Generalized Elastic Scheduling for Real-Time Tasks

Thidapat Chantem, Xiaobo Sharon Hu, and M.D. Lemmon

***Abstract*— The elastic task model is a powerful model for adapting periodic real-time systems in the presence of uncertainty. This paper generalizes the existing elastic scheduling approach in several directions. It reveals that the original task compression algorithm in fact solves a quadratic programming problem that seeks to minimize the sum of the squared deviation of a task's utilization from initial desired utilization. This finding indicates that the task compression algorithm may be applied to efficiently solve other similar types of problems that often arise in real-time applications. In particular, an iterative approach is proposed to solve the task compression problem for real-time tasks with deadlines less than respective periods. Furthermore, the framework is generalized to adjust task deadlines instead of task periods.**

***Index Terms*— Real-time and embedded systems, Sequencing and scheduling, Performance of systems**

I. Introduction

A Desirable property of any real-time system is the guarantee that it will perform at least beyond some pre-specified thresholds defined by system designers. This is usually not a concern under normal situations where analysis has been done offline to ensure system performance based on the regular workload. However, in response to an event such as user's input or changing environment, the load of the system may dynamically change in such a way that a temporal overload condition occurs. The challenge, then, is to provide some mechanism to guarantee the minimum performance level under such circumstances.

Many periodic real-time task models have been proposed to extend timing requirements beyond the hard and soft deadlines based on the observation that jobs can be dropped without severely affecting performance ([5], [19]). For example, Ramanathan et al. proposed both online [17] and offline [27] scheduling algorithms that are based on the (m,k) model, which is analyzed in [18]. In this model, up to $k-m$ consecutive jobs are allowed to be dropped in any sliding window of k. Moreover, [32] presented the Dynamic Window-Constrained Scheduling (DWCS) algorithm, which is similar except that the window k is fixed. Due to their success, these scheduling algorithms have further been enhanced. For example, [26] proposed a pattern rotation scheme to improve schedulability by avoiding some critical instants. Mok at. al. modified DWCS, which is primarily deadline-based, by incorporating the concept of pfairness [3] to improve the success rate for tasks with unit-size execution time [25]. Other frameworks such as the imprecise computation model [14] and reward-based model [1] can be used to capture situations where the quality of service is proportional to the amount of workload completed.

T. Chantem and X.S. Hu are with the department of Computer Science & Engineering and M.D. Lemmon is with the department of Electrical Engineering at the University of Notre Dame, Notre Dame, IN 46556. Emails: {tchantem, shu, lemmon}@nd.edu.

This work is supported in part by NSF under grant number CNS-0410771.

Despite the success of the abovementioned models in alleviating overload situations, it is sometimes more suitable to execute jobs less often instead of dropping them or allocating fewer cycles. For example, limitations on the throughput capacity of ad hoc communication networks [2] make it highly desirable to reduce overall network traffic by having control tasks adaptively adjust their periods in response to the actual activity level of the control application.

The work in [21] was among the first to address this type of requirements. Seto, et al. considered the problem of finding a feasible set of task periods as a non-linear programming problem which seeks to optimize a specific form of control performance measure [28]. In [29], finding all feasible periods of a given set of tasks was studied for the Rate Monotone (RM) scheduling algorithm. Cervin et al. used optimization theory to solve the period selection problem online by adaptively adjusting task periods while optimizing a specific form of control performance [12]. Recently, [6] offered an optimal search algorithm that solves the period selection problem for fixed-priority scheduling schemes. The algorithm may be applicable only during the design phase due to its potentially high complexity. Another interesting framework was introduced in [20] where task periods are adjusted in response to varying computation times.

Buttazzo and his colleagues proposed an elegant and flexible framework known as the elastic task model [9] where deadline misses are avoided by increasing tasks periods. The work in [11] extends the basic elastic task model to handle cases where the computation time is unknown, [10] incorporated a mechanism to handle resource constraints within the elastic framework, and [8] provided a means to smoothly adjust task execution rate. In addition, [15] uses a control performance metric as a cost function to find an optimal sampling interval for each task.

We will focus our attention on the elastic task model where the selection of task periods is central. The existing elastic scheduling algorithm determines task periods based on an elegant analogy between spring systems and task scheduling in which a task's resistance to changing its period is viewed as a spring's resistance to being compressed. In accordance with the *principle of least action* found in classical mechanics, this suggests that the elastic task model is really attempting to minimize some overall measure of task set's *energy*, whose precise nature was not made clear in the original work.

Based on our previous findings in [13], this paper generalizes the existing elastic scheduling approach in several directions. First, we re-examine the problem of period determination in the elastic task model and show that the task compression algorithm in [10] in fact solves a quadratic programming (QP) problem. The QP problem seeks to minimize the sum of the squared deviation of every task's utilization from its initial utilization. Identifying the nature of the optimization problem underlying the task compression algorithm is important in several aspects. For instance, it may suggest other relevant optimization objectives and shed light on determining task periods in the presence of uncertainty for other task models.

Second, the proposed framework is extended to cases where task deadlines are less than task periods. Such task specifications often arise in control systems to reduce jitter, for example. Moreover, in some situations, it is desirable that tasks finish executing sooner, even if their periods are not up. We formulate the problem of period determination as a constrained optimization problem and propose a heuristic approach based on the task compression algorithm in [10] to solve the problem. The heuristic is guaranteed to find a feasible solution, if one exists. It is quite efficient and is hence suitable for online period adjustment. Experimental results show that the heuristic actually finds the global optimal solution in many cases.

Finally, the proposed framework is generalized to solve the deadline selection problem whose objective is to find a set of task deadlines such that the task set is feasible. Solving this problem is useful for systems where the tasks have pre-specified periods but some delays in task completions are tolerable.

The remainder of the paper is organized as follows. We begin by reviewing key background materials in Section II. Section III presents the solutions to the period determination problem for tasks with deadlines equal to periods. The optimization approach is then extended in Section IV to treat the case where task deadlines are less than task periods. In Section V, we demonstrate how the proposed framework can be used to solve the problem of deadline selection. Experimental results are presented and discussed in Section VI. Finally, the paper concludes with Section VII.

II. Preliminaries

This section describes the system under consideration as well as important assumptions pertaining to our task model. We also briefly review the task compression algorithm used for period selection [10].

A. Periodic task model

We consider the system where each task τ_i is periodic and is characterized by the following 6-tuple: $(C_i, D_i, T_i, T_{i0}, T_{i_{max}}, e_i)$, for $i = 1, \ldots, N$, where N is the number of tasks in the system, C_i is the worst case execution time of τ_i, D_i is its deadline, and T_i is τ_i's actual period. Furthermore, T_{i0} 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 elastic coefficient, e_i, represents the resistance of task τ_i to increasing its period in face of changes. The smaller the elastic coefficient of a task, the harder it is to increase that task's period. Given a task set Γ, tasks are arranged in a non-decreasing order of deadlines and all start at time 0.

Task deadlines are first assumed to be equal to task periods. This requirement will be relaxed in Section IV; in that section, we treat the case where task deadlines are less than task periods. All task attributes are real values and are assumed to be known *a priori*. The current utilization of τ_i is $U_i = \frac{C_i}{T_i}$. Similarly, the minimum and the desired utilizations of τ_i is $U_{i_{min}} = \frac{C_i}{T_{i_{max}}}$ and $U_{i0} = \frac{C_i}{T_{i0}}$, respectively.

B. Elastic task model

In [10], Buttazzo, et. al. modeled a task system as a spring system, where increasing or decreasing a task period is analogous to compressing or decompressing a spring. The elastic coefficient, e_i, introduced above hence has its intuitive meaning of the hardness of the spring. The purpose of increasing task periods is to drive the total utilization of the system down to some desired utilization level, U_d, analogous to a spring system trying to minimize its energy under an external force.

The attractiveness of the elastic task model is its accompanying task compression algorithm, which is quite efficient $\big(O(N^2)\big)$ and can readily be used online. (In fact, the elastic task model and the task compression algorithm have already been implemented in the S.Ha.R.K. kernel [16].) The task compression algorithm works as follows. If it is possible to drive the system utilization down to U_d without violating any period bounds, the algorithm will return a set of feasible periods $(T_1, T_2, \ldots, T_N)$ that can be used by the system. Tasks whose periods are fixed (if $e_i = 0$ or $T_{i0} = T_{i_{max}}$) are considered inelastic and are treated as special cases. The amount of utilization that each remaining, non-inelastic task should receive is computed based on its elastic coefficient, initial period, and the amount of utilization that must be reduced to achieve U_d. The resultant period of a task τ_i is guaranteed to fall somewhere between T_{i0} and $T_{i_{max}}$. For completeness, the task compression algorithm is reproduced in Algorithm 1.

