



Improved Elastic Scheduling Algorithms for Implicit-Deadline Tasks

Marion Sudvarg¹   

Washington University in St. Louis, United States

Christopher Gill   

Washington University in St. Louis, United States

Sanjoy Baruah   

Washington University in St. Louis, United States

Abstract

Elastic scheduling provides a framework under which the utilizations of recurrent tasks are reduced by increasing their periods in response to system overload. The original elastic scheduling model was proposed by Buttazzo et al. in 1998 for implicit-deadline tasks on a uniprocessor and decreases task utilizations to satisfy a schedulable utilization bound. In 2019, Orr and Baruah extended the framework to multiprocessor scheduling of implicit-deadline tasks. In this paper, we propose, analyze, and evaluate new elastic scheduling

algorithms for several of the scheduling policies considered in these prior works. In particular, (i) we evaluate an algorithm that we proposed as a short note in the Real-Time Systems journal and demonstrate that it allows for faster admission control than the algorithm of Buttazzo et al. when applied to uniprocessor and fluid scheduling. (ii) We also present faster elastic scheduling algorithms for partitioned EDF scheduling. Finally, (iii) we provide polynomial-time exact elastic scheduling algorithms for global EDF and global RM.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases real-time systems, elastic scheduling, scheduling algorithms

Digital Object Identifier 10.4230/LITES.10.2.2

Category Special Issue on Industrial Real-Time Systems

Funding This research was supported in part by NSF grants CPS-2229290 and CNS-2141256 and a Washington University seed grant.

Acknowledgements The authors would like to thank the anonymous reviewers for their helpful comments. Thanks also to the TPC Chairs of SIES 2024 (Daniel Casini, Qingxu Deng, and Zhishan Guo) for inviting us to submit our work to this special issue.

Received To be completed by Dagstuhl editorial office **Accepted** To be completed by Dagstuhl editorial office **Published** To be completed by Dagstuhl editorial office

1 Introduction

Elastic real-time scheduling models provide a framework for dynamic task adaptation to guarantee schedulability even on an overloaded system. First proposed by Buttazzo et al. [12, 13], elastic scheduling allows tasks to decrease (“compress”) their utilizations by increasing invocation periods or reducing computational workloads (e.g., with imprecise computations [19] or early termination of anytime algorithms [14]). Each task is characterized by a maximum utilization representing the service level required for its desired mode of execution given sufficient computational resources. Tasks with flexible execution requirements — the so-called “elastic” tasks — are each assigned a parameter (its “elasticity”) representing its relative adaptability. On an overloaded system,

¹ Corresponding author



each elastic task’s utilization is reduced in proportion to its elasticity until the system becomes schedulable. Elastic tasks are also characterized by minimum utilizations representing the service level necessary to maintain correct or safe execution; each task’s utilization cannot be compressed below its minimum value.

While elastic scheduling models are therefore useful for adjusting a predefined set of tasks for execution on a resource-constrained system, the original elastic scheduling model of Buttazzo et al. was primarily intended to enable *online adaptation* in dynamic and open systems, e.g., in response to admission of new tasks or changes in available computational resources [12, 13]. Therefore, it is important for elastic scheduling algorithms to be *efficient* (i.e., provide bounded-time complexity guarantees) for online execution while preserving quality of service to the extent possible (i.e., tasks should be compressed only as much as needed to maintain schedulability).

With these two concerns in mind, this paper aims to analyze and improve upon existing approaches to elastic scheduling for sets of **sequential tasks with implicit deadlines** to be scheduled on both uniprocessor and multiprocessor systems. Prior algorithms for these types of task systems can be broadly classified into two categories. For scheduling algorithms with a utilization bound, a quadratic-time algorithm proposed by Buttazzo et al. [12, 13] for uniprocessor elastic scheduling finds an exact solution; this same algorithm was applied to multiprocessor fluid scheduling by Orr and Baruah [22, 21]. For multiprocessor scheduling algorithms where analysis does not simply check total utilization (e.g., partitioned EDF, global EDF, and global RM), Orr and Baruah proposed to iteratively increase the “amount” of utilization compression applied to the task system. At each level of compression, if the system is determined to be schedulable, the algorithm terminates; otherwise, compression is increased. Such algorithms are tunable in their precision: a smaller increase in compression at each iteration allows a more precise result, but increases the algorithm’s running time.

1.1 Contributions

Three key insights can be leveraged to improve these algorithms. First, **an exact solution under a utilization bound can be obtained in quasilinear time**, or in linear time for admission control or changes in utilization bound. In a short note published in the Real-Time Systems Journal [26], we presented, but did not evaluate, an algorithm to do so. In Section 3, we measure its performance in comparison to the original quadratic-time algorithm of Buttazzo et al. and discuss its advantages. In Section 4, we extend this algorithm to partitioned EDF scheduling, for which some partitioning heuristics afford a schedulable utilization bound. We demonstrate that, although pessimistic, compressing to the bound provides substantial speedups over the iterative elastic scheduling algorithm from Orr and Baruah in [22, 21]. We argue that this tradeoff may be beneficial in online scenarios that require rapid mode switches, e.g., in mixed-criticality systems [10].

Second, for the inexact iterative search algorithms, **an amount of compression can be found by *binary*, rather than *linear* search**. Compression is lower-bounded by 0 and upper-bounded by the amount that takes all tasks to their minimum utilizations. By binary searching in this range, Section 4 demonstrates that we can find a result more quickly than linear search for partitioned EDF scheduling.

The utilization bounds for global EDF [15] and global RM [7] scheduling are not static, but instead include among their terms the maximum over the task utilizations, so the bounds change as utilizations are compressed. The algorithm of Buttazzo et al. therefore cannot be applied directly; instead, in [21], Orr and Baruah apply their iterative search technique. However, we notice that **by replacing the maximum-utilization task with a proxy task using transformed parameters, our improved utilization bound algorithm can be applied**. A challenge arises

since we don't know, a priori, *which* task will require the minimum utilization after compression. Nonetheless, we can apply the compression algorithm once for *each* task chosen as the proxy, taking the best consistent result. Sections 5.1 and 5.2 demonstrate this for global EDF and global RM scheduling.

1.2 Extensions to the Prior Work

This work is an **extension** of a paper that we presented at the IEEE International Symposium on Industrial Embedded Systems (SIES) conference in 2024 [27]. In addition to the results in that paper, it includes:

- A more complete background section, with reproductions of the original elastic scheduling algorithm from [11] and our improved algorithm originally presented in [26].
- More detailed execution time statistics for the comparison of uniprocessor elastic scheduling algorithms; in particular, mean times are plotted.
- Plots, which were omitted from [27] due to page limits, relating the execution times of those algorithms to total maximum and minimum utilization of the task set.
- Additional discussion of the implications of faster online adaptation; in particular, we compare the amount of utilization compression necessary for schedulability when task execution times are inflated to accommodate algorithm overheads.
- Expanded plots for partitioned EDF scheduling, showing comparisons of schedulability, execution time, and amount of compression achieved for each parameter combination used in randomly generating the evaluated task sets. In [27], parameters were aggregated in summary plots due to the length restriction.
- A more complete description of our elastic compression algorithm for global EDF scheduling.
- An extension of that algorithm to global RM scheduling.
- Experimental evaluations of our global EDF and global RM elastic scheduling algorithms; evaluation of even the global EDF algorithm was absent from [27].

