Elastic Scheduling for Fixed-Priority Constrained-Deadline Tasks

Marion Sudvarg, Sanjoy Baruah, Chris Gill
Department of Computer Science & Engineering
Washington University in St. Louis
(msudvarg, cdgill, baruah)@wustl.edu

***Abstract*—Elastic scheduling provides a model for systems in which individual task utilizations can adapt to guarantee schedulability despite limited resources. Each task is characterized by a range of acceptable utilizations and an "elastic constant" representing its flexibility to reduce or "compress" its utilization from the desired maximum. Utilization compression is realized by either extending task periods or reducing workloads. This paper extends the model to address period compression for fixed-priority constrained-deadline task systems scheduled on a uniprocessor. We propose two approximate algorithms and one optimal algorithm for determining compression under the model. We then compare the execution times and accuracies of all three, demonstrating that even for large task sets, online compression can be performed feasibly on low-powered embedded systems.**

***Index Terms*—elastic constrained-deadline task model, uniprocessor fixed-priority scheduling, deadline-monotonic priority assignment**

I. Introduction

Elastic real-time scheduling models provide a framework to reduce the utilizations of individual tasks in response to system overload. The original model proposed by Buttazzo et al. [1], [2] considers uniprocessor scheduling of implicit-deadline tasks. Using a physical analogy, it represents each task's utilization as a spring; the task's "elasticity" reflects its ability to adapt its utilization to a lower quality of service. The total length of the springs, placed end-to-end, represents system utilization. If this exceeds the schedulable bound, a compressive force is applied to the system. Each spring (and corresponding utilization) is compressed proportionally to its elasticity until the total utilization no longer exceeds the bound or until the task reaches its minimum serviceable utilization; task periods are adjusted accordingly.

Chantem et al. [3], [4] demonstrated the equivalence of elastic scheduling to the problem of minimizing a weighted sum of squared deviations of each task's compressed utilization from its nominal value, constrained by each task's minimum utilization and the total utilization bound of the system. The model was extended to constrained-deadline tasks, for which utilization compression extends periods while holding the relative deadlines constant. The constraint on total utilization was replaced by a tractable approximation of the processor-demand analysis (PDA) [5] test for earliest-deadline first (EDF) schedulability. Recently, Baruah [6] demonstrated that these approximations result in a high degree of pessimism for certain task sets. An alternative was presented in [6] that uses an iterative approach, increasing the total compression according to a tunable step size until the system is schedulable according to PDA. The algorithm is tuned by selecting the amount to increase compression at each step; a smaller step increases the running time of the algorithm but allows for a more precise result.

In this work, we extend this approach to deadline-monotonic (DM) uniprocessor scheduling of systems of constrained-deadline elastic tasks. We first present an iterative algorithm that similarly increases compression until schedulability is achieved according to the response time analysis (RTA) test [7]. We then present two refinements to this algorithm, both leveraging the observation that once a task has been compressed to schedulability according to RTA, it remains schedulable when more compression is applied to the system. The first refinement iterates over tasks in order of decreasing priority, increasing total compression until that task is schedulable under RTA before considering the next task. The second performs binary search over the range of allowed compression, skipping RTA for tasks that are already known to be schedulable at lower levels of compression.

Next, we formulate the problem of finding the optimal amount of compression to guarantee schedulability for a *single* task as a mixed integer quadratic program (MIQP). This allows us to present an alternative algorithm that considers each task in turn; if a task is not RTA schedulable for the current level of system compression, the MIQP is solved to find the exact amount of compression necessary to schedule the task. By iterating over each task, we can determine the minimum sufficient compression that must be applied to the task system.

We implement the latter three approaches, using SCIP [8] to solve the MIQP. By evaluating each algorithm for randomly-generated synthetic task sets, we demonstrate that the approximate procedures are both highly efficient and typically give a result close enough to optimal to be useful for online scheduling decisions in low-powered embedded devices. We also show that, when an optimal solution is desired, the MIQP may be solved feasibly offline to compress task periods.

The remainder of this paper is organized as follows: in

This research was supported in part by NSF grants CNS-2141256 and CNS-2229290 (CPS), NASA grant 80NSSC21K1741, and a gift from BECS Technology, Inc.

979-8-3503-3902-4/23/$31.00 ©2023 IEEE

Section II, we provide the necessary background on system models used in this paper. Section III presents a basic iterative algorithm to compress tasks until RTA guarantees schedulability. Sections IV and V refine the algorithm to a more efficient iterative approach and a binary search, respectively. Section VI formulates an MIQP representation of the problem of finding the optimal amount of compression to schedule a single task, then applies this to a complete task system. In Section VII, we show results of our evaluation of those approaches. Finally, Section VIII concludes the paper and discusses the contexts under which each approach may be relevant.

II. Background and System Model

A. Implicit-Deadline Elastic Tasks

The elastic model for recurrent, implicit-deadline tasks on a uniprocessor [1], [2] characterizes each task $\tau_i=(C_i, U_i^{\min}, U_i^{\max}, U_i, E_i)$ by five non-negative parameters:

- C_i: The task's worst-case execution time.
- $U_i^{\max}$: The task's maximum utilization, i.e., its nominal value when executing at the desired service level in an uncompressed state.
- $U_i^{\min}$: Its minimum utilization, i.e., a bound on the amount its service can degrade.
- U_i: The task's assigned utilization, constrained to $U_i^{\min} \leq U_i \leq U_i^{\max}$ (the value of this parameter needs to be assigned prior to run-time).
- E_i: An elastic constant, representing "the flexibility of the task to vary its utilization" [1].

A task system $\Gamma = \{\tau_1, \ldots, \tau_n\}$ has a total uncompressed utilization $U_{\text{SUM}}^{\max}$ expressed as

$$U_{\text{SUM}}^{\max} = \sum_{i=1}^{n} U_i^{\max} \tag{1}$$

and a desired utilization U_D representing the utilization bound given by the scheduling algorithm in use. In the event of system overload, i.e., if $U_{\text{SUM}}^{\max} > U_D$, the elastic model assigns a utilization U_i to each task τ_i according to these conditions:

- $\sum_i U_i^n = U_D$, i.e., total utilization is set to the schedulable bound.
- Any task for which $E_i = 0$ is considered inelastic; we consider this equivalent to the case that $U_i^{\min} = U_i^{\max}$.
- For all other tasks τ_i and τ_j, if $U_i > U_i^{\min}$ and $U_j > U_j^{\min}$, then U_i and U_j must satisfy the relationship

$$\left(\frac{U_i^{\max} - U_i}{E_i}\right) = \left(\frac{U_j^{\max} - U_j}{E_j}\right)$$

Put simply, the model compresses each task's utilization such that it is reduced from its desired maximum proportionally to the task's elasticity parameter, subject to the constraint that it remains no less than the specified minimum.[1] Compression is then realized by adjusting each task's period T_i according to its new utilization, i.e., $T_i = C_i/U_i$. The original algorithm presented in [1], [2] for assigning utilizations under this model executed in time quadratic in the number of elastic tasks. A recent improvement [9] yielded a more efficient algorithm that runs in quasilinear time (or linear time for admission of a new task).