Algorithm 1 Task_Compress(Γ, U_d)

```
1:  $U_0 = \sum_{i=1}^{n} C_i/T_{i0}$
2:  $U_{\min} = \sum_{i=1}^{n} C_i/T_{i_{\max}}$
3:  if $(U_d < U_{\min})$ then
4:      return INFEASIBLE
5:  end if
6:  repeat
7:      $U_f = E_v = 0$
8:      for $(each\ \tau_i)$ do
9:          if $((e_i == 0)\,\mathbf{or}\,(T_i == T_{i_{\max}}))$ then
10:             $U_f = U_f + U_i$
11:         else
12:             $E_v = E_v + e_i$
13:             $U_{v0} = U_0 - U_f$
14:         end if
15:     end for
16:     $ok = 1$
17:     for $(each\ \tau_i \in \Gamma_v)$ do
18:         if $((e_i > 0)\,\mathbf{and}\,(T_i < T_{i_{\max}}))$ then
19:             $U_i = U_{i0} - \big(U_{v0} - U_d + U_f\big)\, e_i/E_v$
20:             $T_i = C_i/U_i$
21:             if $(T_i > T_{i_{\max}})$ then
22:                 $T_i = T_{i_{\max}}$
23:                 $ok = 0$
24:             end if
25:         end if
26:     end for
27: until $(ok == 0)$
28: return FEASIBLE
```

Throughout this paper, we will assume that the Earliest Deadline First (EDF) scheduling algorithm [22] is used. Furthermore, we will focus our attention on cases where tasks need to decrease their utilization 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.

III. Period Selection for the Basic Task Model

In this section, we focus on the basic periodic task model where $D_i = T_i$ for task τ_i, $i = 1, \ldots, N$. During an overload, task periods may be adjusted in such a way that the task set becomes schedulable. Given a particular set of real-time tasks, there may exist numerous sets of feasible periods. It is not difficult to see that different sets of periods would lead to different performance of the resultant system. In general, the period selection problem can be expressed as an optimization problem. That is,

```
optimize: performance metric
s.t.:     tasks are schedulable
          period bounds are satisfied
```

Below we introduce a specific performance metric and discuss its implications. We assume that task deadlines equal task periods.

Processor utilization by each task is an important measure for any real-time system. It not only reveals the amount of system resource dedicated to the task but also impacts schedulability. In the elastic task model, one consequence of changing task periods is changing the utilization of tasks. From the stand point of performance preservation, it is desirable to minimize the changes in task utilization. This objective can be captured by the following constrained optimization problem.

$$\text{min:} \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \tag{1}$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} U_i \le U_d \tag{2}$$

$$U_i \ge U_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{3}$$

$$U_i \le U_{i0} \quad \text{for } i = 1, 2, \cdots, N \tag{4}$$

In the formulation, N is the number of tasks in the system, U_{i0} is the desired utilization of task τ_i and $U_{i0} \ge U_{i_{min}}$, U_i is the utilization of τ_i to be determined, and U_d is the desired total utilization. (U_d is usually set to 1 for EDF scheduling.) Constant w_i (≥ 0) is a weighting factor and reflects the criticality of a task. More critical tasks would have larger w_i's. The first constraint simply states the schedulability condition under EDF. The rest of the constraints bounds the utilization, equivalently bounds the task period by T_{i0} and $T_{i_{max}}$ where $T_{i0} = C_i/U_{i0}$ and $T_{i_{max}} = C_i/U_{i_{min}}$.

It is worth noting that for $w_i = 0$, (1) does not change regardless of what U_i value is used. To help satisfying (2), it is natural to simply use $U_i = U_{i_{min}}$. Hence, for the rest of the paper, we will focus on the case where $w_i > 0$ for all $1 \le i \le N$.

The problem in (1)–(4) belongs to the category of quadratic programs and can be solved in polynomial time. However, solving such a problem using a quadratic program solver (such as `Loqo` [30]) during runtime can be too costly. What makes the above formulation attractive is that its solution is exactly the same as that found by the task compression algorithm in [10]. We introduce a lemma and a theorem to support this argument.

Lemma 1: Given the constrained optimization problem as specified in (1)–(4) and $\sum_{i=1}^{N} U_{i0} > U_d$, any solution, U_i^*, to the problem must satisfy $\sum_{i=1}^{N} U_i^* = U_d$ and $U_i^* \ne U_{i0}$, for $i = 1, \ldots, N$.

Proof: We prove the lemma by utilizing the Karush-Kuhn-Tucker (KKT) necessary conditions for the solution to the given problem, which can be written in terms of the Lagrangian function for the problem as

$$\begin{aligned} J_a(\mathbf{U}, \mu) \ =\ & \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 + \mu_0 \left(\sum_{i=1}^{N} U_i - U_d\right) + \\ & \sum_{i=1}^{N} \mu_i \left(U_{i_{min}} - U_i\right) + \sum_{i=1}^{N} \lambda_i \left(U_i - U_{i0}\right) \end{aligned} \tag{5}$$

where μ_0, μ_i, and λ_i are Lagrange multipliers, $\mu_0 \ge 0$, $\mu_i \ge 0$, and $\lambda_i \ge 0$, for $i = 1, \ldots, N$. The necessary conditions for the existence of a relative minimum at U_i^* are, for all $i = 1, \ldots, N$,

$$0 = \frac{\partial J_a}{\partial U_i^*} = -2w_i \left(U_{i0} - U_i^*\right) + \mu_0 - \mu_i + \lambda_i \tag{6}$$

$$0 = \mu_0 \left(\sum_{i=1}^{N} U_i^* - U_d\right) \tag{7}$$

$$0 = \mu_i \left(U_{i_{min}} - U_i^*\right) \tag{8}$$

$$0 = \lambda_i \left(U_i^* - U_{i0}\right) \tag{9}$$

Assume that (2) is inactive, i.e., $\mu_0 = 0$ and $\sum_{i=1}^{N} U_i^* < U_d$. Then at least one constraint in (3) or (4) must be active. Suppose the k-th constraint in (3) is active. That is, $U_k^* = U_{k_{min}}$ and $\mu_k \ge 0$. Then, the k-th constraint in (4) must be inactive, i.e., $\lambda_k = 0$. From (6), we obtain

$$\mu_k = -2w_k(U_{k0} - U_{k_{min}}) < 0 \tag{10}$$

which contradicts the assumption that $\mu_k \ge 0$. Therefore, if any $U_k^* = U_{k_{min}}$, constraint (2) must be active.

Now assume that some constraint in (4) is active while others are inactive. Suppose $U_h^* = U_{h0}$ (active) and $U_{k_{min}} < U_k < U_{k0}$ (inactive). Then $\mu_h = 0, \lambda_h \ge 0$, and $\mu_k = \lambda_k = 0$. From (6), we have

$$\mu_0 = 2w_h(U_{h0} - U_h^*) + \mu_h - \lambda_h = -\lambda_h \tag{11}$$

$$\mu_0 = 2w_k(U_{k0} - U_k^*) \tag{12}$$

Note that (11) and (12) cannot be simultaneously satisfied. Therefore, we can either have all the constraints in (4) be active or all are inactive. If all the constraints in (4) are active, we have

$$\sum_{i=1}^{N} U_i^* = \sum_{i=1}^{N} U_{i0} > U_d \tag{13}$$

which contradicts the assumption that the resultant task set is schedulable. If all the constraints in (4) are inactive, (12) requires that $\mu_0 > 0$, which contradicts the assumption that constraint (2) is inactive. Therefore, for any solution to the optimization problem, constraint (2) must be active, i.e., $\sum_{i=1}^{N} U_i^* = U_d$ and $U_i^* \ne U_{i0}$. ■

Theorem 1: Given the constrained optimization problem as specified in (1)–(4), $\sum_{i=1}^{N} U_{i0} > U_d$, and $\sum_{i=1}^{N} U_{i_{min}} \le U_d$, let $\widehat{U} = \sum_{U_i^* \ne U_{i_{min}}} U_{i0} + \sum_{U_i^* = U_{i_{min}}} U_{i_{min}}$. A solution to the problem, U_i^*, is optimal if and only if

$$U_i^* = U_{i0} - \frac{\frac{1}{w_i}\left(\widehat{U} - U_d\right)}{\sum_{U_j^* \ne U_{j_{min}}} (1/w_j)} \tag{14}$$

for $\widehat{U} > U_d$ and $U_i^* > U_{i_{min}}$, and $U_i^* = U_{i_{min}}$ otherwise.

Proof: Consider the KKT conditions given in (6)–(9). From Lemma 1, we know that any solution to the given optimization problem must satisfy (2), i.e., $U_d = \sum_{i=1}^{N} U_i^*$, and $U_i^* \neq U_{i0}$. Hence, we only need to consider the case where $\lambda_i = 0$, for all $i = 1, \ldots, N$. Suppose that the k-th constraint in (3) is active. We have $U_k^* = U_{k_{min}}$, and

$$\mu_k = \mu_0 - 2w_k \left(U_{k0} - U_{k_{min}}\right), \tag{15}$$

Otherwise, we have $\mu_k = 0$. By summing up (6) for all i and using the conclusions above,

$$\mu_0 = \frac{2\left(\widehat{U} - U_d\right)}{\sum_{U_i^* \neq U_{i_{min}}} (1/w_i)}. \tag{16}$$

As long as $\widehat{U} > U_d$, $\mu_0 > 0$, $\mu_i \geq 0$, and constraints in (4) are satisfied. Therefore, a solution, U_i^*, to the optimization problem either satisfies $U_i^* = U_{i_{min}}$ or can be obtained by combining (16) with (6) for $U_i^* > U_{i_{min}}$. (Note that $\mu_i = 0$ when $U_i^* > U_{i_{min}}$.) That is,

$$U_i^* = U_{i0} - \frac{\frac{1}{w_i}\left(\widehat{U} - U_d\right)}{\sum_{U_j^* \neq U_{j_{min}}} (1/w_j)} \tag{17}$$

Additionally, since the objective function and the inequality constraints in (1)–(4) are convex, the necessary conditions for optimality provided by the KKT conditions also become the sufficient conditions for optimality [24]. Hence, the solution found in Theorem 1 is a global minimum. ■

Corollary 1: Consider a set of N tasks where U_i is the utilization of the ith task. Let U_{i0} denote the initial desired utilization of task τ_i and let $e_i = \frac{1}{w_i} > 0$ be a set of elastic coefficients for $i = 1, \ldots, N$. Let U_d be the desired utilization level and $\sum_{i=1}^{N} U_{i0} > U_d$. The task utilizations U_i, for $i = 1, \ldots, N$ obtained from the task compression algorithm in [10] minimize

$$E(U_1, \ldots, U_N) = \sum_{i=1}^{N} \frac{1}{e_i}(U_{i0} - U_i)^2$$

subject to the inequality constraints $\sum_{i=1}^{N} U_i \leq U_d$, $U_i \geq U_{i_{min}}$, and $U_i \leq U_{i0}$, for $i = 1, \ldots, N$.