1.3 Organization

The remainder of this paper is organized as follows. In Section 2, we provide background on implicit-deadline elastic scheduling. In Section 3, we evaluate our improved (quasilinear/linear-time) algorithm from [26] compared to the original quadratic algorithm [12, 13] in the context of uniprocessor and multiprocessor fluid scheduling. In Section 4, we propose and evaluate faster approaches to elastic scheduling for partitioned EDF. In Section 5, we propose and evaluate exact algorithms for global EDF and global RM. Section 6 concludes the paper and discusses future work.

2 Background

2.1 Elastic Scheduling with Utilization Bounds

The elastic model for implicit-deadline tasks [12, 13] 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 initial value of this parameter needs to be assigned prior to run-time).

108 ■ E_i : An elastic constant, representing “the flexibility of the task to vary its utilization” [12].

109 Under the original model proposed by Buttazzo et al. [12, 13], the elastic model was applied
 110 to uniprocessor scheduling algorithms with utilization bounds, e.g., earliest deadline first (EDF)
 111 scheduling with its bound of 1, or rate monotonic (RM) scheduling under Liu and Layland’s
 112 bound [18]. It was since extended by Orr and Baruah [22] to multiprocessor fluid scheduling [1]
 113 where the utilization bound is equal to the number of processors m .

114 Under these models, a task system $\Gamma = \{\tau_1, \dots, \tau_n\}$ has a total uncompressed utilization $U_{\text{SUM}}^{\text{max}}$
 115 expressed as

$$116 \quad U_{\text{SUM}}^{\text{max}} = \sum_{i=1}^n U_i^{\text{max}} \quad (1)$$

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

- 120 1. $\sum_i U_i = U_D$, i.e., total utilization is set to the schedulable bound.
- 121 2. Any task for which $E_i = 0$ is considered inelastic; this is equivalent to the case that $U_i^{\text{min}} =$
 122 U_i^{max} .
- 123 3. For all other tasks τ_i and τ_j , if $U_i > U_i^{\text{min}}$ and $U_j > U_j^{\text{min}}$, then U_i and U_j must satisfy the
 124 relationship²

$$125 \quad \left(\frac{U_i^{\text{max}} - U_i}{E_i} \right) = \left(\frac{U_j^{\text{max}} - U_j}{E_j} \right) \quad (2)$$

126 A task system Γ for which such U_i exist for all tasks is said to be *feasible*.

127 Intuitively, this model represents each task as a spring, with a length corresponding to the
 128 utilization U_i and an elasticity corresponding to E_i . The total length of the springs, placed
 129 end-to-end, represents U_{SUM} . If this exceeds U_D , the schedulable bound, a force is applied to the
 130 system that compresses each task’s utilization U_i proportionally to its elasticity, subject to the
 131 constraint³ that the utilization remains no less than the specified minimum U_i^{min} . This physical
 132 analogy is illustrated in Figure 1.

133 Compression is often realized by adjusting each task’s period T_i according to its new utilization,
 134 i.e., $T_i = C_i/U_i$. Some work has also explored tasks for which computational workloads can be
 135 decreased ($C_i = T_i \cdot U_i$), e.g., by reducing the quantity of input data to process or by forcing an
 136 iterative anytime algorithm to terminate early [23, 25, 28].

137 2.1.1 An Overview of the Algorithm of Buttazzo et al.

138 Let Γ denote a feasible task system with $E_i > 0$ for all tasks⁴ $\tau_i \in \Gamma$, and consider the U_i values
 139 that satisfy the above conditions. The tasks in Γ may be partitioned into two classes — Γ_{VARIABLE}
 140 (those tasks for which $U_i > U_i^{\text{min}}$, and which can therefore have their utilizations compressed
 141 further if necessary) and Γ_{FIXED} (those for which $U_i = U_i^{\text{min}}$; i.e., their utilizations are now fixed).

142 It has been shown [12, Eqn. 8] that for each $\tau_i \in \Gamma_{\text{VARIABLE}}$, the utilization U_i takes the value

$$143 \quad U_i = U_i^{\text{max}} - \left(\frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}} \right) \times E_i \quad (3)$$

² For tasks τ_i having $E_i = 0$, $U_i = U_i^{\text{min}}$, and therefore the relationship needs not be satisfied.

³ This statement holds true for inelastic tasks, as $E_i = 0$ implies $U_i^{\text{min}} = U_i^{\text{max}}$, and therefore the utilization is not reduced.

⁴ Tasks τ_i with $E_i = 0$ must have $U_i \leftarrow U_i^{\text{max}}$; we assume this is done in a pre-processing step, with U_D updated to reflect the remaining available utilization.

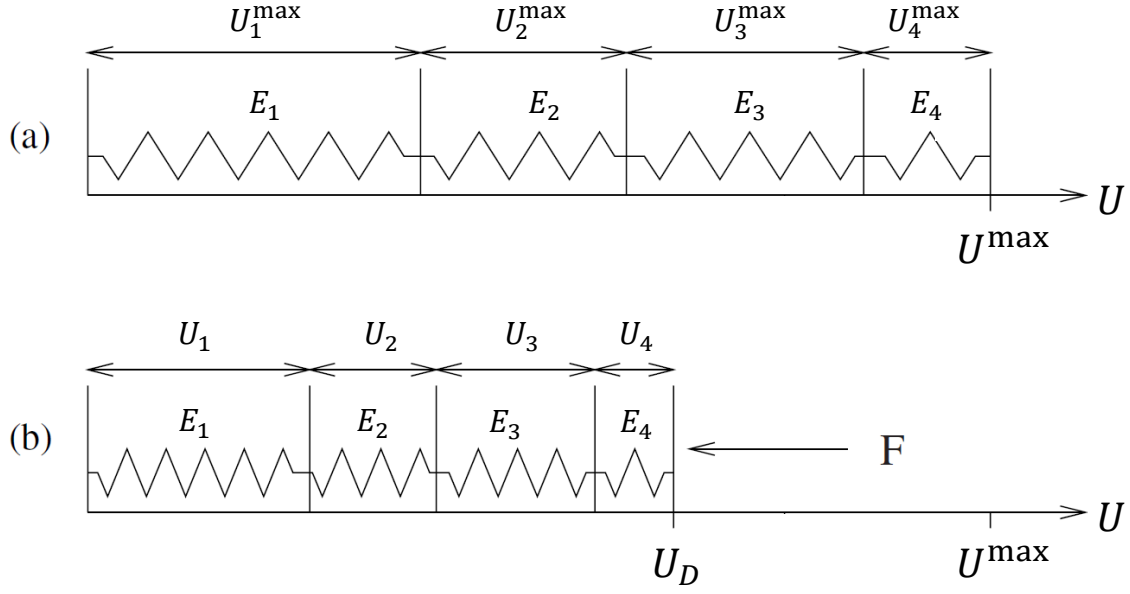


Figure 1 The physical spring analogy of the elastic model of Buttazzo et al., reproduced from [11, Figure 9.27] with modifications to the labels. (a) shows the uncompressed task set, while (b) illustrates the application of elastic compression to achieve schedulability.

where

$$U_{\text{SUM}} = \sum_{\tau_i \in \Gamma_{\text{VARIABLE}}} U_i^{\text{max}} \quad (4)$$

and

$$E_{\text{SUM}} = \sum_{\tau_i \in \Gamma_{\text{VARIABLE}}} E_i \quad (5)$$

respectively denote the sum of the U_i^{max} parameters and the E_i parameters of all the tasks in Γ_{VARIABLE} , and

$$\Delta = \sum_{\tau_i \in \Gamma_{\text{FIXED}}} U_i^{\text{min}} \quad (6)$$

denotes the sum of the U_i^{min} parameters⁵ of all the tasks in Γ_{FIXED} .

