ldots, N$ can be determined as described in Section 4.

We implemented the heuristic proposed in the last section in C++. The user-defined parameters *maxIter* was set to 100, which means that at most 100 search iterations were conducted for each task set (benchmark). The proposed heuristic found a feasible period-deadline combination for 59 out of the 80 task sets. For these benchmarks, if we were to use existing techniques where task periods are fixed (which do not directly apply to the system model under consideration), then no solution will be found for any of these task sets because these techniques assume that if the task set is not schedulable for $(T_i^{Dmin}, D_{i_{min}})$, $i = 1, \ldots, N$, then it cannot be made feasible. (In other words, the schedulability test from Theorem 2 is performed for $(T_i^{Dmin}, D_{i_{min}})$, $i = 1, \ldots, N$. This test is referred to as the "fixed deadline technique" in Table 2.) Clearly, due to the dependency between task periods and task deadlines, the fixed deadline technique is shown to be too pessimistic. Table 2 summarizes the results which show that our heuristic has an overall success rate of over 73% while the fixed deadline technique has a success rate of 0%. Further, since the left-hand side of (17) is a convex function (due to the deadline function used), the solutions found by the heuristic are also optimal solutions to the optimization problem in (17)–(19).

Table 3 shows the running time as well as the number of iterations needed by the proposed heuristic to find a solution for each task set. For reference, the case of the fixed deadline technique required the running time of less than 0.01s in all cases. As can be seen from the table, the task sets that the heuristic could make feasible took less than 0.01s to run with no more than 3 search iterations. On the other hand, 100 search iterations were not enough to find a feasible period-deadline combination for 13 task sets.

6 Conclusions

In this paper, we proposed a more general and realistic real-time task model where each task deadline is a function of the corresponding period. This task model facilitates the feasibility analysis of the real-time control systems where task deadlines reflect the maximum allowable delays as tolerated by any given system and vary according to the sampling periods. Since existing techniques cannot adequately be used to determine schedulability for this

novel task model, we also proposed a heuristic to identify a schedulable period-deadline combination. Our heuristic minimizes the search region and iteratively finds a feasible period-deadline combination. Experimental results show that our method of solving the period and deadline selection problem is much less pessimistic than existing techniques that consider task deadlines to be fixed parameters; our heuristic found a solution to the problem over 73% of the time using less than 3 search iterations and requiring less than 0.02s to run in the worst-case.

As future work, we intend on (i) obtaining more experimental results, particularly using benchmarks derived from real-world applications, and (ii) implementing the proposed heuristic on a real-time operating system such as the S.Ha.R.K. kernel [18].

References

[1] K. Albers and F. Slomka. Efficient feasibility analysis for real-time systems with edf scheduling. In *Proc. Design, Automation and Test in Europe*, pages 492–497, 2005.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. In *Proc. Real-Time Systems Symposium*, pages 79–89, 1999.

[3] J. Baillieul and P. A. (editors). Special issue on networked control systems. *Transactions on Automatic Control*, 49(9):1421–1423, Sept. 2004.

[4] P. Balbastre, I. Ripoll, and A. Crespo. Minimum deadline calculation for periodic real-time tasks in dynamic priority systems. *Transactions on Computers*, 57(1):96–109, Jan. 2008.

[5] S. Baruah, L. Rosier, and R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2(4):301–324, Nov. 1990.

[6] G. Bernat, A. Burns, and A. Llamosí. Weakly hard real-time systems. *Transactions on Computers*, 50(4):308–321, Apr. 2001.

[7] E. Bini and M. D. Natale. Optimal task rate selection in fixed priority systems. In *Proc. Real-Time Systems Symposium*, pages 399–409, 2005.

[8] G. Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real-Time Systems*, 23(1–2):7–24, July 2002.

[9] G. Buttazzo and L. Abeni. Smooth rate adaptation through impedance control. In *Proc. Euromicro Conf. on Real-Time Systems*, pages 3–10, 2002.

[10] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proc. Real-Time Systems Symposium*, pages 286–295, 1998.

[11] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *Transactions on Computers*, 51(3):289–302, Mar. 2002.

[12] M. Caccamo, G. Buttazzo, and L. Sha. Elastic feedback control. In *Proc. Euromicro Conf. on Real-Time Systems*, pages 121–128, 2000.

[13] M. Caccamo, G. Buttazzo, and L. Sha. Handling execution overruns in hard real-time control systems. *Transactions on Computers*, 51(7):835–849, July 2002.

[14] T. Chantem, X. Hu, and M. Lemmon. Generalized elastic scheduling. In *Proc. Real-Time Systems Symposium*, pages 236–245, 2006.

[15] J.-Y. Chung, J. W. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *Transactions on Computers*, 39(9):1156–1175, Sept. 1990.

[16] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proc. Euromicro Conf. on Real-Time Systems*, pages 23–30, 2003.

[17] N. Fisher and S. Baruah. A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proc. International Conference on Real-Time Systems*, pages 233–249, 2005.

[18] P. Gai, L. Abeni, M. Giorgi, and G. Buttazzo. A new kernel approach for modular real-time systems development. In *Proc. Euromicro Conf. on Real-Time Systems*, pages 199–208, 2001.

[19] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *Transactions on Computers*, 44(12):1443–1451, 1995.

[20] G. Koren and D. Shasha. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proc. Real-Time Systems Symposium*, pages 110–117, 1995.

[21] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu. Hybrid supervisory utilization control of real-time systems. In *Proc. Real-Time & Embedded Technology and Applications Symposium*, pages 12–21, 2005.

[22] T.-W. Kuo and A. Mok. Load adjustment in adaptive real-time systems. In *Proc. Real-Time Systems Sympsium*, pages 160–171, 1991.

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

[24] J. W. S. Liu. *Real-Time Systems*. Prentice-Hall, NJ, 2000.

[25] A. Masrur, S. Drossler, and G. Farber. Improvements in polynomial-time feasibility testing for edf. In *Proc. Design, Automation and Test in Europe*, pages 1033–1038, 2008.

[26] D. Seto, J. Lehoczky, and L. Sha. Task period selection and schedulability in real-time systems. In *Proc. Real-Time Systems Symposium*, pages 188–199, 1998.

[27] D. Seto, J. Lehoczky, L. Sha, and K. Shin. On task schedulability in real-time control systems. In *Proc. Real-Time Systems Symposium*, pages 13–21, 1996.

[28] C.-S. Shih and J. W. Liu. State-dependent deadline scheduling. In *Proc. Real-Time Systems Symposium*, pages 3–14, 2002.

[29] X. Wang and M. Lemmon. Self-triggered feedback control systems with finite-gain l2 stability. *Accepted to Transactions on Automatic Control*, 2007.

[30] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort packet streams. In *Proc. Real-Time Systems Symposium*, pages 239–248, 2000.