The above corollary has several significant consequences. First, it reveals the optimization criterion inherent in the task compression algorithm. Second, it provides guidance on the selection of other performance measures . Third, the task compression algorithm may be modified and/or extended to solve similar convex programming problems. An example of this extension will be described in the next section.

IV. Period Selection with Additional Deadline Constraints

In this section, we consider the case where task deadlines are less than task periods. This more general model is useful in situations where it is desirable for a task to finish executing early (before its period ends). By using the optimization framework introduced in Section III, we again formulate the period selection problem as a constrained optimization problem and propose a novel heuristic based on the task compression algorithm. The algorithm is guaranteed to find a solution to the problem, if one exists, and is efficient enough for online use.

A. Simplify feasibility condition

Baruah et al. considered the case where task deadlines are less than or equal to task periods and derived a sufficient and necessary condition for EDF schedulability [4], which is later improved in [7]. The condition is restated in the following theorem.

Theorem 2: **[4]** Given a periodic task set with $D_i \leq T_i$, 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,

$$L \geq \sum_{i=1}^{N} \left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1\right) C_i \tag{18}$$

Based on Theorem 2, the period determination problem can be formulated as follows:

$$\text{min:} \quad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \tag{19}$$

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

$$L \in \{kT_i + D_i \leq \min(B_p, H)\}, k \in \mathcal{N} \tag{20}$$

$$U_i \geq U_{i_{min}} \quad \text{for } i = 1, 2, \cdots, N \tag{21}$$

$$U_i \leq U_{i0} \quad \text{for } i = 1, 2, \cdots, N \tag{22}$$

Solving the above constrained optimization problem can be extremely time consuming. Hence, we investigate solving the problem approximately with an efficient algorithm below. An approximate solution is both acceptable and preferred, as a rapid response allows the system to degrade gracefully instead of entering into potentially catastrophic states due to some dynamic perturbations.

Since verifying the constraint in (18) for all L values is the main source of high complexity, we consider simplifying the schedulability test by using the following stronger schedulability condition,

$$L \geq \sum_{i=1}^{N} \left(\frac{L - D_i}{T_i} + 1\right) C_i \tag{23}$$

It is not difficult to see that if the inequality in (23) is satisfied then the original inequality in (18) must also be satisfied. What makes (23) an excellent candidate for online use is that the schedulability of a task set can be determined based on a single L value, L^*. Below, we introduce several lemmas and a theorem to support this claim.

For simplicity, we denote the set of all possible values of L by a distinct ordered set $\mathcal{L} = \{L_0, L_1, \ldots\}$ where $L_j = kT_i + D_i, k \in \mathcal{N}$ and $L_j \leq \min(B_p, H)$.

Lemma 2: Given a set Γ of N tasks with $D_i \leq T_i$, let L_j and $L_{j+1} \in \mathcal{L}$ and let $L_j < L_{j+1}$. If the constraint in (23) is satisfied for L_j, then it is satisfied for L_{j+1}.

Proof: By regrouping the terms in (23), we can rewrite the inequality as follows.

$$L \geq \frac{\sum_{i=1}^{N} C_i - \sum_{i=1}^{N} U_i D_i}{1 - \sum_{i=1}^{N} U_i} \tag{24}$$

Given that L_j satisfies the constraint in (24) and $L_{j+1} > L_j$, it immediately follows that L_{j+1} satisfies the constraint in (24). ■

Based on the above lemma, we can conclude that if the constraint in (23) is satisfied for L_j, then it is also satisfied for all $L_k \in \mathcal{L}$, where $L_k > L_j$. It may then seem natural to simply set L^* to be the minimum of all L values in $\mathcal{L}$. However, such a choice can be extremely pessimistic, often resulting in finding no feasible solutions to the problem. To avoid being too pessimistic, we introduce the next lemma, which identifies useful necessary conditions for any feasible task set. The lemma helps to eliminate pessimistic choices of L^*.

Lemma 3: Let D_i be the deadline of task τ_i in a given task set Γ, $i = 1, \ldots, N$. Assume that $\tau_i \in \Gamma$ starts at time 0. Further, let the tasks in Γ be ordered in a non-decreasing order of deadlines and suppose that D_{min} is unique. Regardless of the choices of periods, any task set that is schedulable must satisfy the following property:

$$\sum_{i=1}^{j} C_i \quad \leq \quad D_j, \forall j = 1, \ldots, N \tag{25}$$

Proof: We prove Lemma 3 by contradiction. Suppose the task set is schedulable and $\sum_{i=1}^{j} C_i > D_j$ for some j. By time D_j, at least one instance of $\tau_1, \ldots, \tau_j$ must each finish executing. Thus, The total processor demand at D_j is at least $\sum_{i=1}^{j} C_i$ time units. However, since there are only D_j time units available and $\sum_{i=1}^{j} C_i > D_j$, at least one instance of $\tau_1, \ldots, \tau_j$ has to miss its deadline. This contradicts the assumption that the task set is schedulable. Hence, (25) must be true for all $j = 1, \ldots, N$. ■

We are now ready to introduce two lemmas which form the basis for our selection of L^*.

Lemma 4: Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadlines. If $D_1 + T_1 \leq D_2$, and $L^* = D_2$ satisfies the inequality constraint in (23), then the task set is guaranteed to be schedulable.

Proof: Let $L_h = L^* = D_2$. By Lemma 2, any $L_j \in \mathcal{L}$ with $j > h$ satisfies constraint in (23) and hence satisfies (18). Now consider $j < h$. Since $D_i \geq D_2$ for $i > 2$, L_j can only be equal to $D_1 + kT_1$ (since the task set must satisfy Lemma 3) for some $k \in \mathcal{N}$. In order for L_j to satisfy (18), noting that $|D_1 + kT_1 - D_i| < T_i$ for $i \geq 2$, we need $D_1 + kT_1 \geq (k+1) \cdot C_1$, which holds true according to Lemma 3. Therefore, for all values of $L \in \mathcal{L}$, (18) is satisfied. ■

Lemma 5: Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadlines. If $D_1 + T_1 > D_2$, and $L^* = \min_{i=1}^{N}(T_i + D_i)$ satisfies the inequality constraint in (23), then the task set is guaranteed to be schedulable.

Proof: Let $L_h = L^* = min_{i=1}^{N}(T_i + D_i)$. By Lemma 2, any $L_j \in \mathcal{L}$ with $j > h$ satisfies constraint in (23) and hence satisfies (18). Now consider $j < h$. L_j can only be equal to D_k, $k = 1, \ldots, N$, such that $D_k < L_h$. In order for L_j to satisfy (18), we need $\sum_{i=1}^{j} C_i \leq D_j$, which holds true according to Lemma 3. Therefore, for all values of $L \in \mathcal{L}$, (18) is satisfied. ■

From Lemmas 3–5, we can conclude that given an arbitrary task set Γ of N tasks, a maximum number of $N + 1$ checks need to be performed to test the schedulability of that task set. Namely, at most N checks must be performed to determine whether Γ satisfies Lemma 3 and one check must be performed to determine whether either Lemmas 4 or 5 is satisfied. We collect these conclusions in the following theorem.

Theorem 3: Consider a set Γ of N tasks that satisfy the condition in Lemma 3. Let the tasks in Γ be sorted in a non-decreasing order of deadlines. A given task set is schedulable if

$$L^* \quad \geq \quad \sum_{i=1}^{N} \left(\frac{L^* - D_i}{T_i} + 1 \right) C_i \tag{26}$$

where

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

Proof: From Lemmas 4 and 5, we know that the only L that needs to be checked against the constraint in (23) is D_2 if $D_1 + kT_1 \leq D_2$, $k = 0, 1, \ldots$, and $\min_{i=1}^{N}(T_i + D_i)$ otherwise. Moreover, we have proved in Lemma 2 that if (23) is satisfied for some L_j then it is also satisfied for L_{j+1} where $L_j < L_{j+1}$. This, in turns, implies that if (23) is satisfied for L_j then it is also satisfied for all $L_k \in L$, where $L_k > L_j$. Taken together, if the constraint in (23) is satisfied for L^* then the task set is schedulable. ■

The above theorem paves the way to finding a simpler constrained optimization problem for the purpose of period determination. We present the actual problem formulation in the following subsection.