Given a set of elastic tasks Γ , the algorithm of [12, Figure 3] starts out computing U_i values for the tasks assuming that they are all in Γ_{VARIABLE} — i.e., their U_i values are computed according to Equation 3. If any U_i so computed is observed to be smaller than the corresponding U_i^{min} then ① that task is moved from Γ_{VARIABLE} to Γ_{FIXED} ; ② the values of U_{SUM} , E_{SUM} , and Δ are updated to reflect this transfer; and ③ U_i values are recomputed for all the tasks.

The process terminates if no computed U_i value is observed to be smaller than the corresponding U_i^{min} . It is easily seen that one such iteration (i.e., computing U_i values for all the tasks) takes

⁵ Observe that Δ equals the amount of utilization that is allocated to the tasks in Γ_{FIXED} ; therefore $(U_D - \Delta)$ represents the amount available for the tasks in Γ_{VARIABLE} , and $(U_{\text{SUM}} - (U_D - \Delta))$ the amount by which the cumulative utilizations of these tasks must be reduced from their desired maximums. As shown in the RHS of Equation 3, under elastic scheduling this reduction is shared among the tasks in proportion to their elasticity parameters: τ_i 's share is (E_i/E_{SUM}) .

2:6 Improved Elastic Scheduling Algorithms for Implicit-Deadline Tasks

159 $\mathcal{O}(n)$ time. Since an iteration is followed by another only if some task is moved from Γ_{VARIABLE} to
 160 Γ_{FIXED} and there are n tasks, the number of iterations is bounded from above by n . The overall
 161 running time for the algorithm is therefore $\mathcal{O}(n^2)$.

162 In his hard real-time computing systems textbook, Buttazzo presents a corrected and improved
 163 version of the algorithm that avoids explicitly maintaining separate lists to track Γ_{VARIABLE} and
 164 Γ_{FIXED} [11, Figure 9.29]. Nonetheless, the overall running time for the algorithm remains quadratic
 165 in the number of tasks. For reference, we present a modified version of this procedure in Algorithm 1.
 166 Notation has been modified to match our own.

■ **Algorithm 1** Elastic_Compression(Γ, U_D) (adopted from [11, Figure 9.29])

```

1   $U_{\text{SUM}}^{\min} \leftarrow \sum_{\tau_i} C_i / T_i^{\max}$ 
2  if  $U_D < U_{\text{SUM}}^{\min}$  then return INFEASIBLE
3  forall  $\tau_i \in \Gamma$  do  $U_i \leftarrow C_i / T_i^{\max}$ 
4  do
5     $U_{\text{FIXED}} \leftarrow 0, U_{\text{VARIABLE}} \leftarrow 0, E_{\text{SUM}} \leftarrow 0$ 
6    forall  $\tau_i \in \Gamma$  do
7      if  $(E_i = 0)$  or  $(T_i = T_i^{\max})$  then
8         $U_{\text{FIXED}} \leftarrow U_{\text{FIXED}} + U_i$ 
9      else
10        $E_{\text{SUM}} \leftarrow E_{\text{SUM}} + E_i$ 
11        $U_{\text{VARIABLE}} \leftarrow U_{\text{VARIABLE}} + U_i$ 
12     end
13   end
14    $\text{OK} \leftarrow \text{TRUE}$ 
15   forall  $\tau_i \in \Gamma$  do
16     if  $E_i > 0$  or  $T_i < T_i^{\max}$  then
17        $U_i \leftarrow U_i^{\max} - (U_{\text{variable}} - U_D + U_{\text{FIXED}})E_i / E_{\text{SUM}}$ 
18        $T_i \leftarrow C_i / U_i$ 
19       if  $T_i > T_i^{\max}$  then
20          $T_i \leftarrow T_i^{\max}$ 
21          $\text{OK} \leftarrow \text{FALSE}$ 
22       end
23     end
24   end
25 while  $\text{OK} = \text{FALSE}$ 
26 return FEASIBLE

```

167 The same algorithm was also repurposed in [12] for admission control — i.e., for determining
 168 whether a new task seeking to join an already-executing system could be admitted without
 169 compromising feasibility, and if so, recomputing the utilization values for all tasks.

2.1.2 Our Improved Algorithm

In a short note in the Real-Time Systems journal [26], we presented an algorithm that provides better guarantees on running time in terms of computational complexity. We defined an attribute ϕ_i for each elastic task τ_i :

$$\phi_i \stackrel{\text{def}}{=} \left(\frac{U_i^{\max} - U_i^{\min}}{E_i} \right) \quad (7)$$

We proved in [26, Theorem 1] that in the algorithm of Buttazzo et al. in [12, Figure 3], tasks may be “moved” from Γ_{VARIABLE} to Γ_{FIXED} in order of their ϕ_i parameters. Assuming that the tasks are indexed such that $\phi_i \leq \phi_{i+1}$ for all $i, 1 \leq i < n$, one can simply make a *single* pass through all the tasks from τ_1 to τ_n , identifying, and computing U_i values for, all those in Γ_{FIXED} before any in Γ_{VARIABLE} . This can all be done in $\mathcal{O}(n)$ time with the procedure in [26, Algorithm 1]; we reproduce it here as Algorithm 2. The cost of sorting the tasks in order to arrange them according to non-increasing ϕ_i parameters is $\mathcal{O}(n \log n)$, and hence dominates the overall run-time complexity. Determining feasibility and computing the U_i parameters can therefore be done in $\mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$ time.

■ **Algorithm 2** Elastic_Implicit_Uniprocessor(Γ, U_D) (adopted from [26, Algorithm 1])

Input: A list Γ of elastic tasks sorted in non-decreasing order of their ϕ_i parameters (see Equation 7) and a desired utilization U_D

Output: Feasibility and the list Γ with computed U_i values

```

1   $U_{\text{SUM}} \leftarrow 0$ ;  $E_{\text{SUM}} \leftarrow 0$ ;  $\Delta \leftarrow 0$ 
2  forall  $\tau_i \in \Gamma$  do
3     $U_{\text{SUM}} = U_{\text{SUM}} + U_i^{\max}$ 
4     $E_{\text{SUM}} = E_{\text{SUM}} + E_i$ 
5  end
6  forall  $\tau_i \in \Gamma$  do
7    if  $\left( U_i^{\max} - \frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}} \times E_i \leq U_i^{\min} \right)$  then
8       $\triangleright$  Task  $\tau_i$  is no longer compressible — it's in  $\Gamma_{\text{FIXED}}$ 
9       $U_i \leftarrow U_i^{\min}$   $\triangleright$  Since  $\tau_i \in \Gamma_{\text{FIXED}}$ 
10      $\Delta \leftarrow \Delta + U_i^{\min}$   $\triangleright$  This additional amount of utilization is allocated to
        tasks in  $\Gamma_{\text{FIXED}}$ 
11     if  $(\Delta > U_D)$  then return INFEASIBLE  $\triangleright$  Cannot accommodate the minimum
        requirements
12      $U_{\text{SUM}} \leftarrow U_{\text{SUM}} - U_i^{\max}$   $\triangleright$  Since  $\tau_i$  is removed from  $\Gamma_{\text{VARIABLE}}$ 
13      $E_{\text{SUM}} = E_{\text{SUM}} - E_i$   $\triangleright$  As above — since  $\tau_i$  is removed from  $\Gamma_{\text{VARIABLE}}$ 
14   else
15      $\triangleright$  Remaining tasks are all compressible (i.e., in  $\Gamma_{\text{VARIABLE}}$ )
16      $U_i \leftarrow U_i^{\max} - \frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}} \times E_i$   $\triangleright$  As per Equation 3
17   end
18 end
19 return FEASIBLE

```

Admission control — determining whether it is safe to add a new task and recomputing all the U_i parameters if so — requires that the new task be inserted at the appropriate location in the

186 already sorted list of preexisting tasks. This can be achieved in $\mathcal{O}(\log n)$ time by implementing
 187 the list as a sorted iterable data structure. Once this is done, the U_i values can be recomputed
 188 in $\mathcal{O}(n)$ time by the same algorithm. Similarly, removing a task from the system and recomputing
 189 the U_i values also takes $\mathcal{O}(n)$ time. Furthermore, if U_D changes — e.g., in response to changes
 190 in available utilization due to dynamic resource reallocation — the sorted list of tasks and their
 191 parameters do not change, and so the U_i values can be updated in linear time.