[1]This statement holds true for inelastic tasks, as $E_i = 0$ implies $U_i^{\min} = U_i^{\max}$, and therefore the utilization is not reduced.

B. Constrained-Deadline Tasks

In [3], [4], Chantem et al. showed that utilizations selected by the elastic model also solve the following quadratic programming problem:

$$\min_{U_i} \quad \sum_{i=1}^{n} \frac{1}{E_i}(U_i^{\max} - U_i)^2 \tag{2a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} U_i \leq U_D \tag{2b}$$

$$\forall_i, \quad U_i^{\min} \leq U_i \leq U_i^{\max} \tag{2c}$$

This allowed for an extension of the model to constrained-deadline tasks. A task $\tau_i = (C_i, D_i, U_i^{\min}, U_i^{\max}, U_i, E_i)$ is now characterized with an additional parameter, D_i, representing a relative deadline that remains fixed even if the task's period is extended in response to reduced utilization. Under the constrained deadline model, only tasks for which $D_i \leq T_i$ (i.e., $D_i \leq C_i/U_i^{\max}$) are considered.

The schedulability constraint (Eqn. 2b) is replaced by a representation of the PDA [5] schedulability test. PDA is an optimal technique for schedulability analysis of constrained-deadline sporadic task systems under preemptive EDF scheduling on a uniprocessor. It considers the demand bound function $\text{DBF}_i(t)$ for each task τ_i, which denotes the maximum possible cumulative execution required by jobs of the task that arrive and have their deadlines within any contiguous interval of duration $t \geq 0$. This can be computed as:

$$\text{DBF}_i(t) = \max\left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \tag{3}$$

For $t \geq 0$ and $D_i \leq T_i$, this can be expressed more simply as:

$$\text{DBF}_i(t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \times C_i \tag{4}$$

A constrained-deadline task system $\Gamma = \{\tau_1, \ldots, \tau_n\}$ is schedulable under preemptive EDF on a uniprocessor if and only if for all $t > 0$,

$$\sum_{i=1}^{n} \text{DBF}_i(t) \leq t \tag{5}$$

In [5], it was shown that it is sufficient to check this condition for values of t within the first hyperperiod that take the form (kT_i+D_i) for non-negative integers k. This set of values constitute the PDA *testing set*. For elastic scheduling, Chantem et al. [3], [4] add a constraint in the form of Eqn. 5 for each element of the testing set to the optimization problem represented by Eqn. 2.

However, the resulting problem may not be tractable. The size of the testing set may be exponential in general, and

pseudo-polynomial for bounded-utilization tasks, and may result in an optimization problem with too many constraints to be efficiently solvable. Also, due to its use of the floor function, $\text{DBF}_i(t)$ is not a linear expression, and so the optimization problem does not remain a quadratic program. Chantem et al. [3], [4] over-approximate the demand-bound function by removing the floor. Nonetheless, because the test set itself depends on the task periods, the times defining the RHS of the constraints formed by Eqn. 5 are variables in the optimization problem, and the problem remains non-linear. Chantem et al. [3], [4] introduced an approximate form of the problem, and a heuristic approach to solving it, that were proved correct in the sense that the resulting compressed system would be guaranteed schedulable.

In recent work [6], Baruah showed that these approximations are highly conservative and may result in significant over-compression for certain task sets. Two alternative approaches were presented; in this paper, we extend these to fixed-priority DM scheduling. Baruah introduces a term λ representing the degree by which compression is applied to the task system. Recall that in the original model [1], [2], each task τ_i is compressed proportionally to its elasticity E_i, but not beyond its minimum utilization $U_i^{\min}$. This allows us to express the utilization U_i of each task as:

$$U_i = \max(U_i^{\min}, U_i^{\max} - \lambda E_i) \tag{6}$$

Since $T_i = C_i/U_i$, we define $T_i^{\min} = C_i/U_i^{\max}$ and $T_i^{\max} = C_i/U_i^{\min}$. Then for $\lambda < C_i/(E_i T_i^{\min})$:

$$T_i = \min\left(T_i^{\max}, \frac{C_i T_i^{\min}}{C_i - \lambda E_i T_i^{\min}}\right) \tag{7}$$

In an uncompressed state, $\lambda = 0$ and for each task, $U_i = U_i^{\max}$ (equivalently, $T_i = T_i^{\min}$). The values $U_i^{\min}$ also imply a value $\lambda_i^{\max}$ beyond which the utilization of task τ_i cannot be compressed further:

$$\lambda_i^{\max} \stackrel{\text{def}}{=} \begin{cases} \frac{U_i^{\max} - U_i^{\min}}{E_i} & \text{if } E_i > 0 \\ 0 & \text{if } E_i = 0 \end{cases} \tag{8}$$

Baruah also defines a value $\lambda_{\max}$ beyond which the system cannot be further compressed [6]; we express it as:

$$\lambda_{\max} \stackrel{\text{def}}{=} \max_i \lambda_i^{\max} \tag{9}$$

which implies an alternative expression for T_i as follows:

$$T_i = \begin{cases} \frac{C_i T_i^{\min}}{C_i - \lambda E_i T_i^{\min}} & \text{if } 0 \le \lambda < \lambda_i^{\max} \\ T_i^{\min} & \text{if } \lambda \ge \lambda_i^{\max} \end{cases} \tag{10}$$

Baruah [6] also introduces the notation $\Gamma(\lambda)$, representing the task system obtained from Γ by applying compression λ, i.e., with each task τ_i having a period T_i according to Eqn. 7. An optimal algorithm, then, for elastic scheduling of constrained-deadline task systems under EDF finds the value λ^* representing the minimum value λ for which $\Gamma(\lambda)$ is schedulable. In [6], Baruah presents two algorithms that, while not optimal, are nonetheless tunable by a parameter ϵ; both algorithms are guaranteed to find a value $\lambda < \lambda^* + \epsilon$ for which $\Gamma(\lambda)$ is schedulable. We summarize both:

1) ELASTIC*:* This algorithm iterates over values of $\lambda \in [0, \lambda_{\max}]$ with granularity ϵ. For each value of λ tested, it performs PDA over the task set $\Gamma(\lambda)$. Once PDA indicates schedulability, the search stops, and compression is applied. For efficiency, binary search is proposed as an alternative. For a considered value of λ, if PDA indicates schedulability, a smaller value of λ is subsequently tested; if not, a larger value is checked. The binary search limits the number of times PDA is performed to $\left\lceil \log_2\left(\frac{\lambda_{\max}}{\epsilon}\right)\right\rceil$; PDA is itself pseudo-polynomial for bounded-utilization task systems.