B. Minimize utilization perturbation with deadline constraints

By using Theorem 3, we can express the period selection problem where task deadlines are less than task periods as a constrained optimization problem similar to that in (1)–(4). Letting $r_i = L - D_i$, (23) can be rewritten as

$$\sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i. \tag{27}$$

Then the period determination problem where task deadlines are less than task periods can be formulated as

$$\text{min:} \qquad E(U_1, \cdots, U_N) = \sum_{i=1}^{N} w_i \left(U_{i0} - U_i\right)^2 \tag{28}$$

$$\text{s.t.:} \qquad \sum_{i=1}^{N} r_i U_i \leq L - \sum_{i=1}^{N} C_i \tag{29}$$

$$L = \begin{cases} D_2 & : \; D_1 + \frac{C_1}{U_1} \leq D_2 \\ \min(\frac{C_i}{U_i} + D_i) & : \; otherwise \end{cases} \tag{30}$$

$$U_i \geq U_{i_{min}} \qquad \text{for } i = 1, 2, \cdots, N \tag{31}$$

$$U_i \leq U_{i0} \qquad \text{for } i = 1, 2, \cdots, N \tag{32}$$

Note that the above constrained optimization problem would have exactly the same format as the QP problem in (1)–(4) if L and r_i can be treated as constants. Unfortunately, this is not the case, as the actual value of L is dependent on variable U_i. Consequently, the above optimization problem must be treated as a nonlinear program which can be too costly to solve in terms of both processor time and memory usage.

The challenge, then, is to solve the problem efficiently so as to allow the system to respond to dynamic changes in a timely manner. We propose using an iterative heuristic based on the task compression algorithm to tackle the challenge. The main idea of our heuristic is as follows. Suppose at iteration h, a set of periods $T_i(h)$ is found by solving the optimization problem in (28)–(32). In iteration $h+1$, we compute the value of L based on Theorem 3 using $T_i(h)$. A check is then performed to see whether the constraint in (23) is satisfied. If this is the case and if $T_i(h)$

also minimizes the objective function till now, then the algorithm keeps $T_i(h)$ as the current best solution to the problem. Otherwise, if the constraint in (23) is not satisfied, we modify the periods found in iteration h in some manner and use them as the periods found in iteration $h+1$. This process is repeated in an attempt to find the best set of periods that the heuristic can offer. Below, we introduce a lemma and a couple of theorems to show how the optimization problem in (28)–(32) can be optimally solved for a fixed value of L. We then give the details of our heuristic algorithm and discuss the solution quality.

Lemma 6: Given the constrained optimization problem as specified in (28)–(32) and $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$, any solution, U_i^*, to the problem must satisfy $\sum_{i=1}^{N} r_i U_i = L - \sum_{i=1}^{N} C_i$.

We skip the proof for Lemma 6 since it can be proved using the same technique as in Lemma 1. As Lemma 6 states that the constraint in (29) must be active, we consider solving the optimization problem in (28)–(32) when $D_1 + \frac{C_1}{U_1} \leq D_2$. According to Lemma 4, we only need to check $L^* = D_2$ for schedulability, which indeed leads to a constant L value in (29). It follows that we can solve the optimization problem efficiently by using the following theorem.

Theorem 4: Given the constrained optimization problem as specified in (28)–(32), for $L = D_2$, $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$, and $U_{1_{min}} \leq U_1^* < U_{10}$, a solution, U_i^*, is optimal if and only if

$$U_i^* = \begin{cases} \frac{D_2 - \sum_{j=1}^{N} C_j - \sum_{j=3}^{N} r_j U_{j0}}{D_2 - D_1} & : \quad i = 1 \\ U_{i0} & : \quad otherwise \end{cases}$$

for $D_2 > \sum_{j=1}^{N} C_j + \sum_{j=3}^{N} r_j U_{j0}$.

Proof: Let $L_d = L - \sum_{i=1}^{N} C_i$. The KKT conditions for the solution to the optimization problem in (28)–(32) can be written as follows:

$$0 = -2w_i \left(U_{i0} - U_i^*\right) + r_i \mu_0 - \mu_i + \lambda_i \tag{33}$$

$$0 = \mu_0 \left(\sum_{j=1}^{N} r_j U_j^* - L_d \right) \tag{34}$$

$$0 = \mu_i \left(U_{i_{min}} - U_i^*\right) \tag{35}$$

$$0 = \lambda_i \left(U_i^* - U_{i0}\right) \tag{36}$$

for $i = 1, \cdots, N$, where μ_0, μ_i's and λ_i's are Lagrange multipliers, $\mu_0 \geq 0$, $\mu_i \geq 0$, and $\lambda_i \geq 0$ for $i = 1, \cdots, N$.

Consider first those tasks with $D_k = D_2$. Then $r_k = L - D_k = 0$. Now (33) reduces to

$$\mu_k - \lambda_k = -2w_k \left(U_{k0} - U_k^*\right) \tag{37}$$

Assume that $U_{k_{min}} < U_k^* < U_{k0}$. In order to satisfy (35) and (36), we must have $\mu_k = \lambda_k = 0$, which contradicts (37). Now assume that $U_k^* = U_{k_{min}}$. Then to satisfy (36), we need $\lambda_k = 0$. However, this leads to $\mu_k < 0$ from (37), which violates the KKT conditions. Therefore, for those tasks with $D_k = D_2$, $U_k^* = U_{k0}$. (It can be readily proved that such a solution indeed satisfies the KKT conditions.)

Similarly, consider next those tasks with $D_h > D_2$. Then, $r_h = L - D_h < 0$. Now (33) becomes

$$-2w_h \left(U_{h0} - U_h^*\right) = \mu_h - \lambda_h - r_h \mu_0 \tag{38}$$

In order to satisfy (35) and (36), we must have $\mu_h = \lambda_h = 0$, which will cause (38) to become

$$-2w_h (U_{h0} - U_h^*) = D_h - D_2. \tag{39}$$

This is clearly a contradiction, since $D_h > D_2$ and $U_{h0} - U_h^* \geq 0$. Now, assume that $U_h^* = U_{h_{min}}$. Then, to satisfy (36), we need $\lambda_h = 0$. However, this leads to $\mu_h < 0$ in (38), which violates the KKT conditions. Therefore, for any task with $D_h > D_2$, $U_h^* = U_{h0}$.

For $i = 1$, we first note that λ_1 must be equal to 0, since $U_i^* = U_{i0}$, for $i = 2, \ldots, N$, and $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$. By replacing $U_i^* = U_{i0}$ for $i \geq 2$ in (32), we obtain the value of U_1^* exactly as defined in Theorem 4. Moreover, if $U_1^* \geq U_{i_{min}}$ then $\mu_1 \geq 0$. Otherwise, if $U_1^* < U_{i_{min}}$, then the task set is infeasible and $\mu_1 = 0$. In any case, $\mu_1 \geq 0$.

We have shown that the values of U_i^* as defined in Theorem 4 satisfy the KKT conditions and form a feasible solution to the problem under consideration. Since the constrained optimization problem is convex, it follows that this feasible solution is also an optimal one [24]. ■

Theorem 4 immediately leads to an efficient algorithm to solve the optimization problem in (28)–(32) when $D_1 + T_1 \leq D_2$. Let us next consider the case where $D_1 + T_1 > D_2$. According to Lemma 5, one needs to check whether $L^* = \min_{i=1}^{N} (T_i + D_i)$ satisfies (23) to determine feasibility. The following theorem forms the basis for solving the optimization problem in (28)–(32) when $D_1 + T_1 \leq D_2$.

Theorem 5: Given the constrained optimization problem as specified in (28)–(32), for a fixed value of L (where $L = \min_{i=1}^{N}\{\frac{C_i}{U_i} + D_i\}$) and $\sum_{i=1}^{N} r_i U_{i0} > L - \sum_{i=1}^{N} C_i$, let

$$R = \sum_{U_j^* \neq U_{j_{min}}} \frac{r_j^2}{w_j} - \sum_{U_j^* = U_{j0}} \frac{r_j^2}{w_j}$$

$$\overline{V} = \sum_{U_j^* \neq U_{j_{min}}} r_j U_{j0} - (L - \sum_{i=1}^{N} C_j) + \sum_{U_j^* = U_{j_{min}}} r_j U_j.$$

If a solution, U_i^*, is optimal then

$$U_i^* = U_{i0} - \frac{r_i}{w_i R} \cdot \overline{V} \tag{40}$$

for $r_i > 0$ and $0 \leq \frac{\overline{V}}{R} \leq \frac{w_i}{r_i}(U_{i0} - U_{i_{min}})$, and $U_i^* = U_{i0}$ for $r_i \leq 0$.

Proof: According to Lemma 6, the constraint in (29) must be active. In other words, any solution to the given optimization problem must satisfy $L_d = L - \sum_{i=1}^{N} C_i = \sum_{i=1}^{N} r_i U_i$. We consider $r_k \leq 0$ and $r_k > 0$ separately.

Case 1 ($r_k \leq 0$): Consider the KKT conditions given in (33)–(36). Assume that both constraints in (31) and (32) are inactive (i.e., $U_{k0} < U_k^* < U_{k_{min}}$, $\mu_k = \lambda_k = 0$). Then, (33) becomes

$$2w_k (U_{k0} - U_k^*) = r_k \mu_0. \tag{41}$$

However, since $U_k^* < U_{k0}$, $r_k \leq 0$, and $\mu_0 \geq 0$, the above equation cannot hold. Therefore, for any solution U_k^*, either the constraint in (31) or (32) must be active.

Let us first assume that $\mu_k > 0$ but $\lambda_k = 0$. Then (33) gives

$$2w_k (U_{k0} - U_k^*) + \mu_k = r_k \mu_0, \tag{42}$$

which contradicts the assumption that $\mu_k > 0$. Consequently, $U_k^* = U_{k0}$ for $r_k \leq 0$.