192 Though we proved better asymptotic time complexity in [26], we did not evaluate the algorithm’s
 193 performance for realistic task sets. In Section 3, we perform this evaluation and extend the
 194 algorithm to fluid scheduling.

195 2.2 Scheduling Without a Utilization Bound

196 In addition to fluid scheduling, in [22], Orr and Baruah also developed elastic models for multi-
 197 processor partitioned EDF and global EDF scheduling. They later applied elastic scheduling to
 198 partitioned RM in a journal extension [21]. All of these scheduling policies have feasibility tests
 199 that are more involved than simply checking total utilization against a bound that is constant in
 200 the number of tasks. To deal with this, they observed that the degree by which compression is
 201 applied to a task system can be quantified by the relationship in Equation 2. In doing so, they
 202 introduce a term λ that is representative of this relationship, and express the utilization U_i of
 203 each task τ_i as:

$$204 \quad U_i(\lambda) \stackrel{\text{def}}{=} \max(U_i^{\max} - \lambda E_i, U_i^{\min}) \quad (8)$$

The value of λ beyond which the utilization U_i of task τ_i takes its minimum value U_i^{\min} can
 therefore be derived as:

$$U_i^{\min} = U_i^{\max} - \lambda E_i \quad \rightarrow \quad \lambda = \left(\frac{U_i^{\max} - U_i^{\min}}{E_i} \right)$$

205 which is equal to the value ϕ_i in Equation 7. As such, we may hereafter refer to ϕ_i interchangeably
 206 as λ_i^{\max} . For a complete set of tasks Γ we also denote the maximum compression that may be
 207 applied to the task system as:

$$208 \quad \lambda^{\max} \stackrel{\text{def}}{=} \max_{\tau_i} (\lambda_i^{\max}) \quad (9)$$

209 The problem of elastic scheduling under the model of Buttazzo et al. [12, 13] can therefore be
 210 reduced to the problem of finding the minimum value of λ for which a set of tasks are schedulable.
 211 For partitioned EDF and RM, global EDF and RM, and algorithm PriD, Orr and Baruah propose
 212 an approximate search technique that iterates over values of λ in the interval $[0, \lambda^{\max}]$ with some
 213 “granularity” ϵ . For each value of λ , they assess schedulability, terminating the search once the
 214 compressed task system is deemed schedulable [22, 21]. In this paper, we explore and propose
 215 alternatives to this idea for partitioned and global EDF and global RM scheduling.

216 2.2.1 Partitioned EDF

217 Under partitioned earliest deadline first (EDF) scheduling, each task is assigned to a single
 218 processor core, though each core may be assigned multiple tasks. On an individual core, jobs are
 219 prioritized according to their absolute deadlines — in other words, each core schedules its tasks in
 220 an EDF manner independently of the other cores. The problem of deciding whether a set of tasks
 221 is schedulable on m cores under partitioned EDF can be stated as follows:

222 Given a set Γ of n tasks τ_i , each having utilization U_i , is there a partition of tasks into
 m sets such that the sum of utilizations in any set does not exceed 1?

This is equivalent to the bin-packing problem, and is therefore NP-hard in the strong sense. Nonetheless, there exist heuristic algorithms to solve bin-packing problems, and Lopez et al. have compared several in the context of partitioned EDF scheduling [20]. For each value of λ tested, Orr and Baruah employ the first fit, worst fit, and best fit heuristics, with tasks τ_i considered in order of decreasing utilization $U_i(\lambda)$. If any one heuristic deems feasibility, the algorithm terminates.

For n tasks on m cores, sorting tasks and partitioning them with each heuristic takes at most $\Theta(n \log n + n \cdot m)$ time. As this must be performed for each tested value of λ — of which there are up to $(\lfloor \frac{\lambda^{\max}}{\epsilon} \rfloor + 1)$ — the overall complexity is $\Theta(\frac{\lambda^{\max}}{\epsilon} \cdot (n \log n + n \cdot m))$.

2.2.2 Global EDF

Under global EDF scheduling, if at any instant there are more active jobs than processors, those jobs with the earliest *absolute* deadlines are selected for execution. Goossens et al. showed [15, Theorem 5] that a set Γ of preemptive implicit-deadline tasks is schedulable on m identical processors with global EDF if:

$$\sum_{\tau_i \in \Gamma} U_i \leq m - (m - 1) \cdot \max_{\tau_i \in \Gamma} \{U_i\} \quad (10)$$

Because the utilization bound includes a term representing the maximum among task utilizations, and because that maximum may change (indeed, the *task* with the maximum utilization may change) as utilizations are compressed, the original algorithm of Buttazzo et al. cannot be applied directly. Orr and Baruah instead perform a linear search over the space of possible values of λ , terminating when Equation 10 holds true [22].

In Section 5.1, we propose and evaluate a polynomial-time algorithm that finds an exact solution, if one exists.

2.2.3 Global RM

Under global rate monotonic (RM) scheduling, if at any instant there are more active jobs than processors, those jobs of tasks with the shortest periods are selected for execution. Bertogna et al. showed [7, Theorem 5] that a set Γ of preemptive implicit-deadline tasks is schedulable on m identical processors with global RM if:

$$\sum_{\tau_i \in \Gamma} U_i \leq \frac{m}{2} \left(1 - \max_{\tau_i \in \Gamma} \{U_i\} \right) + \max_{\tau_i \in \Gamma} \{U_i\} \quad (11)$$

As with global EDF, the utilization bound includes terms with the maximum task utilization, which may change as utilizations are compressed. Orr and Baruah again perform a linear search with precision ϵ for the smallest λ that guarantees feasibility [21]. In Section 5.2, we present a polynomial-time exact algorithm for comparison.

3 Compressing to a Constant Utilization Bound

This section considers elastic scheduling applied to scheduling policies with utilization bound tests; in particular, we consider earliest deadline first (EDF) and rate monotonic (RM) scheduling on a uniprocessor and fluid scheduling on a multiprocessor.

3.1 Complexity of the Algorithm of Buttazzo et al.

As noted in Section 2.1, the original elastic scheduling algorithm [12, 13] has worst-case execution time complexity that is quadratic in the number of tasks. Buttazzo et al. note in [12] that this is due to the enforcement of constraints on minimum utilization. If tasks are not thus constrained, the algorithm can run in linear time. Intuitively, we may consider that some tasks, representing non-critical best-effort computation, need not be characterized with minimum utilizations. However, we note that *without* these constraints, the algorithm can assign negative utilizations.

► **Example 1.** Consider a set Γ of implicit-deadline elastic tasks to be scheduled by EDF on a uniprocessor and parameterized as follows.