2) ELASTIC-EFFICIENT*:* A more efficient algorithm is supported by two observations in [6], repeated here:

Observation 1. *If a given sporadic task system Γ satisfies Expression 5 for a given value of t (say, t_o), then any task system Γ' obtained from Γ by increasing the period parameters of one or more tasks also satisfies Expression 5 for t_o.*

Observation 2. *Let Γ denote some constrained-deadline elastic sporadic task system, and λ, ϵ, and t_s denote positive numbers. If all elements in the testing set of $\Gamma(\lambda)$ that are $\le t_s$ satisfy Condition 5, then all elements in the testing set of $\Gamma(\lambda + \epsilon)$ that are $\le t_s$ also satisfy Condition 5*

The algorithm proceeds by iterating over values of λ, beginning with $\lambda = 0$. It considers elements of the PDA testing set in increasing order. Baruah observes that the testing set need not be enumerated in its entirety a priori [6]; instead, the current element being tested, t_o, can be updated to the smallest value from amongst the *next* deadlines of each task. When an element is reached for which PDA fails at the current test set element t_o, λ is incremented by ϵ. Observation 2 implies that once PDA succeeds at t_o, only larger values need to be tested. Once the test set is exhausted, the current value λ is returned. However, if λ reaches $\lambda_{\max}$, the algorithm terminates, as the task system remains unschedulable even under compression.

Because the algorithm essentially performs a single PDA (the testing set is only traversed once), while additionally recomputing the periods of each task τ_i in $\Gamma(\lambda)$ for each value of λ, the worst-case running time of the algorithm is:

$$O\left(n \times \left\lceil \frac{\lambda_{\max}}{\epsilon} \right\rceil\right) + \text{ the running time of PDA.}$$

where n denotes the number of tasks in Γ. For constant ϵ, this is dominated by the running time of PDA.

III. EXTENSION TO FIXED-PRIORITY SCHEDULING

In this section, we present a simple extension of Baruah's algorithm ELASTIC [6] (summarized in Section II-B2) to fixed-priority deadline-monotonic (DM) scheduling, which maintains the priority order of tasks as their periods are extended. The procedure is outlined in Algorithm 1. Given a system Γ of elastic constrained-deadline tasks (characterized as described in Section II), it seeks to determine the smallest value of λ for which $\Gamma(\lambda)$ is schedulable. Like the ELASTIC algorithm, its precision is tunable by a parameter ϵ; the value λ found is guaranteed to be less than $\lambda^* + \epsilon$.

Algorithm 1: Elastic-FP(Γ)

```
1  Input: Elastic constrained-deadline task system Γ
2  Output: Smallest λ such that Γ(λ) is DM-schedulable
3  λ ← 0
4  λ_max computed according to Eqn. 9
5  repeat
6      Perform RTA for Γ(λ)
7      if Γ(λ) is schedulable then
8          return λ
9      else
10         λ ← λ + ε
11 until λ > λ_max;
12 return FAILURE
```

The algorithm initializes λ, the amount of compression to be applied, to 0. It then increases λ in steps of size ϵ, performing RTA for the complete task set $\Gamma(\lambda)$ for each value of λ. Once λ is found for which schedulability is achieved, the algorithm terminates and the value is returned. However, if λ exceeds $\lambda_{\max}$, the utilization constraints on each task prevent the system from being scheduled under the elastic model.

Running Time: Since at most $\lceil \lambda_{\max}/\epsilon \rceil$ calls are made to RTA by the algorithm ELASTIC-FP, its worst-case running time is $\lceil \frac{\lambda_{\max}}{\epsilon} \rceil \times$ the worst-case running time of RTA (which is pseudo-polynomial in the representation of the task system).

IV. An Efficient Iterative Approach

In this section, we present the first of two refinements to Algorithm 1 (ELASTIC-FP). We begin with a brief summary of Audsley et al.'s response time analysis (RTA) [7], which will provide a key observation leveraged by both refinements.

A. Response-Time Analysis

A task set Γ is schedulable if and only if the response time of each task does not exceed its deadline. Under fixed-priority preemptive scheduling, the response time R_i of a task τ_i is characterized as the sum of its execution time C_i and the interference I_i of the higher priority tasks; the interference is, itself, a function of the response time. Assuming without loss of generality that tasks are indexed by decreasing priority, the following expression describes the response time:

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{11}$$

Audsley et al. [7] describe a recursive process by which to determine the response time; the system is schedulable if and only if R_i does not exceed the deadline D_i for each task τ_i. If used in the context of RTA, our algorithm requires up to $\lceil \lambda_{\max}/\epsilon \rceil$ calls to RTA in the worst case. However, the following observation provides a slight improvement to execution time:

Observation 3. *If the condition $R_i \leq D_i$ holds for a task τ_i in $\Gamma(\lambda)$, then the condition also holds for the same task τ_i in $\Gamma(\lambda + \delta)$ for any $\delta > 0$.*

Proof. Since the period T_i only appears in the denominator in the expression for computing response time (Eqn. 11), and the period does not decrease as λ increases (from Eqn. 10), it follows that R_i does not increase when increasing λ. Therefore, if $R_i \leq D_i$ for some λ, the inequality still holds as λ increases. □

B. The Algorithm

This observation implies that Algorithm 1 can be improved by considering only a *single* task at a time. RTA requires checking every task in a task system, but once a task is shown to be schedulable for a given value of λ, it need not be rechecked for larger values. The resulting improvement is outlined in Algorithm 2.

Algorithm 2: Elastic-FP-Efficient(Γ)

```
1  Input: Elastic constrained-deadline task system Γ
2  Output: Smallest λ such that Γ(λ) is DM-schedulable
3  λ ← 0
4  λ_max computed according to Eqn. 9
5  i ← 1
6  repeat
7      Perform RTA for task τ_i(λ)
8      if τ_i(λ) is schedulable then
9          if i == n then
10             return λ
11         i ← i + 1
12     else
13         λ ← λ + ε
14 until λ > λ_max;
15 return FAILURE
```

As in Algorithm 1 (ELASTIC-FP), ELASTIC-FP-EFFICIENT begins by initializing λ to 0. It considers tasks in turn, beginning with τ_1, the highest-priority task. The algorithm introduces the notation $\tau_i(\lambda)$ to refer to task $\tau_i \in \Gamma(\lambda)$, i.e., task τ_i having a period T_i according to Eqn. 10 for the given value of λ. When RTA determines that the current task under consideration, τ_i, is unschedulable for the current value of λ, the algorithm increases λ in steps of ϵ until the task is schedulable. At this point, it considers the next task in the system. If there are no tasks remaining to be checked (line 9), the algorithm terminates and returns the current value of λ. However, if the value of λ exceeds $\lambda^{\max}$, the algorithm fails.