Case 2 ($r_k > 0$): Suppose that the k-th constraint in (31) is active. That is, $U_k^* = U_{k_{min}}$, $\mu_k > 0$, and $\lambda_k = 0$. Then, from (33),

$$\mu_k = r_k\mu_0 - 2w_k(U_{k0} - U_{k_{min}}). \tag{43}$$

If the constraint in (31) is inactive, then $\mu_k = 0$. Similarly, if the h-th constraint in (32) is active, then $U_h^* = U_{h0}$, $\lambda_h > 0$, $\mu_h = 0$ and

$$\lambda_h = -r_h\mu_0 \tag{44}$$

and $\lambda_h = 0$ if the constraint in (32) is inactive. Multiplying (33) by r_i, summing it up for all i, and using the conclusions above, we have

$$\mu_0 = \frac{2\overline{V}}{R} \tag{45}$$

By combining (45) with (33), we get

$$U_i^* = U_{i0} - \frac{r_i\overline{V}}{w_iR} \tag{46}$$

To enforce the condition of $U_{i_{min}} \leq U_i^* \leq U_{i0}$, $\frac{\overline{V}}{R}$ must satisfy $\frac{w_i}{r_i}\left(U_{i0} - U_{i_{min}}\right)$. Summarizing Case 1 and Case 2, we have that a solution to the optimization problem either satisfies $U_i^* = U_{i0}$ for $r_i \leq 0$ or $U_i^* = U_{i0} - \frac{r_i\overline{V}}{w_iR}$, for $r_i > 0$ and $0 \leq \frac{\overline{V}}{R} \leq \frac{w_i}{r_i}\left(U_{i0} - U_{i_{min}}\right)$. ■

Theorems 4 and 5 show how an optimal set of task periods can be determined given a fixed value of L. We now explain our heuristic in more details.

C. Our heuristic

A summary of the heuristic is given in Algorithms 2 and 3. Algorithm 2 shows the main procedure and Algorithm 3 is called by Algorithm 2 to perform a specific set of functions as will be explained below.

We will first describe the main procedure (Algorithm 2). In each iteration h, we fix the value of L as either $L(h) = D_2$ if $T_1(h-1)+D_1 \leq D_2$ or $L(h) = \min_{i=1}^{N}(T_i(h-1)+D_i)$ otherwise (Lines 14–18). For any task τ_i whose $r_i = L(h) - D_i \leq 0$, its period is immediately set to T_{i0} (Lines 32–37). For any task τ_i whose $r_i > 0$, its utilization, $U_i(h)$, can be determined using Theorem 4 (Line 26) or Theorem 5 (Line 39), respectively. If $L(h) = D_2$ and $h = 0$, our heuristic will only require one iteration to find an optimal solution and exit immediately, since the solution set will remain unchanged in subsequent iterations. In the case of $L(h) = \min_{i=1}^{N}(T_i(h-1) + D_i)$, U_i is obtained by using a slightly modified task compression algorithm **Mod_Task_Compress()** (Line 39). The following modifications were made to the original task compression algorithm: (i) the inputs to the task compression algorithm are task set Γ and $L(h)$, instead of Γ and U_d, and (ii) the equation $U_i = U_{i0} - \left(U_{v0} - U_d + U_f\right)E_i/E_v$ in the original algorithm is replaced by (40). For the case where $L(h) = D_2$, Theorem 4 is applied straightforwardly.

As described in the last section, during iteration $h + 1$, our heuristic will perform a check to determine whether the set of periods found in iteration h is feasible and if the solution is the best one so far. Algorithm 3 is used to accomplish this task. If the constraint in (23) is not satisfied, the heuristic will perform a period "rollback" (Lines 26–33 in Algorithm 3). Essentially, the idea behind a period rollback is to reconsider the current best solution and reduce the corresponding periods by some factor in

Algorithm 2 Task_Compress_Deadline(Γ, *maxIter*)

```
1: $sumC = 0$
2: for each ($\tau_i \in \Gamma$) do
3:    $sumC = sumC + C_i$
4:    if ($sumC > D_i$) then
5:       return NULL // By Lemma 3, no feasible solution exists
6:    end if
7: end for
8: $bestObjF = \infty$
9: for each ($\tau_i \in \Gamma$) do // Initialize some variables
10:    $prevT_i = T_{i0}$
11:    $currT_i = T_{i_{max}}$
12: end for
13: for $h = 0, h < maxIter, h = h + 1$ do
14:    if ($D_1 + currT_1 \leq D_2$) then // Compute $L$ using the set of periods from the previous iteration
15:       $L = D_2$
16:    else
17:       $L = \min_{i=1}^{N}(currT_i + D_i)$
18:    end if
19:    $status$ = Check_Record_Rollback($\Gamma$, $L$, $currT$, $prevT$, $bestT$, $bestObjF$, $h$)
       // The following variables are passed by reference: $currT$, $bestT$ and $bestObjF$
20:    if $status = -1$ then
21:       return NULL // No feasible solution can be found
22:    else if $status = 1$ then
23:       break // Solution cannot be improved further
24:    end if
25:    if ($D_1 + currT_1 \leq D_2$) then
26:       Compute $currT$ following Theorem 4
27:       if ($h = 0$) then
28:          break
29:       end if
30:    else
31:       for each ($\tau_i \in \Gamma$) do
32:          $r_i = L - D_i$
33:          if ($r_i \leq 0$) then // For such a task, set its period to its desired period
34:             $T_i = T_{i0}$
35:             $e_i = 0$
36:             $prevT_i = T_{i0}$
37:          end if
38:       end for
39:       $currT$ = Mod_Task_compress($\Gamma, L$) // Using Theorem 5
40:    end if
41: end for
42: return bestT
```

Algorithm 3 Check_Record_Rollback(Γ, L, $currT$, $prevT$, $bestT$, $bestObjF$, h)

```
1: $objF = 0$
2: for each $(\tau_i \in \Gamma)$ do // Compute objective value
3:     $objF = objF + \frac{1}{e_i}(U_{i0} - C_i/currT_i)^2$
4: end for
5: $cns = 0$
6: for each $(\tau_i \in \Gamma)$ do // Compute the right-hand side of the constraint in (23) to check for feasibility
7:     $cns = cns + \left(\frac{L-D_i}{currT_i} + 1\right) C_i$
8: end for
9: if $(cns > L)$ and $h = 0$ then // No feasible solution found
10:     return -1
11: else if $(cns \leq L)$ and $(objF < bestObjF)$ then // Best solution seen so far, so keep it
12:     $bestObjF = objF$
13:     for each $(\tau_i \in \Gamma)$ do
14:         $bestT_i = currT_i$
15:     end for
16:     if $(cns = L + \epsilon)$ then // The schedulability constraint is active and the task set is feasible so quit
17:         return 1
18:     end if
19:     for each $(\tau_i \in \Gamma)$ do // Check whether task periods have converged to some fixed values
20:         $deltaT_i = |currT_i - prevT_i|$
21:         $prevT_i = currT_i$
22:     end for
23:     if $deltaT_i \leq \Delta$ then
24:         return 1 // Solution converges
25:     end if
26: else if $(cns > L)$ then // The current set of periods is infeasible, perform a period rollback
27:     $prec = prec - 1$ // $prec$ is a global variable
28:     if $(prec < 0)$ then
29:         return 1
30:     end if
31:     for each $(\tau_i \in \Gamma)$ do
32:         $currT_i = bestT_i - \frac{prec}{100} \cdot bestT_i$
33:     end for
34: end if
35: return 0 // Continue improving on solution
```

order to find an even better solution. In our heuristic, the rollback process is controlled by a user-defined parameter, *prec*, which denotes the starting percentage value for period reductions. The iterative process will terminate when certain stopping criterion is met (to be discussed later). The solution thus found may not be optimal but it is guaranteed to be schedulable by the EDF policy.

On the other hand, if the set of periods found in iteration h is feasible and if it is the best one we have seen so far, the heuristic will keep track of this solution (Line 14 in Algorithm 3). To improve on the feasible solution, if one has already been discovered, the heuristic will continue until the periods found in iterations h and $h+1$ are the same. To determine such a solution, a user-defined parameter, Δ, is included as a stopping criterion; if the difference between U_i found in the current iteration and U_i found in the previous iteration is smaller than Δ for all i, the algorithm terminates and returns the best set of periods it has encountered (Lines 23–25 in Algorithm 3). The same action is taken by the heuristic when the constraint in (23) is active (Lines 16–18 in Algorithm 3), where ϵ is some small constant. To handle the case where task periods do not converge to some fixed values (or when it may take too long for the solution to converge), the algorithm uses another user-defined parameter, *maxIter*, to limit the maximum number of iterations.

An additional challenge is how to assign the initial value of L. We propose to set the initial value of L to $\min_{i=1}^{N}(T_{i_{max}}+D_i)$. In this way, if the task set is found to be infeasible, then the algorithm immediately exits since the task set cannot be made schedulable without violating the given period bounds. The following lemma serves to support our choice of L as well as the iterative approach.

Lemma 7: Let T_i for $1 \leq i \leq N$ be a set of periods that satisfy the constraint in (23) and let $L = D_2$ if $T_1 + D_1 \leq D_2$ and $L = \min_{i=1}^{N}(T_i + D_i)$ otherwise. Then any set of $T_i' \geq T_i$ also satisfy the constraint in (23).