Task τ_i	U_i^{\max}	E_i
τ_1	0.9	1
τ_2	0.9	1
τ_3	0.2	8

The total uncompressed utilization U_{SUM}^{\max} is 2.0, but the desired utilization is $U_D = 1.0$. Then, in the absence of a constraint U_i^{\min} , the utilization U_i of each task τ_i will be assigned according to Equation 3:

$$U_i = U_i^{\max} - \left(\frac{2.0 - 1.0}{E_{\text{SUM}}} \right) \times E_i = U_i^{\max} - \left(\frac{1}{10} \right) \times E_i$$

Computing for each task, we obtain $U_1 = U_2 = 0.8$ and $U_3 = -0.6$. While this set of assignments does achieve a total utilization of 1.0, these assignments are not valid: a *negative* utilization does not have semantic meaning.

Thus, the elastic problem *with* minimum utilization constraints U_i^{\min} is the only meaningful expression of the problem in the context of task scheduling, even if the constraints are set to 0 just for the purpose of enforcing non-negative utilization assignments. Therefore, the algorithm of Buttazzo et al. cannot be guaranteed to have better than quadratic time complexity in the number of tasks. On the other hand, our procedure in Algorithm 2 is quasilinear in the number of tasks, and linear for admission control or changes to the utilization bound. The remainder of this section compares the two algorithms empirically using synthetic task sets with randomly-generated parameters.

3.2 Performance Evaluation on a Uniprocessor

We now compare experimentally the performance of our improved algorithm for elastic scheduling of implicit-deadline tasks on a uniprocessor outlined in Algorithm 2 to that of Buttazzo et al. [11, Figure 9.29] listed in Algorithm 1.

3.2.1 Implementation

Evaluations are performed on a Raspberry Pi 3 Model B+, which has a Broadcom BCM2837B0 System on Chip (SoC) with a 4-core ARMv8 Cortex-A53 running⁶ at 700 MHz and 1GB of RAM. We used version 6.1.21 of the Linux kernel, compiled for the ARMv7l 32-bit ISA. We implement

⁶ The processor supports a CPU clock speed of 1.4 GHz. However, as noted in [9], this frequency cannot be sustained continuously, and may lead to throttling and instability. To maintain predictability, we boot the Raspberry Pi with a constant 700 MHz CPU clock speed, set the GPU to 250 MHz, and disable throttling. Details can be found at https://www.raspberrypi.com/documentation/computers/config_txt.html

both algorithms in C++ and quantify execution time performance by measuring elapsed processor cycles. We read directly from the cycle counter using a custom driver and kernel module that enables access to the ARM performance monitoring unit (PMU) from userspace. Algorithms are compiled statically using version 10.2.1 of the Gnu Compiler Collection (GCC) at optimization level 00; this allows us to avoid undesirable instruction reordering, especially around reads to the cycle counter. To avoid interference from other processes, we disable real-time throttling by writing `-1` to the file `/proc/sys/kernel/sched_rt_runtime_us`, isolate CPU core 3 from the scheduler, and run our algorithms on that core at the highest real-time priority under Linux's `SCHED_FIFO` scheduling class.

Each task τ_i is represented as a data structure (`struct`) containing single-precision floating-point representations of U_i^{\max} , U_i^{\min} , U_i , and E_i . The derived parameter ϕ_i from Equation 7 is also an attribute of the structure. For the following experiments, besides those of Section 3.2.5, we are only concerned with the assignment of U_i values; therefore we do not represent the WCET C_i or period T_i of any task.

For the algorithm of Buttazzo et al. (Algorithm 1), since we are considering only utilization assignments, and not period updates, we do not implement Line 18. Line 3 is also not implemented, as we assign $U_i \leftarrow U_i^{\max}$ when task parameters are generated. Line 20 is implemented as $U_i \leftarrow U_i^{\min}$ instead. Furthermore, all period-related checks (Lines 7, 16, and 19) are converted to the equivalent comparison of U_i to U_i^{\min} or U_i^{\max} .

For our improved algorithm (Algorithm 2), we compare three implementations:

1. The set of tasks Γ is implemented as an array (`std::vector`), which is sorted prior to executing the algorithm. Inserting or removing tasks takes linear time to move array elements (and, in the case of insertion, to find the location to insert to maintain sorted order).
2. The set of tasks Γ is implemented as a balanced binary tree (`std::set`), sorted by ϕ_i . Constructing the set takes quasilinear time, but subsequent insertion and removal requires only logarithmic time, while enabling sequential iteration over tasks in sorted order.
3. The set of tasks Γ is implemented as a linked list (`std::list`), sorted by ϕ_i . Removing a task takes constant time, but adding a task takes linear time to find the location to insert in sorted order.

3.2.2 Generating Task Sets

We generate sets Γ of tasks τ_i using the following methodology:

1. We consider sets of n tasks, generating 10 000 sets for each value of n in 2–50.
2. Each set of tasks has a total maximum utilization U_{SUM}^{\max} selected at random uniformly from $(1.0, 2.0]$ and a total minimum utilization U_{SUM}^{\min} selected at random uniformly from $(0.0, 1.0]$.
3. We apply the Dirichlet Rescale (DRS) algorithm [16] to distribute the total maximum utilization U_{SUM}^{\max} in an unbiased random fashion across the U_i^{\max} values for each individual task. We note that, in this context, the result should be equivalent to an application of the earlier UUniFast and UUniSort techniques [8].
4. We then apply the DRS algorithm to distribute the total minimum utilization U_{SUM}^{\min} across the individual U_i^{\min} values. DRS allows us to select these values uniformly from the space of selections satisfying the conditions that (i) the total $\sum_i U_i^{\min}$ equals the specified U_{SUM}^{\min} and (ii) each value U_i^{\min} does not exceed the corresponding U_i^{\max} .
5. Each task τ_i is assigned an elasticity E_i at random, selected uniformly from the range $(0, 1]$.

329 **3.2.3 Execution Time of Utilization Compression for Schedulability**

330 We execute our implementations of Algorithms 1 and 2 to compress each set of tasks to a total
 331 utilization of 1.0 (to be EDF-schedulable on a single processor). We measure execution time by
 332 reading directly from the cycle counter, reporting elapsed CPU cycles. We separately measure the
 333 initialization (“Init”) and compression (“Compress”) times for each algorithm. For the original
 334 procedure in Algorithm 1, initialization involves computing the U_{SUM}^{\min} value and checking whether
 335 it exceeds U_D , while compression is the **do . . . while** loop in the algorithm. For our improved
 336 algorithm in Algorithm 2, compression is the **forall** loop that iterates over tasks in order of their
 337 ϕ_i parameters. The dominant contribution to the algorithm’s execution time complexity is the
 338 sorting of tasks by their ϕ_i values; we therefore include in initialization time both the computation
 339 of U_{SUM} and E_{SUM} as well as the total time to calculate each task’s ϕ_i value and establish the
 340 sorted order. For the array and list, the sort is performed over the complete data structure; for
 341 the binary tree, we insert tasks individually as their ϕ_i values are calculated.