Running Time: As before, at most $\lceil \lambda_{\max}/\epsilon \rceil$ values of λ are checked by the algorithm ELASTIC-FP-EFFICIENT. However, RTA is only performed for a *single task* at a time. For a single value of λ, no more than a single failing check can be made. Additionally, each task need only have a single successful

check. Therefore, for a task system Γ of size n, the total running time of the algorithm can be expressed as:

$$\left(\left\lceil \frac{\lambda_{\max}}{\epsilon} \right\rceil + n - 1\right) \times \text{the running time of RTA for a single task} \tag{12}$$

V. A Binary Search Implementation

Our second refinement to Algorithm 1 (ELASTIC-FP) instead performs binary search over values of λ in $[0, \lambda_{\max}]$. Observation 3 implies that, when testing using RTA to test $\Gamma(\lambda)$ for schedulability, any tasks already known to be schedulable for smaller values of λ do not need to be rechecked. The complete procedure is outlined in Algorithm 3.

Algorithm 3: Elastic-FP-BS(Γ)

```
Input: Elastic constrained-deadline task system Γ
Output: Smallest λ such that Γ(λ) is DM-schedulable
λ_lo ← 0
λ_max computed according to Eqn. 9
S ▷ Set of schedulable task indices
Determine Γ(λ_max) ▷ Using Eqn. 10
Perform RTA for Γ(λ_max)
if Γ(λ_max) is schedulable then
    λ_hi ← λ_max
else
    return FAILURE
repeat
    λ ← (λ_hi + λ_lo)/2
    SCHEDULABLE ← TRUE
    S' ← S
    forall τ_i ∈ Γ, i ∉ S do
        Perform RTA for τ_i(λ)
        if τ_i(λ) is schedulable then
            Add i to S'
        else
            SCHEDULABLE ← FALSE
    if SCHEDULABLE then
        λ_hi ← λ
    else
        λ_lo ← λ
        S ← S'
until λ_hi − λ_lo ≤ ε;
return λ_hi
```

The algorithm first performs RTA for $\Gamma(\lambda_{\max})$; if it is not schedulable, the algorithm fails. Otherwise, it performs binary search over values of λ in the range $[0, \lambda_{\max}]$: λ_{hi} (initialized to $\lambda_{\max}$) tracks the smallest value of λ tested for which $\Gamma(\lambda)$ is schedulable, while λ_{lo} (initialized to 0) tracks the largest tested value for which $\Gamma(\lambda)$ is *not* schedulable. A variable S tracks the indices of tasks in $\Gamma(\lambda_{\text{lo}})$ that *are* schedulable.[2] At each step, the algorithm performs RTA for those tasks $\tau_i \in \Gamma(\lambda)$ that are *not* in S. If all tasks are schedulable, λ_{hi} is decreased to the tested value of λ; otherwise, λ_{lo} is increased to the tested value of λ and S is updated to include those tasks for which RTA nonetheless succeeded. The algorithm terminates when the difference between λ_{hi} and λ_{lo} does not exceed ϵ, at which point it is guaranteed that $\lambda_{\text{hi}} < \lambda^* + \epsilon$ for optimal λ^*, since $\lambda^* > \lambda_{\text{lo}}$.

Running Time: Since algorithm ELASTIC-FP-BS requires RTA to be performed for all tasks in $\Gamma(\lambda_{\max})$ prior to the binary search, in the worst case $\lceil \log_2(\lambda_{\max}/\epsilon) \rceil + 1$ total calls are made to RTA. The use of the variable S to track tasks already known to be schedulable for smaller values of λ may improve the execution time for some task sets; indeed, if $\lambda^* > \lambda_{\max} - \epsilon$, and if RTA determines schedulability for all but one task at $\lambda = \lambda_{\max}/2$, then binary search will only proceed upward, and RTA will only need to be performed for a single task at each checked value of λ thereafter. In this case, RTA for a *single* task is performed only

$$\left\lceil \log_2 \left(\frac{\lambda_{\max}}{\epsilon} \right) \right\rceil + 2n - 1$$

times. This is more efficient than Algorithm 2 (ELASTIC-FP-EFFICIENT) if:

$$\left\lceil \log_2 \left(\frac{\lambda_{\max}}{\epsilon} \right) \right\rceil + n < \left\lceil \frac{\lambda_{\max}}{\epsilon} \right\rceil \tag{13}$$

For ϵ chosen such that $\lambda_{\max}/\epsilon = 1000$, this is more efficient for systems of fewer than 990 tasks.

On the other hand, if $\lambda^* < \epsilon$, binary search will only proceed downward, so S remains empty, and so RTA must be performed for *all* tasks in Γ for each tested value of λ. In this case, RTA must be performed (i.e., RTA will succeed for each value of λ checked), in which case the analysis must be performed for each elastic task $\tau_i \in \Gamma$. In this case, RTA for a single task is performed

$$\left(\left\lceil \log_2 \left(\frac{\lambda_{\max}}{\epsilon} \right) \right\rceil + 1 \right) \times n$$

times. This is more efficient than Algorithm 2 (ELASTIC-FP-EFFICIENT) only if:

$$\left\lceil \log_2 \left(\frac{\lambda_{\max}}{\epsilon} \right) \right\rceil \times n + 1 < \left\lceil \frac{\lambda_{\max}}{\epsilon} \right\rceil \tag{14}$$

For ϵ chosen such that $\lambda_{\max}/\epsilon = 1000$, this is more efficient only if the system has fewer than 100 tasks. In Section VII, we evaluate both algorithms to compare their performance in the context of randomly-generated synthetic task sets.

VI. An MIQP Representation

In this section we describe how the problem of finding the value λ_i^* representing the minimum amount of compression to apply to a fixed-priority, preemptive task system Γ to guarantee

[2] S can be implemented as a bitmask or an array, allowing $\mathcal{O}(1)$ checking and insertion for a given task index.

schedulability of a *single* task τ_i can be formulated as a mixed integer quadratic program (MIQP). We then use this result to present an algorithm that finds the optimal compression value λ^* for the complete task system.

A. Formulating the MIQP

We build upon the mixed integer linear programming representation of RTA given in [10]. In [11], it is shown that for a fixed-priority, preemptive task system Γ to be schedulable, it is necessary and sufficient that for each $\tau_i \in \Gamma$, there exists some value of $t \leq D_i$ for which:[3]

$$t \geq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil \times C_j \tag{15}$$

The minimum value of t satisfying this condition for τ_i is the *response time* of the task. However, unlike in [10], we do not seek to find the minimum value of t for each task. Instead, for a *single* task τ_i, we seek to find the minimum value of λ_i for which there exists a $t < D_i$ satisfying Eqn. 15 for $\Gamma(\lambda)$. In other words, it must satisfy:

$$t \geq C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j(\lambda_i)} \right\rceil \times C_j \tag{16}$$

for $T_j(\lambda_i)$ as defined in Eqn. 10. The MIQP problem is formulated as follows:

1) For task τ_i, we define a real-valued variable t_i representing some value of t for which Eqn. 15 holds. Unlike in [10], where the corresponding R_i is non-negative, we restrict t_i to be positive: if $t_i = 0$ satisfies Eqn. 15, then $C_i = 0$, in which case the task can be ignored. This becomes an important distinction in a later step.
2) We specify the constraint:
$$t_i \leq D_i \tag{17}$$
3) As in [10], for each $j \in \{1, \ldots, i-i\}$, we define a non-negative *integer* variable Z_{ij} that represents the term $\lceil t_i / T_j(\lambda_i) \rceil$.
4) As in [10], to enforce this intended interpretation on the Z_{ij} variables, we must add constraints of the form $Z_{ij} \geq t_i / T_j(\lambda_i)$ for each $j \in \{1, \ldots, i-i\}$. Since Z_{ij} is specified to be an integer variable, this will respect the ceiling operator that appears in Eqn. 15.