Proof: Given that $T_i' \geq T_i$, $\forall i$, it follows that $L' \geq L$. Let $L' = L + \Delta L$ where $\Delta L \geq 0$. Since L and T_i, for all $i = 1, \dots, N$ satisfy the constraint in (23), the following must hold true.

$$L + \Delta L \geq \sum_{i=1}^{N}\left(\frac{L - D_i}{T_i} + 1\right) C_i + \Delta L \tag{47}$$

Using the fact that $\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$, we obtain

$$L + \Delta L \geq \sum_{i=1}^{N}\left(\frac{L - D_i}{T_i} + \frac{1}{T_i}\Delta L + 1\right) C_i \tag{48}$$

It follows that

$$L' \geq \sum_{i=1}^{N}\left(\frac{L' - D_i}{T_i} + 1\right) C_i \tag{49}$$

■

The above lemma has two significant consequences. First, if our algorithm cannot find a feasible solution when setting $L(0) = \min_{i=1}^{N}(T_{i_{max}} + D_i)$, it is not fruitful to continue with the algorithm as any smaller T_i's would not satisfy the constraints in (18). Second, even if a set of feasible periods is found, the algorithm can still attempt to improve on the previously obtained periods in the subsequent iterations. For such iterations h, we set $L(h) = \min_{i=1}^{N}(T_{i_{h-1}} + D_i)$.

To further improve on the quality of the solutions, we will run our proposed algorithm twice. In the first run, we set the initial value of L to be $\min_{i=1}^{N}(T_{i_{max}} + D_i)$ for the reason mentioned above. In the second run, the initial value of L is set to $\min_{i=1}^{N}(T_{i_{min}} + D_i)$. The same heuristic can be used to accomplish this task with some small changes. (Additional checkpoints merely need to be included.) In this way, if a better solution can be found at or near this second value of L, the heuristic will be more likely to find it. Finally, the solution from both runs are compared and the better one will be returned.

The following theorem states the correctness and complexity of our proposed algorithm.

Theorem 6: Consider the period selection problem of a task set with N periodic tasks whose deadline is less than period as formulated in (28)–(32). If there exists a set of task periods such that the task set is schedulable, then Algorithm 2 will always return a feasible solution. That is, the set of periods returned by Algorithm 2 is guaranteed to be schedulable by the EDF policy. In addition, the time complexity of Algorithm 2 is $O(N^2 \cdot maxIter)$.

Proof: We prove Theorem 6 by first considering the case where a feasible solution does not exist. In such a case, when Algorithm 2 calls Algorithm 3 on Line 19, Algorithm 3 will return -1, and consequently cause Algorithm 2 to return NULL in the first iteration. According to Lemma 7, if the task set is infeasible when $T_i = T_{i_{max}}$, then it cannot be made feasible. Since our heuristic initialize the current set of periods to be $T_{i_{\max}}$, for all $i = 1, \ldots, N$ (Line 11 in Algorithm 2), whenever our heuristic returns an empty solution set, it is guaranteed that no feasible set of periods exists.

Now let consider what happens when a solution exists. In such a case, the first set of feasible periods, must be $T_{i_{max}}$, since $currT_i$ is initialized to $T_{i_{max}}$ (Line 11). Additionally, In Algorithm 2, for the set of periods obtained in iteration h, our heuristic performs a check to see whether the task set is schedulable during iteration $h+1$ (Line 19). When the algorithm terminates, the set of task periods that minimizes the objective function, $bestT_i$, for all $i = 1, \ldots, N$, is returned. From our heuristic, we can see that the only place where $bestT_i$'s is updated is on Line 14 of Algorithm 3; the update takes place when the task set with $T_i(h)$, for all $i = 1, \ldots, N$, is feasible and if it is the best set of periods that our heuristic has seen so far (i.e., the set of periods that minimizes the objective value until now). In addition, the value *bestObjF* is initialized to ∞, which means that the first set of feasible periods will be recorded as $bestT_i$'s until a better set of feasible periods is found. Hence, our heuristic will always return a solution, if one exists. In addition, if a solution is returned, then that solution is feasible.

We now examine the time complexity of our heuristic. In [10], Buttazzo et al. proved that the task compression algorithm takes $O(N^2)$ time. Since the changes made to said algorithm does not affect its complexity, **Mod_Task_compress()** will also take $O(N^2)$ time. In addition, since the modified task compression algorithm constitutes the most expensive step in the main for-loop controlled by the user-defined parameter $maxIter$, the worst-case running time of the proposed heuristic is $O(N^2 \cdot maxIter)$. ■

Finally, with sufficiently small *maxIter*, the time complexity makes the proposed algorithm suitable for online period adjustments. In Section VI, we will provide some guidance on how to adjust user-defined parameters (*maxIter*, Δ, and *prec*) based on experimental results.

V. Deadline Selection for Real-Time Tasks

In real-time systems, task deadlines are usually considered fixed parameters. However, there exist situations where it may be more suitable to treat the deadline as an adjustable parameter while keeping the task period constant. For instance, for some control tasks, pre-determined sampling periods (e.g, task periods) must be maintained to satisfy the performance requirement. In such a case, it would make more sense to keep the executing interval unchanged while increasing task deadlines just enough so that the task set becomes schedulable. Many control systems are quite robust and slightly increasing the deadline of a control task will usually not have a significant impact on the performance of the system. Since we assume that EDF is used to schedule tasks, adjusting task deadlines really means adjusting scheduling priorities. Contrary to period adjustment, changing task deadlines allows for the workload to be preserved. In other words, the amount of required work remains the same. This last observation is particularly useful when designing a system where certain amount of work must be done in a specific interval.

Given a range of allowable deadlines, we can redefine our task model as follow. A task τ_i is characterized by the following 6-tuple: $(C_i, D_i, D_{i0}, D_{i_{max}}, T_i, e_i)$, where D_{i0} and $D_{i_{max}}$ is the desired and maximum allowable deadlines, and the rest are the same as defined in Section II-A. The deadline selection problem can be formulated as follows.

$$\text{min:} \quad J(D_1, \ldots, D_N) = \sum_{i=1}^{N} w_i \left(D_{i0} - D_i\right)^2 \tag{50}$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} U_i D_i \geq L \sum_{i=1}^{N} (U_i - 1) + \sum_{i=1}^{N} C_i \tag{51}$$

$$L = \begin{cases} D_2 & : D_1 + T_1 \leq D_2 \\ \min(T_i + D_i) & : otherwise \end{cases} \tag{52}$$

$$D_i \leq D_{i_{max}}, \forall i = 1, \ldots, N \tag{53}$$

$$D_i \geq D_{i0}, \forall i = 1, \ldots, N \tag{54}$$

where (51) is obtained by moving D_i's in (23) to the left-hand side of the inequality and regrouping the terms.

The above optimization problem has the same form as the optimization problem in (28)–(32). Since L cannot be treated as constant, the optimization problem in (50)–(54) belongs to the class of nonlinear programs. Hence, we can take the same approach as in the last section. Namely, L can be treated as a constant (in each iteration) and a heuristic can be used to solve the problem efficiently. The following theorem describes the optimal value of D_i^* for the optimization problem in (50)–(54) when L is a constant.

Theorem 7: Given the constrained optimization problem as specified in (50)–(54), for a fixed value of L (where $L = D_2$ if $D_1 + T_i \leq D_2$ and $L = \min_{i=1}^{N}\{T_i + D_i\}$ otherwise) and $\sum_{i=1}^{N} U_i D_{i0} < L \sum_{i=1}^{N} (U_i - 1) + \sum_{i=1}^{N} C_i$, let

$$\begin{aligned} S &= \sum_{D_j^* \neq D_{j_{max}}} \frac{U_j^2}{w_j}, \text{and} \\ \overline{D} &= \sum_{D_j^* \neq D_{j_{max}}} U_j D_{j0} + \left(L \sum_{j=1}^{N} \big(U_j - 1\big) + \sum_{j=1}^{N} C_j \right) \\ &- \sum_{D_j^* = D_{j_{max}}} U_j D_{j_{max}}. \end{aligned}$$

A solution, D_i^*, is optimal if and only if

$$D_i^* = D_{i0} - \frac{U_i}{w_i S} \cdot \overline{D} \tag{55}$$

for $\frac{w_i}{U_i}\left(D_{i0} - D_{i_{max}}\right) \leq \frac{\overline{D}}{S} \leq 0$.

Proof: The theorem can be proved by the same argument used in proving Theorem 5. ■

The similarity between the solutions to the optimization problem in (50)–(54) and those to the one in (28)–(32) should be apparent. Hence, the heuristic proposed in the last section can easily be modified to solve this problem with the same time complexity. Namely, (55) is used instead of (40) and the deadline bounds must be checked instead of the period bounds.

Interestingly, the idea of deadline selection can be extended to treat systems consisting of sporadic tasks. A sporadic task is a real-time task whose arrival time is not know *a priori*, but there exists some minimum inter-arrival time between any two instances of such task [23]. Although sporadic tasks usually have hard deadlines, for systems where some delays are acceptable, our proposed framework can be used. Specifically, the optimization problem in (50)–(54) can be straightforwardly applied to adjust the deadline of sporadic tasks with a minor change. That is, T_i now denotes the (known) minimum inter-arrival time of a sporadic task, instead of the period of a periodic task as originally defined.

An important implication of using the optimization problem in (50)–(54) for a system with sporadic tasks is that, as long as $I_i \geq T_i$, for all $i = 1, \ldots, N$, where I_i is the actual inter-arrival time of a sporadic task τ_i, the system will remain schedulable using the set of deadlines obtained from solving the optimization problem in (50)–(54). Lemma 7 can be straightforwardly applied to validate this claim.