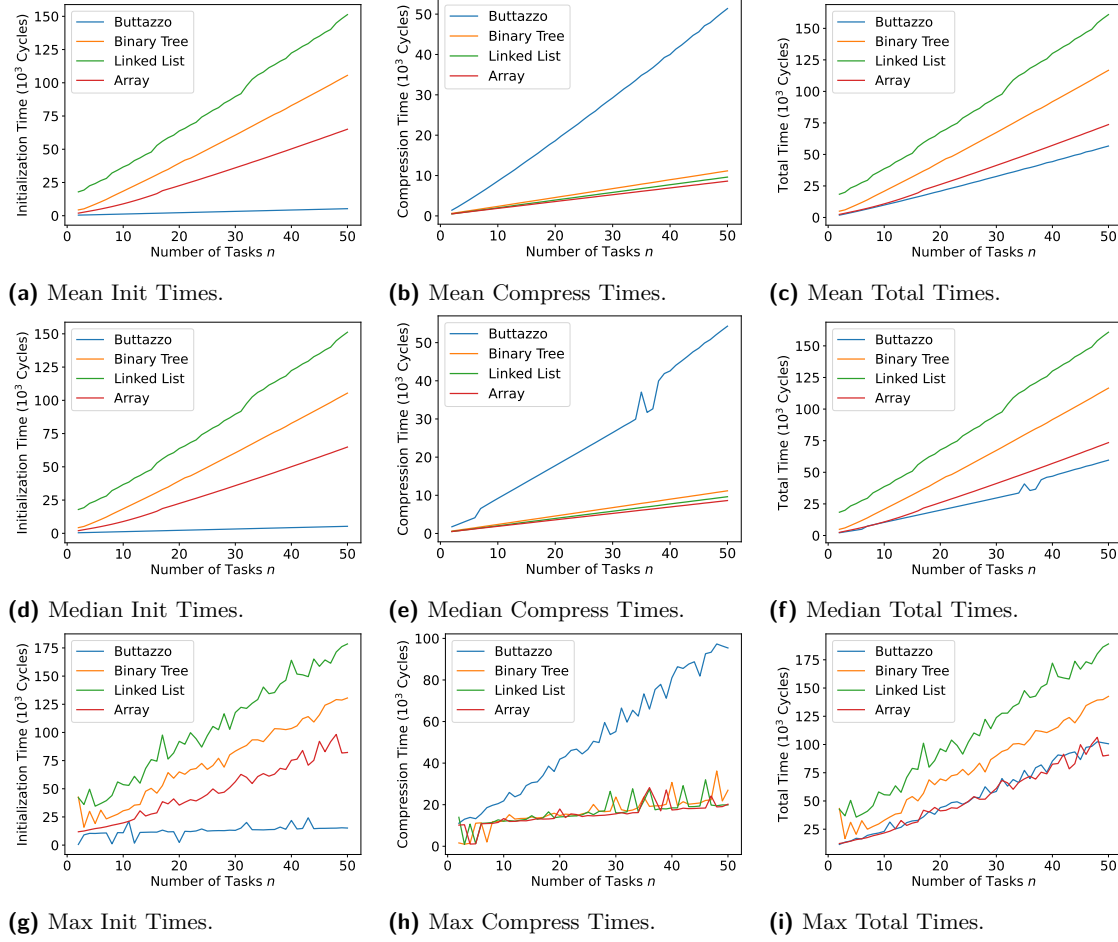
342 The mean, median, and maximum times for the 10 000 sets of tasks generated for each size n
 343 from 2–50 are reported in Figure 2. As expected, the time to initialize the algorithm of Buttazzo
 344 et al. is much faster than our improved algorithm, which has to sort tasks by their ϕ_i values. Of
 345 the three implementations of our algorithm, the linked list was the slowest to initialize, while the
 346 array was the fastest; we assume that this was due to the data locality and simplicity of managing
 347 the data structure.

348 Also, as expected, the compression time for the algorithm of Buttazzo et al. was much longer
 349 than for our algorithm. On average, the array tended to be the fastest, followed by the linked list,
 350 followed by the binary tree.⁷ This makes sense; while all three data structures enable linear time
 351 traversal, the array is the simplest to iterate (in fixed-size strides) and has the best data locality;
 352 the linked list is still simple, but requires following pointers between nodes, and does not have as
 353 good locality; and the binary tree requires even more complex pointer chasing.

354 Most interesting, we observe that our algorithm does *not* strictly dominate the original
 355 algorithm of Buttazzo et al. in total running time. In fact, in the average case, the original
 356 algorithm performs better because of the low initialization overhead. In the worst case, both the
 357 original algorithm and the array-based implementation of our algorithm dominate the other two
 358 implementations, but neither clearly dominates the other. This is partially explained by the fact
 359 that the compression times for the algorithm of Buttazzo et al. grow roughly linearly with the
 360 number of tasks, rather than quadratically, for sets of up to 50 tasks. Under non-pathological
 361 cases, the outer loop (Line 6 of Algorithm 2) of the original algorithm only runs a handful of
 362 times.

363 Nonetheless, we argue that our algorithm is better in practice. While there is not a clear
 364 advantage to using our algorithm to perform compression over a complete set of tasks, there is
 365 no clear *disadvantage* either. Furthermore, our algorithm performs better in situations where
 366 initialization has already happened, e.g. for online adjustment in response to changes in available
 367 utilization. In this case, when only compression needs to occur, associated overheads are summar-
 368 ized in Table 1. The worst execution times that we observed for the array-based implementation
 369 of our algorithm were 3.45× faster than those of the algorithm of Buttazzo et al. when just
 370 compressing tasks. Moreover, as we will demonstrate, there is a clear advantage to using our
 371 algorithm during online admission of a new task.

⁷ We do not observe a significant difference in the trends of worst-case compression times versus number of tasks among the three implementations.



■ **Figure 2** Performance scaling with number of tasks.

3.2.4 Execution Time of Task Admission

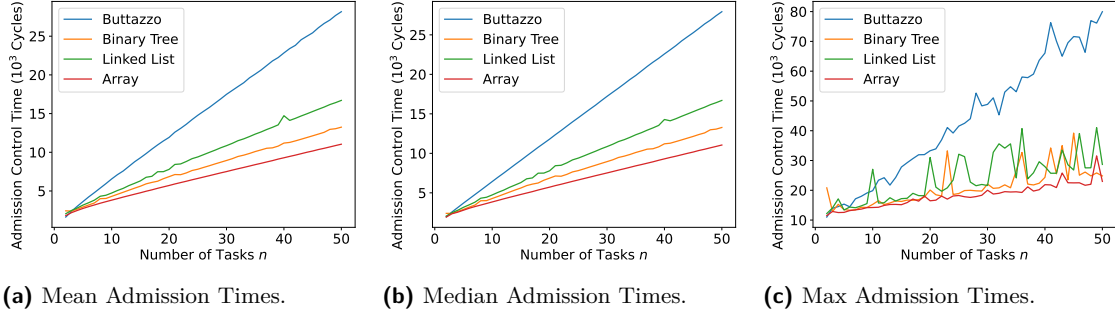
We modify our implementations of Algorithms 1 and 2 to measure admission of a single task. For the sets of n tasks of size 2–50 that we already generated, we apply each algorithm to the first $n - 1$ tasks. We then measure the time to compress them after adding the final task from the set. For the algorithm of Buttazzo et al., this requires rerunning the complete **do . . . while** loop. For our algorithm, this requires ① computing the value ϕ_i of the new task τ_i according to Equation 7, then ② adding its utilization and elasticity to U_{SUM} (Equation 4) and E_{SUM} (Equation 5) or Δ (Equation 6), as appropriate. The task is then ③ inserted into the sorted container, before finally ④ executing the **forall** loop in Algorithm 2.

Results are illustrated in Figure 3. We observe that, when admitting a new task, all implementations of our improved algorithm dominate the original algorithm of Buttazzo et al. for more than 3 tasks in the average case, and more than 10 tasks in the worst case. The array (which enables logarithmic time search for the location to insert the new task, then requires linear time to perform the insertion) performs the best on average, followed by the balanced binary tree (which allows logarithmic-time insertion, but requires pointer chasing), then the linked list (which allows constant-time insertion after linear time search for the insert location). Overheads and speedups are summarized in Table 2. The array-based implementation of our algorithm admits tasks $2.53\times$

2:14 Improved Elastic Scheduling Algorithms for Implicit-Deadline Tasks

	Buttazzo	Binary Tree	Linked List	Array
mean	51 380	11 157 (4.61 \times)	9 617 (5.34 \times)	8 616 (5.96 \times)
median	54 315	11 175 (4.86 \times)	9 629 (5.64 \times)	8 626 (6.30 \times)
maximum	97 342	36 231 (2.69 \times)	31 984 (3.04 \times)	28 202 (3.45 \times)

■ **Table 1** Greatest mean, median, and maximum **compression times** (cycles) observed for up to 50 tasks. Values in parentheses indicate speedup compared to the algorithm of Buttazzo et al.