   From the formulation of $T_j(\lambda_i)$ in Eqn. 10, we can express the term $\lceil t_i / T_j(\lambda_i) \rceil$ as:
   $$\max\left(\left\lceil \frac{t_i}{T_j^{\max}} \right\rceil, \left\lceil \frac{t_i (C_j - \lambda_i E_j T_j^{\min})}{C_j T_j^{\min}} \right\rceil \right)$$
   This requires two linear constraints for each Z_{ij}:
   $$\forall j \in \{1, \ldots, i-1\},\ Z_{ij} \geq \left(\frac{t_i}{T_j^{\max}} \right) \tag{18}$$
   $$\forall j \in \{1, \ldots, i-1\},\ Z_{ij} \geq \left(\frac{t_i (C_j - \lambda_i E_j T_j^{\min})}{C_j T_j^{\min}} \right)$$
   Because t_i is itself a variable, and λ_i is the value we want to minimize ultimately, we rewrite this expression:
   $$0 \leq Z_{ij} - \left(\frac{1}{T_j^{\min}} \right) t_i + \left(\frac{E_j}{C_j} \right) t_i \lambda_i \tag{19}$$
   This is a quadratic constraint, as it contains the term $t_i \lambda_i$.
5) As in [10], we add a final constraint for Eqn. 16:
$$C_i + \sum_{j=1}^{i-1} Z_{ij} \times C_j \leq t_i \tag{20}$$
6) To find the minimum value of λ_i for which the problem can be satisfied, we add the following **objective function**:
$$\textbf{minimize } \lambda_i \tag{21}$$

Recall that in Step 1 of the MIQP, t_i is restricted to the positive reals; this implies that if a solution exists, λ_i is well-defined for any value of t_i.

B. The Resulting Algorithm

By solving an MIQP, we can find the exact value of λ_i by which to compress a task system Γ to guarantee schedulability for an *individual* task according to RTA. We now present an algorithm that extends this approach to finding the optimal value, λ^*, representing the minimum amount of compression to guarantee schedulability of *all* tasks. The procedure is outlined in Algorithm 4.

Algorithm 4: Elastic-FP-MIQP(Γ)

```
Input: Elastic constrained-deadline task system Γ
Output: Smallest λ such that Γ(λ) is DM-schedulable
λ ← 0
forall τ_i ∈ Γ do
    Perform RTA for τ_i(λ)
    if τ_i(λ) is not schedulable then
        Construct MIQP
        Solve for λ_i*
        if MIQP infeasible then
            return FAILURE
        λ ← λ_i*
return λ
```

The algorithm initializes λ to 0. Each task in the system is checked for schedulability under the current compression level using RTA. Once a task τ_i is found that cannot be scheduled, the MIQP described in Section VI-A is constructed and solved for that task. If no solution is found, the minimum utilization constraints of its constituent tasks prevent the system from compressing to schedulability. Otherwise, λ is updated to λ_i^*, and the remaining tasks are checked in the same manner.

[3] Without loss of generality, we again assume tasks are indexed in decreasing order of priority.

Running Time: For a task system Γ of n tasks, algorithm ELASTIC-FP-MIQP must perform RTA for each task (the equivalent of performing RTA once over the complete task *system*). It must also solve, in the worst-case, n instances of the MIQP problem. Even the decision version of the problem (showing the existence of a λ that satisfies schedulability) is $\mathcal{NP}$-complete. Nonetheless, as we demonstrate in Section VII, this procedure can often efficiently find the optimal amount of compression using off-the-shelf solvers.

VII. EVALUATION

To evaluate the effectiveness and efficiency of the ELASTIC-FP-EFFICIENT (Alg. 2), ELASTIC-FP-BS (Alg. 3), and ELASTIC-FP-MIQP (Alg. 4) procedures, we run them over a large collection of randomly-generated constrained-deadline task sets. We track their execution, both in time and calls to RTA. We also analyze the overhead incurred by using these algorithms for online admission control in an embedded system running an ARM Cortex-A53 processor. Finally, we compare the compression values λ produced by each algorithm.

A. Generating Task Sets

To evaluate elastic scheduling for constrained-deadline tasks, we consider tasks that *begin* with a relative deadline D_i equal to their period T_i. Tasks individually have a utilization not exceeding 1, but the task systems *as a whole* are not schedulable due to their joint utilizations exceeding the utilization bound of the system. To accommodate such a task system, each task has its *period* extended while its *deadline* remains the same. We generate task sets of size $[10{-}100]$ in steps of 10. For each size, we consider total utilizations in the range $[1.0{-}2.0]$ in steps of 0.1. For each combination of size/utilization we generate 100 sets of tasks (for a total of 11 000) according to the following methodology:

1) Uncompressed task periods $T_i^{\min}$ are generated using a log-uniform distribution (per [12]) in $[1{-}1000]$.
2) Task deadlines D_i are set equal to $T_i^{\min}$.
3) Tasks are sorted according to increasing deadline (decreasing priority under DM scheduling).
4) The total utilization of the task system is partitioned among tasks using the UUniSort technique (a more elegant version of UUniFast) [13], such that each task is assigned a utilization $U_i^{\max}$.
5) Execution time is assigned according to $C_i{=}U_i^{\max}{\times}T_i^{\min}$.
6) A minimum utilization $U_i^{\min}$ is assigned to each task so that the *total* utilization cannot exceed 0.69, the Liu and Layland schedulability bound of a rate-monotonic implicit-deadline task system [14]. To do so, we define a constant $s = 0.69/U_{\text{SUM}}$, where U_{SUM} is the total utilization of the task set. We then obtain $U_i^{\min}$ by multiplying each task's $U_i^{\max}$ by a random value uniformly selected from the range $[0{-}s]$. On average, the total minimum utilization is expected to be 0.345, with a narrower distribution for larger task sets. This is illustrated in Fig. 1.

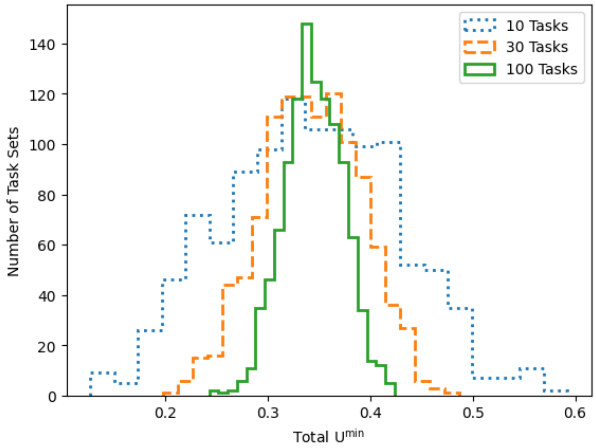


Fig. 1: Distributions of total minimum utilizations from 1100 randomly-generated task sets each of size 10, 30, 100.