VI. Experimental Results

In this section, we begin by verifying our claims made in Section III with regards to the optimality of the task compression algorithm in [10]. We then compare our simplified sufficient condition presented in Section IV to the exact formula in (18). The quality of our heuristic is also discussed.

A. Period selection with deadlines equal to periods

To demonstrate that the task compression algorithm solves the optimization problem in (1)–(4), we reuse the task set provided in the experimental results section of [10] (reproduced below in Table I). The task compression algorithm was written in `C++`, while `MatLab` was used to obtain the results for the constrained optimization problem in (1)–(4).

TABLE I
Task set parameters used

Task	C_i	T_{i0}	T_{i0}	$T_{i_{max}}$	e_i
τ_1	24	100	30	500	1
τ_2	24	100	30	500	1
τ_3	24	100	30	500	1.5
τ_4	24	100	30	500	2

In this experiment, all tasks start at time 0 with an initial period of 100 time units and the task set is schedulable under EDF. The required minimum utilization of the overall system is $\frac{24}{500} + \frac{24}{500} + \frac{24}{500} + \frac{24}{500} = 0.192$. Since the current utilization is $\frac{24}{100} + \frac{24}{100} + \frac{24}{100} + \frac{24}{100} = 0.96$, the task set is schedulable under EDF. Assume that, at time 10000, τ_1 needs to reduce its period to 33 time units, perhaps due to some changes in system dynamics not experienced by other tasks. Since the new required minimum utilization of the system is $\frac{24}{33} + \frac{24}{500} + \frac{24}{500} + \frac{24}{500} = 0.871$, which is less than 1, τ_1 will be allowed to change its periods as desired. However, τ_2, τ_3, and τ_4 can no longer execute with their initial period, as this would drive the system utilization to $\frac{24}{33} + \frac{24}{100} + \frac{24}{100} + \frac{24}{100} = 1.45$. In other words, To allow for τ_1 to change its period, the period of tasks τ_2, τ_3, and τ_4 must increase for the system to remain schedulable. It is worth noting that although τ_2 has the exact same parameters as τ_1, we made the assumptions that the system whose task is τ_2 is not in as critical state as that of τ_1 and hence it is possible for τ_2 to increase its period temporarily. At time 20000, τ_1 goes back to its original period. Figures 1 shows the cumulative number of executed instances for each task as its period changes

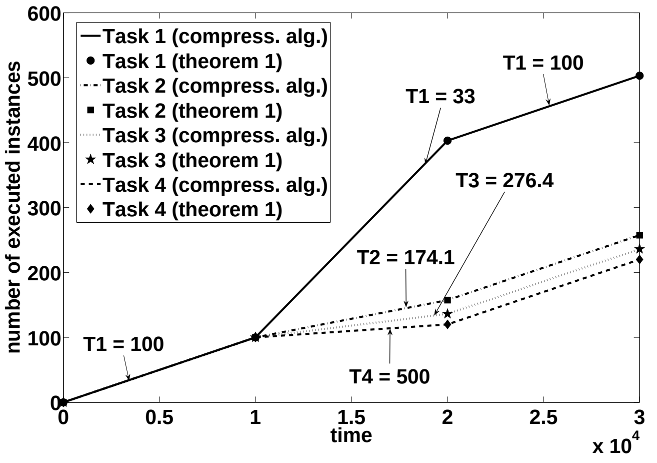

Fig. 1. Utilization perturbation example

over time. First of all, the data verifies that the results obtained from the task compression algorithm and those from Theorem 1 match perfectly.

Furthermore, it can be seen from the graph that the number of executed instances of a task is inversely proportional to its elastic coefficient. Recall that the weight of a task is the inverse of its elastic coefficient. Although τ_2, τ_3, and τ_4 all have the same computation time, initial period, and period range, τ_2 is determined to have the smallest (e.g., best) sampling period because of its weight. On the other hands, τ_4 has the largest sampling period because it is considered to be of least importance.

B. Period selection with deadlines less than or equal to periods

To illustrate the practicality and performance of our heuristic approach, we present the following comparisons in this section. First, we compare the simplified sufficient condition in (23) with the original necessary and sufficient condition in (18). Second, we demonstrate the capability of the proposed heuristic by comparing the number of problems it is able to solve with what can be solved by an optimization solver for the optimization problem in (28)–(32). Third, to assess the performance of the heuristic, we compare the quality of the solutions obtained by using the heuristic with that of the optimal solutions.

To perform the aforementioned comparisons, 1000 task sets consisting of 5 tasks each were randomly generated for 9 different utilization levels ($U_{level} = 0.1, \ldots, 0.9$) with a total of 9000 task sets in overall. The utilization level is defined to be $U_{level_i} = \sum_{j=1}^{N} \frac{C_j}{T_{j_{max}}}$, $i = 0.1, \ldots, 0.9$. Each task set is made to be initially unschedulable with $T_{i0} = D_i$, $\forall i$, but at least a feasible schedule can be found by setting $T_i = T_{i_{max}}$, $\forall i$ using the necessary and sufficient condition in (18). This setup aims to eliminate all trivial solutions. In addition to the utilization level, the maximum hyperperiod, minimum period, maximum period, precision, and maximum number of tries must also be specified. 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.

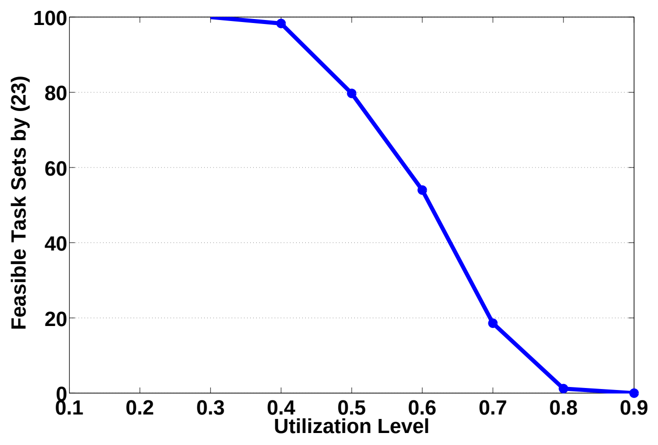

Fig. 2. Performance of simplified sufficient condition

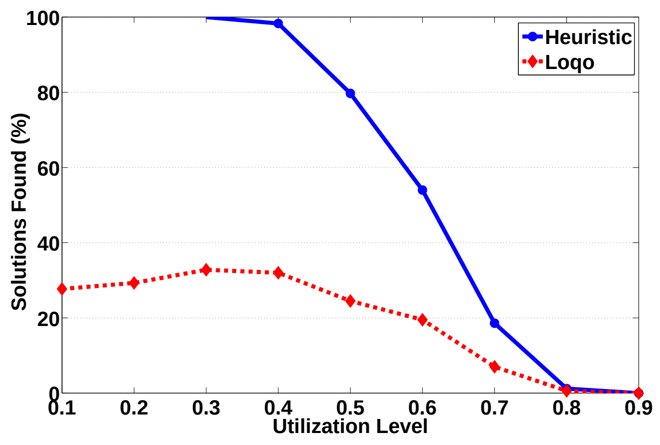

Fig. 3. Heuristic vs. Loqo solver

In a nutshell, the following steps were taken to generate a task set. First of all, a set of periods were randomly generated based on the minimum period, maximum period, hyperperiod bound, and precision. 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.) 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 assigned a deadline that ensure that $\sum_{i=1}^{N} \frac{C_i}{D_i} > 1$. As a final step, the random task generator tests the schedulability of a task set using the necessary and sufficient condition in (18). 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.

To perform experiments for comparing the simplified sufficient condition in (23) with the original condition in (18), we make the following observation. We know that the task sets generated pass the test in (18), provided that we use $T_{i_{max}}$ as the period for each task τ_i, for $i = 1, \ldots, N$. Hence, we only need to test the condition in (23) in the same fashion.

Figure 2 compares the condition in (23) to that in (18). The x-axis shows each utilization level and the y-axis indicates the percentage of task sets that were found to be feasible by the condition in (23). (Recall that the percentage of schedulable task sets using the test in (18) is 100% for each utilization level.) As can be seen from the plot, the simplified sufficient condition in (23) found that a feasible solution exists for all task sets with $U_{level} \leq 0.3$. As expected, said condition becomes more pessimistic as U_{level} increases. However, it still finds over 50% of the task set with U_{level} of 0.6 to be feasible. Note that we could have tested each task set against the existing sufficient condition (a task set is schedulable if $\sum_{i=1}^{N} \frac{C_i}{D_i} \leq 1$ [23]), but said test will determine all task sets to be infeasible, since the task set generator returns a task set while guaranteeing that $\sum_{i=1}^{N} \frac{C_i}{D_i} > 1$. Indirectly, we can conclude that, our modified sufficient condition is less pessimistic than the existing sufficient condition.

In the second experiment, we compare the percentage of solutions found by our heuristic, as opposed to solving the optimization problem by an optimization solver. Our heuristic was implemented in `C++`. For the optimization solver, we used `Loqo` [30], a nonlinear solver with an interior-point algorithm. In this experiment, *maxIter* and Δ in our heuristic were set to 200 and 1×10^{-5}, respectively. The precision parameter, *prec*, used in period rollback as described in the last section, was set to 20 (denoting 20% period reductions). We limited the maximum number of iterations allowed by `Loqo` to be 200. We did not allow `Loqo` to run longer, as, according to [31], it is unlikely that an optimal solution would be found after the $100th$ iteration.