■ **Figure 3** Execution time for admitting the n^{th} task.

389 faster than original algorithm in the worst case.

	Buttazzo	Binary Tree	Linked List	Array
mean	28 165	13 246 (2.13 \times)	16 704 (1.69 \times)	11 043 (2.55 \times)
median	27 931	13 274 (2.10 \times)	16 698 (1.67 \times)	11 053 (2.53 \times)
maximum	79 967	39 213 (2.04 \times)	41 044 (1.95 \times)	31 566 (2.53 \times)

■ **Table 2** Greatest mean, median, and maximum task **admission times** (cycles) observed for up to 50 tasks. Values in parentheses indicate speedup compared to the algorithm of Buttazzo et al.

390 3.2.5 Implications for Online Adaptation

391 During online adjustment of task utilizations, e.g., in response to admission of a new task, the
 392 overhead associated with computing new task utilizations must be accounted to avoid deadline
 393 misses. Full treatment of the several alternative methods for handling such transient overheads is
 394 outside the scope of this work; here we consider the usual approach in which scheduling and other
 395 operating system kernel overheads are added to the execution times of each task in the system.
 396 Under elastic scheduling, if the worst-case execution time of the selected compression algorithm
 397 is added to each task's execution time, then clearly *more* utilization compression is necessary to
 398 maintain schedulability in overload scenarios when a slower-running algorithm is used.

399 To highlight the implications of this, we compare the amount of utilization compression
 400 necessary to admit the last task of our task sets when adding the maximum overheads observed
 401 in Figure 3c for the algorithm of Buttazzo et al. and the array-based implementation of our
 402 algorithm. We quantify this using the value λ defined as in Equation 8. We restrict attention to
 403 sets of size 3–50 since the algorithm of Buttazzo et al. is slightly faster in the worst case when
 404 applied to a set of only 2 tasks.

405 Until now, we have considered only utilization compression in a general sense; thus, our
 406 randomly generated synthetic tasks from Section 3.2.2 are only characterized by acceptable

utilization ranges and elasticities. To consider the implications of algorithm overheads, we must also assign periods and execution times. To demonstrate the practical applicability of our approach in real-world scenarios, we draw period values from real-world automotive benchmarks. We assign minimum periods T_i^{\min} at random to each task from the distribution listed in [17, Table III].⁸ Execution times are then derived as $C_i = U_i^{\max} \cdot T_i^{\min}$ and maximum periods are computed as $T_i^{\max} = \frac{C_i}{U_i^{\min}}$. Algorithm overheads, though presented in cycles, were measured using a constant 700 MHz CPU clock speed; we convert them to times accordingly.

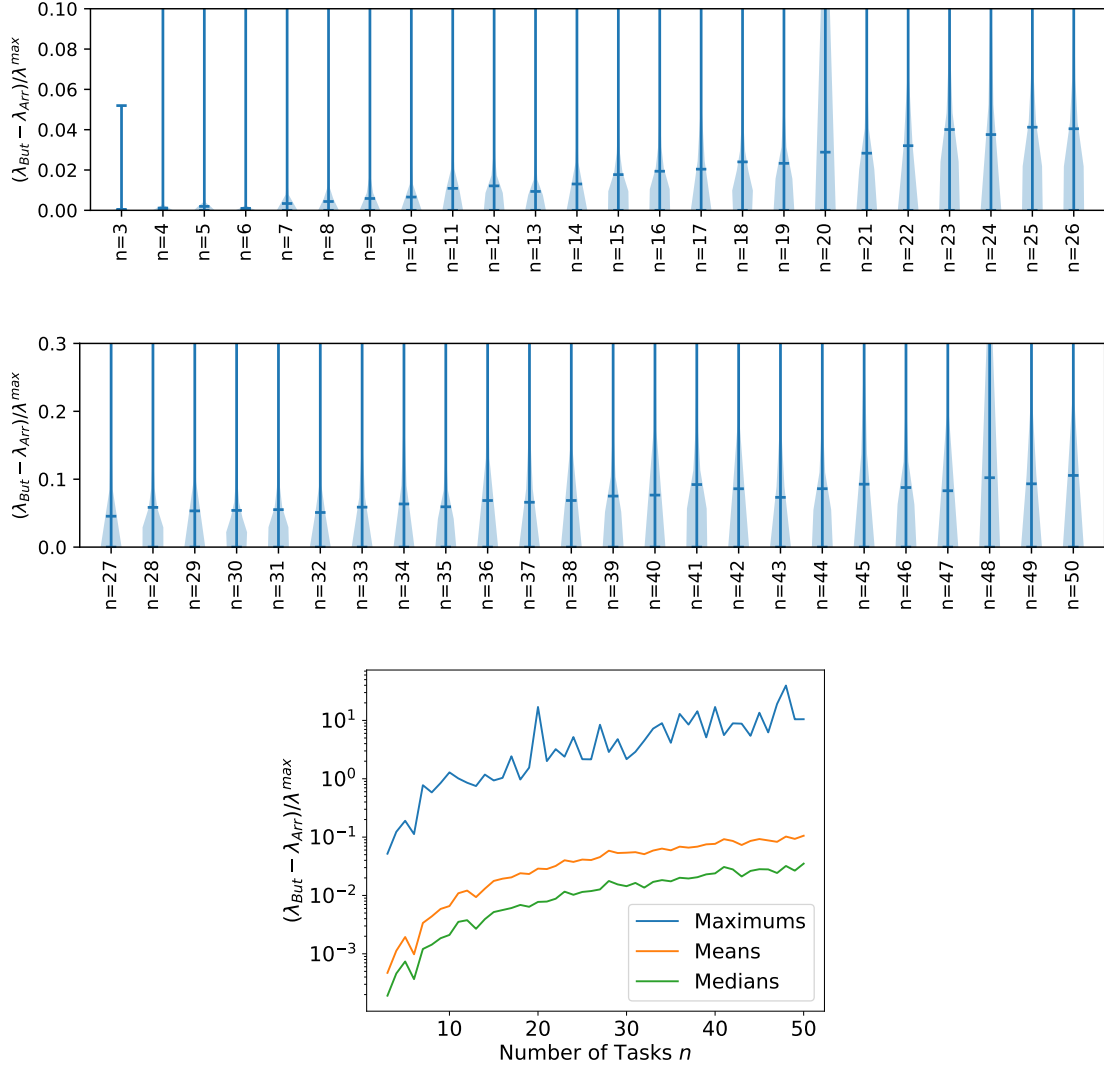


Figure 4 Difference between λ_{But} and λ_{Arr} , normalized by λ^{\max} .

Figures 4 (Top) and (Middle) show the distribution of differences $\frac{(\lambda_{But} - \lambda_{Arr})}{\lambda^{\max}}$ between the value λ_{But} necessary to account for the overhead of the algorithm of Buttazzo et al. and λ_{Arr} of our array-based algorithm, normalized by the value λ^{\max} taken from the original task set without overheads. Figure 4 (Bottom) shows the mean, median, and maximum of the normalized values

⁸ We ignore the “angle-synchronous” entry and renormalize probabilities over the remaining distribution.

for each number n of tasks. Task sets not requiring compression ($\lambda = 0$) are excluded, as are those deemed infeasible under any amount of compression.