7) The maximum period is derived as $T_i^{\max} = U_i^{\min}/C_i$.
8) Elasticity E_i is uniformly selected in $[0{-}1]$.

B. Execution Efficiency

We implement the recursive RTA of Audsley et al. [7] and the three algorithms we consider in C++, into which we link version 8 [15] of the SCIP constraint integer program solver [8] to execute the MIQP.

Offline Elastic Scheduling: Elastic scheduling can be leveraged in situations where a task set is compressed online for scheduling on an overutilized target system. To consider this case, we evaluate the execution of our algorithms on a server with two Intel Xeon Gold 6130 (Skylake) processors running at 2.1 GHz, and with 32GB of memory.

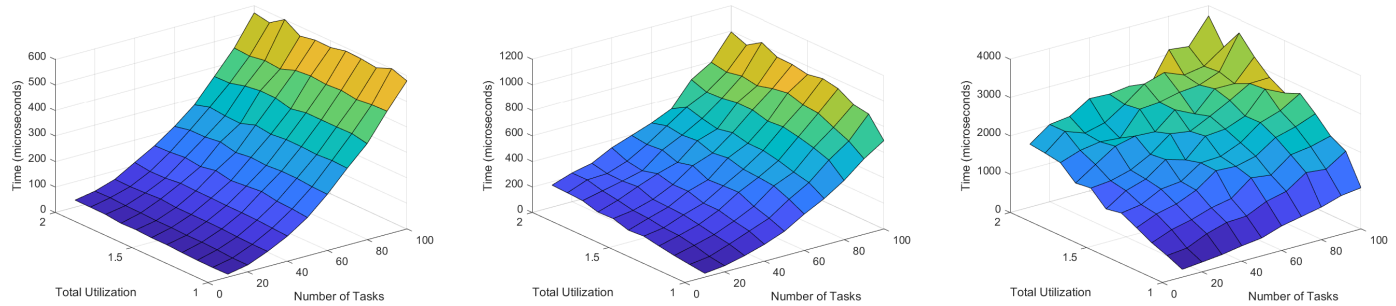
We begin by executing both ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS sequentially in a single-threaded environment over all 11 000 task sets. For each task set, we consider values of ϵ that divide the compression limit $\lambda^{\max}$ respectively 100, 1000, and 10 000 times. Mean execution times are illustrated in Fig. 2. Both algorithms are quite efficient, with mean execution times not exceeding 4 milliseconds for the task sets tested. For larger values of ϵ, iteration is more efficient on average than binary search; but as ϵ gets smaller, ELASTIC-FP-BS becomes faster. Worst observed execution times (WOET) and maximum total calls to RTA[4] are listed in Table I. As expected, algorithm running times are closely related to the number of calls to RTA.

Algorithm	$\lambda^{\max}/\epsilon$	WOET (ms)	Max RTA Calls
ELASTIC-FP-EFFICIENT	100	0.91	122
ELASTIC-FP-EFFICIENT	1000	2.14	1021
ELASTIC-FP-EFFICIENT	10 000	14.5	10 023
ELASTIC-FP-BS	100	2.45	700
ELASTIC-FP-BS	1000	3.47	1000
ELASTIC-FP-BS	10 000	3.97	1400

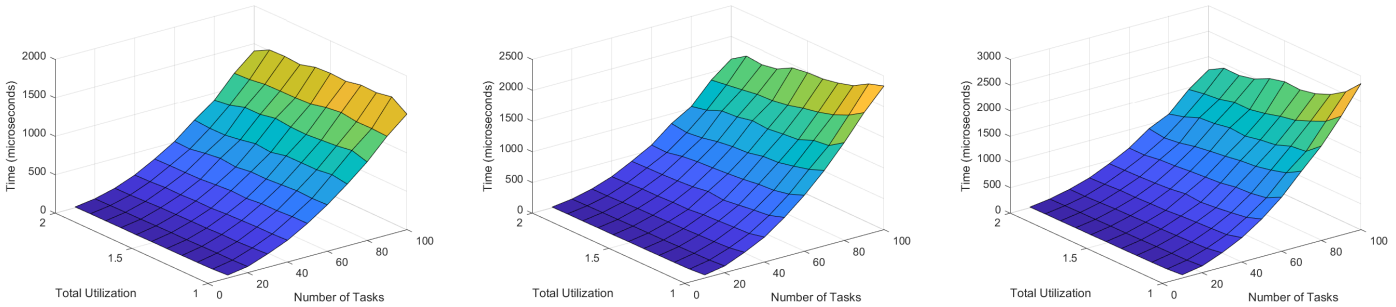
TABLE I: Algorithm performance comparison on Xeon-based server.

We then run ELASTIC-FP-MIQP over a subset of our generated task systems, limited to those with no more than 50

[4]Performing response time analysis for a *single* task counts 1 call to RTA.



(a) ELASTIC-FP-EFFICIENT, 100 (b) ELASTIC-FP-EFFICIENT, 1000 (c) ELASTIC-FP-EFFICIENT, 10 000



(d) ELASTIC-FP-BS, 100 (e) ELASTIC-FP-BS, 1000 (f) ELASTIC-FP-BS, 10 000

Fig. 2: Mean algorithm execution times on Intel Xeon Gold 6130.

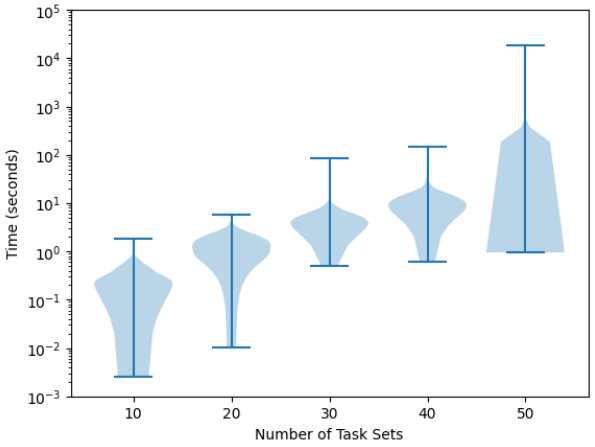


Fig. 3: Execution time distributions for ELASTIC-FP-MIQP.