Figure 3 compares the number of solutions found by `Loqo` and that found by our heuristic. As before, the x-axis shows the different utilization levels, whereas the y-axis shows the percentage of solutions found. It is clear from the plot that the heuristic is able to find a much larger number of solutions that `Loqo`. This is not surprising, as an optimization software does not guarantee that a solution to a non-linear programming problem will be found, even if one exists. This is one important advantage of the heuristic; according to Theorem 6, it is guaranteed to always return a feasible task set if one exists.

The last experiment examines the quality of the solutions found by our heuristic in comparison with that found by `Loqo`. For the 9000 task sets (1000 task sets for each utilization level), `Loqo` only solved 1416 task sets. Therefore, we could only compare the results from the heuristic to these solutions. It must be emphasized, however, that the solutions from the heuristic is a superset of those from `Loqo`.

Figure 4 shows a bar chart depicting the performance of the proposed heuristic as compared to the optimal solutions found by `Loqo`. The x-axis shows the differences in the objective function values (from the heuristic and `Loqo`). The y-axis shows the percentage of the solutions by our heuristic that result in a given difference. The first, and most obvious, observation is that the heuristic is able to find the global optimal solution to over 50% of the solutions, ignoring numerical errors. Second, almost 30% of the solutions found by the heuristic is very close to the global optimal ones.

The above experimental results demonstrate that our heuristic performs well enough to be deployed in real applications; not only has the heuristic found the global or close to global optimal solution in many cases, it also guarantees to always return a feasi-

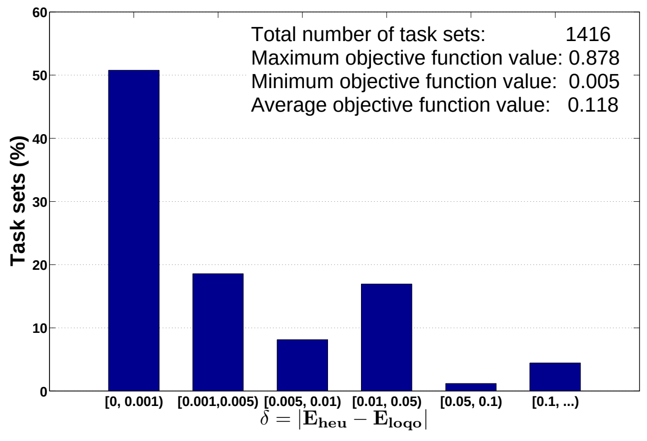

Fig. 4. Comparison of the heuristic and Loqo in terms of objective function values

ble schedule if one exists. In addition, the low time complexity of our heuristic makes it suitable for online use for dynamic period adjustments. The experiment also suggests that the maximum number of iterations, *maxIter*, need not be greater than 200 for the heuristic to find a solution. The user-defined parameter Δ can be set to be equal to the time granularity used by the operating system, since this time granularity is the smallest time unit that the operating system can handle. Finally, we suggest setting the period rollback precision, *prec*, to some small value in comparison to the value of *maxIter*. It does not make sense to set *prec* to be a large value (e.g. 50) if *maxIter* is relatively small (e.g. 200), as the heuristic will then be performing the rollback process for most of the time.

VII. Conclusions and Future Work

In this paper, the following main contributions were made. First, we introduced a general framework which formulates a trade-off between task set schedulability and a specific performance metric (such as task utilization) as an optimization problem. Such a framework allows for real-time systems under temporal overloads to graciously adapt by adjusting their performance level. Second, we proved the optimality of the existing task compression algorithm in [10]. Said algorithm allows for the period selection problem for tasks with deadlines equal periods to be solved optimally in an online manner. Third, our framework is further generalized to consider situations where task deadlines are less than task periods. In this case, we propose an efficient heuristic to solve the problem online while making use of the slightly modified task compression algorithm. Experimental results show that our heuristic performs satisfactorily and in many cases finds the global optimal solution. Fourth, we provide and motivate the use of a framework for solving the deadline selection problem, which can be applied to some control systems with pre-determined sampling time. Last, our flexible framework can be used to solve other problems where the schedulability-performance trade-off is central. The framework will permit the development and comparisons of efficient algorithms.

Since the algorithm presented in Section IV is best-effort, it would be interesting to study whether there exists a way to select the value of L at every iteration such that the solution found will always be optimal. Finally, as future work, it would be interesting to explore different classes of objective functions and constraints that may be even harder to solve.

Acknowledgment

The authors would like to thank Bren Mochocki for providing his tool to generate task sets used in the experiment.

References

[1] Aydin, H., Melhem, R., Mosse, D., and Alvarez, P. Optimal reward-based scheduling for periodic real-time tasks. In *Proc. Real-Time Systems Symposium* (1999), pp. 79–89.
[2] Baillieul, J., and (editors), P. A. Special issue on networked control systems. *Transactions on Automatic Control 49*, 9 (Sept. 2004), 1421–1423.
[3] Baruah, S., Cohen, N., Plaxton, G., and Varvel, D. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica 15*, 6 (June 1996), 600–625.
[4] Baruah, S., Rosier, L., and Howell, R. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems 2*, 4 (Nov. 1990), 301–324.
[5] Bernat, G., Burns, A., and Llamosí, A. Weakly hard real-time systems. *Transactions on Computers 50*, 4 (Apr. 2001), 308–321.
[6] Bini, E., and Natale, M. D. Optimal task rate selection in fixed priority systems. In *Proc. Real-Time Systems Symposium* (2005), pp. 399–409.
[7] Buttazzo, G. *Hard real time computing systems: predictable scheduling algorithms and applications.* Springer, 2005.
[8] Buttazzo, G., and Abeni, L. Smooth rate adaptation through impedance control. In *Proc. Euromicro Conf. on Real-Time Systems* (2002), pp. 3–10.
[9] Buttazzo, G., Lipari, G., and Abeni, L. Elastic task model for adaptive rate control. In *Proc. Real-Time Systems Symposium* (1998), pp. 286–295.
[10] Buttazzo, G., Lipari, G., Caccamo, M., and Abeni, L. Elastic scheduling for flexible workload management. *Transactions on Computers 51*, 3 (Mar. 2002), 289–302.
[11] Caccamo, M., Buttazzo, G., and Sha, L. Elastic feedback control. In *Proc. Euromicro Conf. on Real-Time Systems* (2000), pp. 121–128.
[12] Cervin, A., Eker, J., Bernhardsson, B., and K.-E.Årzén. Feedback-feedforward scheduling of control tasks. *Real-Time Systems 23*, 1 (July 2002), 25–53.
[13] Chantem, T., Hu, X., and Lemmon, M. Generalized elastic scheduling. In *Proc. Real-Time Systems Symposium* (2006), pp. 236–245.
[14] Chung, J.-Y., Liu, J. W., and Lin, K.-J. Scheduling periodic jobs that allow imprecise results. *Transactions on Computers 39*, 9 (Sept. 1990), 1156–1175.
[15] Eker, J., Hagander, P., and Årzén, K.-E. A feedback scheduler for real-time controller tasks. *Control Engineering Practice 8*, 1369–1378 (Dec. 2000), 12.
[16] Gai, P., Abeni, L., Giorgi, M., and Buttazzo, G. A new kernel approach for modular real-time systems development. In *Proc. Euromicro Conf. on Real-Time Systems* (2001), pp. 199–208.
[17] Hamdaoui, M., and Ramanathan, P. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. *Transactions on Computers 44*, 12 (1995), 1443–1451.
[18] Hamdaoui, M., and Ramanathan, P. Evaluating dynamic failure probability for streams with (m,k)-firm deadlines. *Transactions on Computers 46*, 12 (1997), 1325–1337.
[19] Koren, G., and Shasha, D. Skip-over: Algorithms and complexity for overloaded systems that allow skips. In *Proc. Real-Time Systems Symposium* (1995), pp. 110–117.
[20] Koutsoukos, X., Tekumalla, R., Natarajan, B., and Lu, C. Hybrid supervisory utilization control of real-time systems. In *Proc. Real-Time & Embedded Technology and Applications Symposium* (2005), pp. 12–21.
[21] Kuo, T.-W., and Mok, A. Load adjustment in adaptive real-time systems. In *Proc. Real-Time Systems Sympsium* (1991), pp. 160–171.
[22] Liu, C., and Layland, J. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM 20*, 1 (Jan. 1973), 46–61.
[23] Liu, J. W. S. *Real-Time Systems.* Prentice-Hall, NJ, 2000.
[24] Luenberger, D. *Linear and Nonlinear Programming.* Addison-Wesley Publishing Co., 1989.

[25] Mok, A., and Wang, W. Window-constraint real-time periodic task scheduling. In *Proc. Real-Time Systems Symposium* (2001), pp. 15–24.

[26] Quan, G., and Hu, X. Enhanced fixed-priority scheduling with (m,k)-firm guarantee. In *Proc. Real-Time Systems Symposium* (2000), pp. 79–98.

[27] Ramanathan, P. Overload management in real-time control applications using (m,k)-firm guarantee. *Transactions on Parallel and Distributed Systems 10*, 6 (June 1999), 549–559.

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

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

[30] Vanderbei, R. Loqo. $<$http://www.princeton.edu/ rvdb/$>$.

[31] Vanderbei, R. LOQO: An interior point code for quadratic programming. *Optimization methods and software 12*, 5 (1999), 451–484.

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