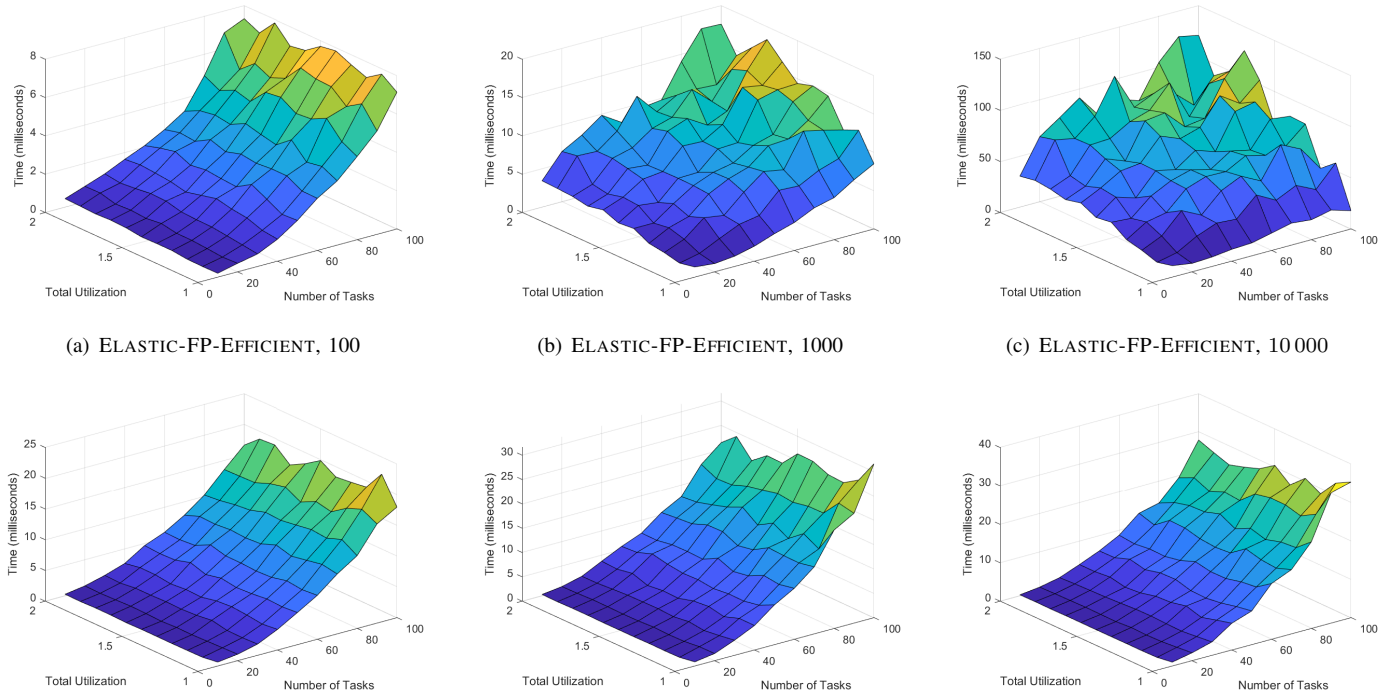
tasks. The solver is configured to execute in a single thread, which allows us to run separate instances of the algorithm sequentially on each of the unused logical cores on our server, splitting up the work of compressing the 5500 considered task systems. Execution time distributions are illustrated in Fig. 3. For task sets with 20 or fewer tasks, the algorithm consistently completes in under 10 seconds. Even with up to 40 tasks, the algorithm finds an optimal value of λ in under 3 minutes (and often under 10 seconds). However, for systems of 50 tasks, the solver may be slow to produce the optimal solution. For 95% of task sets with 50 tasks, an optimal solution is returned in under 80 seconds. However, three of the task sets take over 2 hours to complete, with the longest taking 5.1 hours.

Online Elastic Scheduling: Elastic scheduling can also be used for online adaption of task rates, e.g., when admitting new tasks on a fully-utilized system, or when task execution times change in response to dynamic exogenous factors. Despite its precision, the uncertain execution time to solve the MIQP make the ELASTIC-FP-MIQP algorithm unsuitable for online scheduling decisions in real-time systems. We consider instead the worst observed execution times of the ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS algorithms when running on an embedded system. We apply both algorithms to each of the $11\,000$ randomly-generated task sets, again using values of $\lambda^{\max}/\epsilon$ in $\{100, 1000, 10\,000\}$. We measure their execution times on a single core of a Raspberry Pi 3 Model B+, which has a Broadcom BCM2837B0 system-on-chip with an ARM Cortex-A53 running at 1.4GHz. For each combination of size/utilization, we take the maximum execution time among the 100 task sets tested; these are plotted in Fig. 4.

As before, the iterative algorithm outperforms the binary search for larger values of ϵ, but as ϵ decreases, ELASTIC-FP-BS begins to outperform ELASTIC-FP-EFFICIENT. For $\lambda^{\max}/\epsilon = 100$, ELASTIC-FP-EFFICIENT remains under 8 ms even for task sets of size 100; and for $\lambda^{\max}/\epsilon = 10\,000$, ELASTIC-FP-BS takes less than 36 ms. The low overhead suggests that either algorithm may be effective even for online compression of large fixed-priority constrained-deadline task sets on low-power embedded hardware.

C. *Effectiveness of the Approximate Algorithms*

When choosing between the three presented algorithms (ELASTIC-FP-EFFICIENT, ELASTIC-FP-BS, ELASTIC-FP-MIQP), one must consider the tradeoffs between *execution*



(a) ELASTIC-FP-EFFICIENT, 100 (b) ELASTIC-FP-EFFICIENT, 1000 (c) ELASTIC-FP-EFFICIENT, 10 000

(d) ELASTIC-FP-BS, 100 (e) ELASTIC-FP-BS, 1000 (f) ELASTIC-FP-BS, 10 000

Fig. 4: Maximum observed execution times on ARM Cortex-A53 (Raspberry Pi 3B+).

time and *accuracy*. We first compare the relative compression achieved by the three algorithms to illustrate when approximation may be sufficient. We define the metric

$$\delta = \frac{\lambda - \lambda^*}{\lambda^{\max}}$$

representing the difference between the compression value λ returned by an approximate algorithm and the optimal λ^* returned by ELASTIC-FP-MIQP, normalized by $\lambda^{\max}$. We compare values of δ between ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS for each of the three values of $\epsilon/\lambda^{\max}$ considered. The relative distributions are plotted in Fig. 5. To achieve the same range of values in the horizontal axes of each plot, we multiply by $\epsilon/\lambda^{\max}$, with values closer to 0 representing better agreement with λ^*. We observe that, while similar, ELASTIC-FP-BS tends to do better than ELASTIC-FP-EFFICIENT. Since it is also more efficient for finer granularity of ϵ, this likely makes it the preferred choice among the two approximate algorithms.

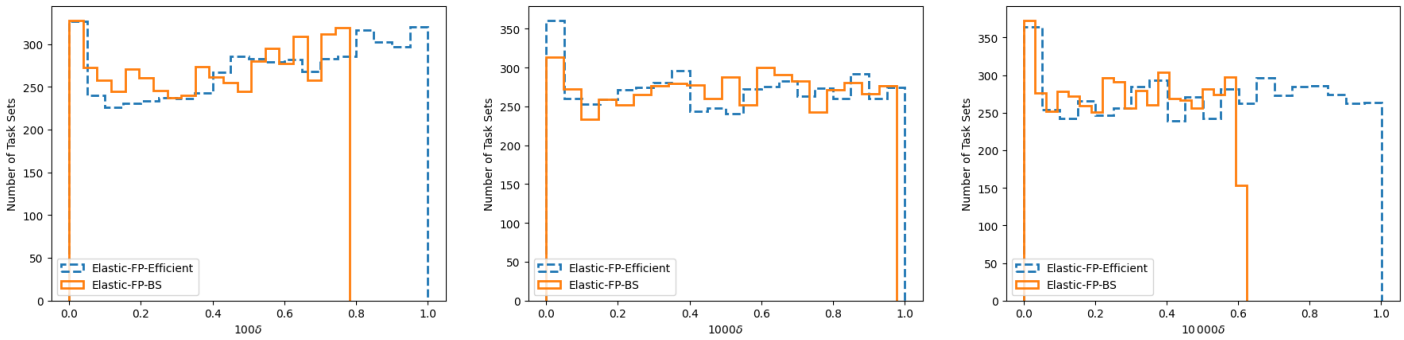
To quantify the effect of using an approximate algorithm to enact period compression over an elastic task set, we next compare the relative periods achieved by each algorithm. We define the metric $\theta_i = T_i(\lambda)/T_i(\lambda^*)$ representing the ratio of a task's period when compressed by an approximate algorithm to the period under optimal compression. We compare values of θ_i between ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS for each of the three values of $\epsilon/\lambda^{\max}$ considered. Results for the 162 420 total tasks for which compression was achieved are summarized in Table II.

We observe that the relative overcompression of the ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS algorithms tends to be small (under 10%), especially for large values of $\lambda^{\max}/\epsilon$. Nevertheless, occasionally tasks are significantly overcompressed. Even for $\lambda^{\max}/\epsilon$=10 000, ELASTIC-FP-EFFICIENT and ELASTIC-FP-BS overcompress 10 and 5 task periods respectively by over $10\times$.

VIII. CONCLUSION

In this work, we have extended uniprocessor elastic scheduling to fixed-priority, constrained-deadline tasks. We have presented three algorithms: ELASTIC-FP-EFFICIENT, which applies compression iteratively using step sizes of a tunable value ϵ; ELASTIC-FP-BS, which performs a binary search over the range of possible compression values, also using a precision of ϵ; and ELASTIC-FP-MIQP, which formulates the problem of finding the *exact* amount of compression to apply as a mixed integer quadratic program.

We have demonstrated that both approximate algorithms are highly efficient. Even for systems of 100 tasks and small values of ϵ, ELASTIC-FP-BS enables compression on the order of tens of milliseconds on a Raspberry Pi 3 Model B+. We also observed that, when compared to the optimal compression achieved by the MIQP approach, the approximate algorithms overcompress periods by less than 10% for 99.5% of tasks tested. However, in very rare corner cases ($< 0.006\%$ of tested tasks), the iterative algorithms overcompress periods by more than $10\times$ the optimal compression. Therefore, for offline scheduling decisions, ELASTIC-FP-MIQP may still remain a better choice. Even for task sets of size 50, the



(a) $\epsilon/\lambda^{\max} = 100$ (b) $\epsilon/\lambda^{\max} = 1000$ (c) $\epsilon/\lambda^{\max} = 10\,000$

Fig. 5: Relative distance from optimal of λ returned by approximate algorithms.

	ELASTIC-FP-EFFICIENT			ELASTIC-FP-BS		
θ	100	1000	10 000	100	1000	10 000
$[1, 1.1)$	127 807	155 190	161 590	132 184	155 354	161 928
$[1.1, 2)$	26 542	6 233	740	23 756	6 061	442
$[2, 10)$	6 994	866	80	5 602	876	45
$[10, 100)$	980	122	10	795	120	5
≥ 100	97	9	0	83	9	0

TABLE II: Relative overcompression of tasks by approximate algorithms.

algorithm completed in under 1.5 minutes 95% of the time, though we did observe 3 task sets for which the MIQP took over 2 hours to finish.

As future work, we intend to extend constrained-deadline elastic scheduling (including the PDA-based approach in [6] for EDF scheduling) to consider multiprocessor and distributed systems and computationally elastic tasks for which workloads, instead of periods, are compressed. We also plan to investigate whether an MIQP can be formulated to find optimal compression for dynamic-priority algorithms (e.g., EDF). Extensions to the MIQP may prove useful for handling additional constraints imposed under different system models.

REFERENCES

[1] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proc. of IEEE Real-Time Systems Symposium*, 1998. [Online]. Available: https://doi.org/10.1109/REAL.1998.739754

[2] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Transactions on Computers*, vol. 51, no. 3, pp. 289–302, Mar. 2002. [Online]. Available: http://dx.doi.org/10.1109/12.990127

[3] T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling," in *Proc. of IEEE International Real-Time Systems Symposium*, 2006, pp. 236–245. [Online]. Available: https://doi.org/10.1109/RTSS.2006.24

[4] ——, "Generalized elastic scheduling for real-time tasks," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 480–495, Apr. 2009. [Online]. Available: https://doi.org/10.1109/TC.2008.175

[5] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Syst.*, vol. 2, no. 4, p. 301–324, oct 1990. [Online]. Available: https://doi.org/10.1007/BF01995675

[6] S. Baruah, "Improved uniprocessor scheduling of systems of sporadic constrained-deadline elastic tasks," in *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS 2023)*. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3575757.3575759

[7] N. Audsley, A. Burns, M. Richardson, and A. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," *IFAC Proceedings Volumes*, vol. 24, no. 2, pp. 127–132, 1991, iFAC/IFIP Workshop on Real Time Programming, Atlanta, GA, USA, 15-17 May 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017512835

[8] T. Achterberg, "SCIP: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, Jul 2009. [Online]. Available: https://doi.org/10.1007/s12532-008-0001-1

[9] M. Sudvarg, C. Gill, and S. Baruah, "Linear-time admission control for elastic scheduling," *Real-Time Systems*, vol. 57, no. 4, pp. 485–490, Oct 2021. [Online]. Available: https://doi.org/10.1007/s11241-021-09373-4

[10] S. Baruah and P. Ekberg, "An ILP representation of response time analysis," 2021, short note available from https://research.engineering.wustl.edu/~baruah/Submitted/2021-ILP-RTA.pdf.

[11] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 01 1986. [Online]. Available: https://doi.org/10.1093/comjnl/29.5.390

[12] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS workshop at the Euromicro Conference on Real-Time Systems*, Jul. 2010, pp. 6–11, 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems ; Conference date: 06-07-2010.

[13] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, no. 1–2, p. 129–154, May 2005. [Online]. Available: https://doi.org/10.1007/s11241-005-0507-9

[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, p. 46–61, jan 1973. [Online]. Available: https://doi.org/10.1145/321738.321743

[15] K. Bestuzheva *et al.*, "The SCIP Optimization Suite 8.0," Optimization Online, Technical Report, December 2021. [Online]. Available: http://www.optimization-online.org/DB_HTML/2021/12/8728.html