As expected, greater online overhead incurs additional compression when it is accounted for. We observe that for $n = 50$ tasks, the algorithm of Buttazzo et al. must, on average, compress tasks by nearly 11% more of the allowable range than ours. In the worst case, it compresses tasks by nearly $40\times$ more than the original λ^{\max} value. These results highlight that the speedups achieved by our algorithm have practical relevance as they **reduce the amount of compression required during online admission of new tasks** for workloads with parameters drawn from real-world applications.

3.3 Extension to Fluid Scheduling

A set of tasks is fluid schedulable on m identical processor cores if and only if (i) its total utilization does not exceed m , and (ii) no individual task's utilization exceeds 1 [1]. Orr and Baruah therefore argued that, so long as each elastic task's maximum utilization $U_i^{\max} \leq 1$, the algorithm of Buttazzo et al. algorithm can be extended to fluid scheduling simply by setting the desired utilization $U_D = m$. The results and conclusions drawn in this section are therefore applicable to fluid scheduling as well: our procedure [26] in Algorithm 2 may be used in place of the original procedure [11] in Algorithm 1 to achieve faster compression (once initialized) and admission of new tasks.

Sets of sequential tasks requiring multiple processors to execute — i.e., those which benefit from fluid scheduling — have total maximum utilizations $U_{\text{SUM}}^{\max} > 1$, and may also have total minimum utilizations $U_{\text{SUM}}^{\min} > 1$. Rather than evaluating new set of tasks, we consider whether our existing results extend to fluid scheduling by asking the question, “Do the values selected for U_{SUM}^{\max} or U_{SUM}^{\min} affect execution time?”

For the 10 000 sets of 50 tasks generated in Section 3.2.2, we produce 3 scatter plots for the initialization (“Init”) and compression (“Compress”) times of each implementation: these plot cycles against the values U_{SUM}^{\min} , U_{SUM}^{\max} , and the absolute distance between them, $(U_{\text{SUM}}^{\max} - U_{\text{SUM}}^{\min})$, as shown in Figure 5. We do not observe a significant dependence of execution time on these values. Therefore, the performance results illustrated in Figures 2 and 3 should also extend to fluid scheduling.

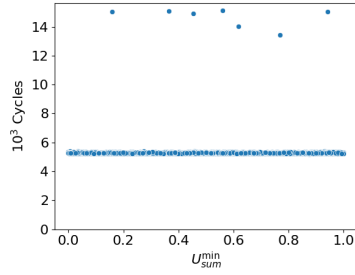
4 Partitioned EDF

In this section, we propose two alternative approaches to elastic scheduling of partitioned EDF tasks. First, we consider a binary, rather than linear, search over the space of compression allowed due to the minimum utilization constraint on each task. Second, using the insight that under partitioned EDF scheduling a set of tasks is guaranteed to be schedulable if its utilization does not exceed a function of the number of cores, we apply our procedure [26] in Algorithm 2 for compressing to this utilization bound.

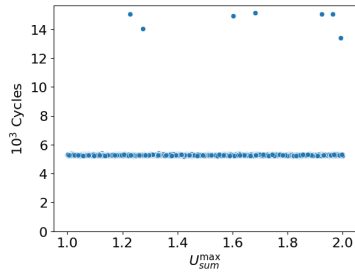
4.1 Binary Search

We observe that a straightforward optimization may be applied to the approach of Orr and Baruah [22] summarized in Section 2.2. Rather than iterating over all values of $\lambda \in [0, \lambda^{\max}]$ with granularity ϵ in *sequential order*, we can instead perform a *binary search* in time $\Theta(\log \frac{\lambda^{\max}}{\epsilon})$, as outlined in Algorithm 3. Thus, total time complexity is reduced to

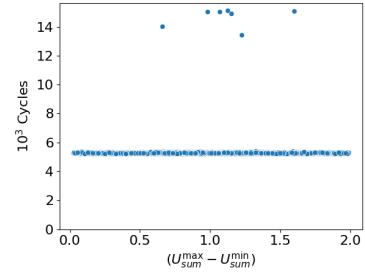
$$\Theta\left((n \log n + n \cdot m) \cdot \log\left(\frac{\lambda^{\max}}{\epsilon}\right)\right) \quad (12)$$



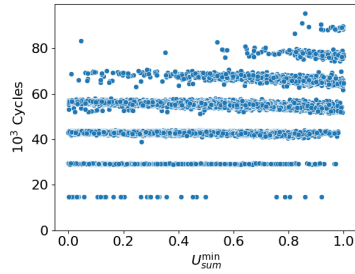
(a) Buttazzo Init.



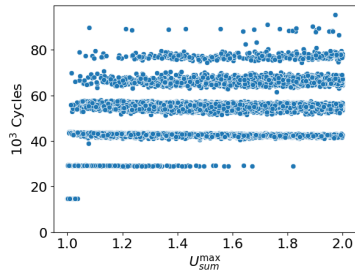
(b) Buttazzo Init.



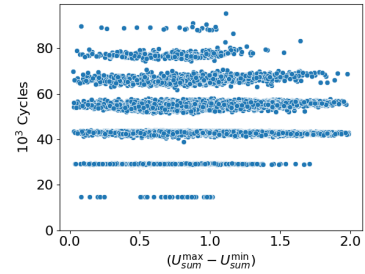
(c) Buttazzo Init.



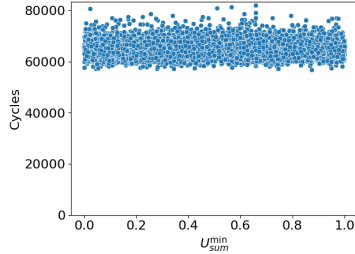
(d) Buttazzo Compress.



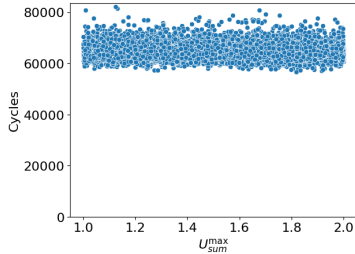
(e) Buttazzo Compress.



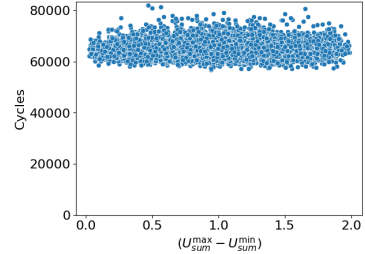
(f) Buttazzo Compress.



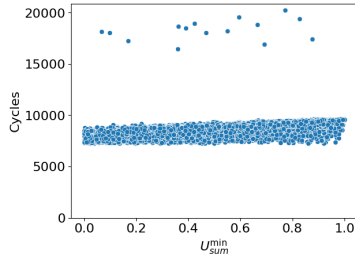
(g) Improved Init: Array.



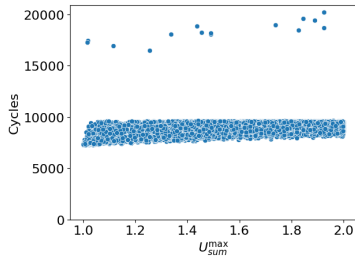
(h) Improved Init: Array.



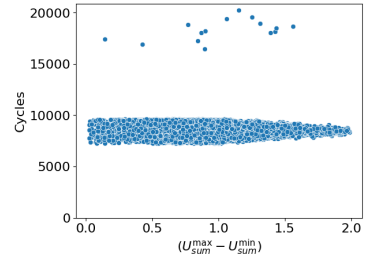
(i) Improved Init: Array.



(j) Improved Compress: Array.

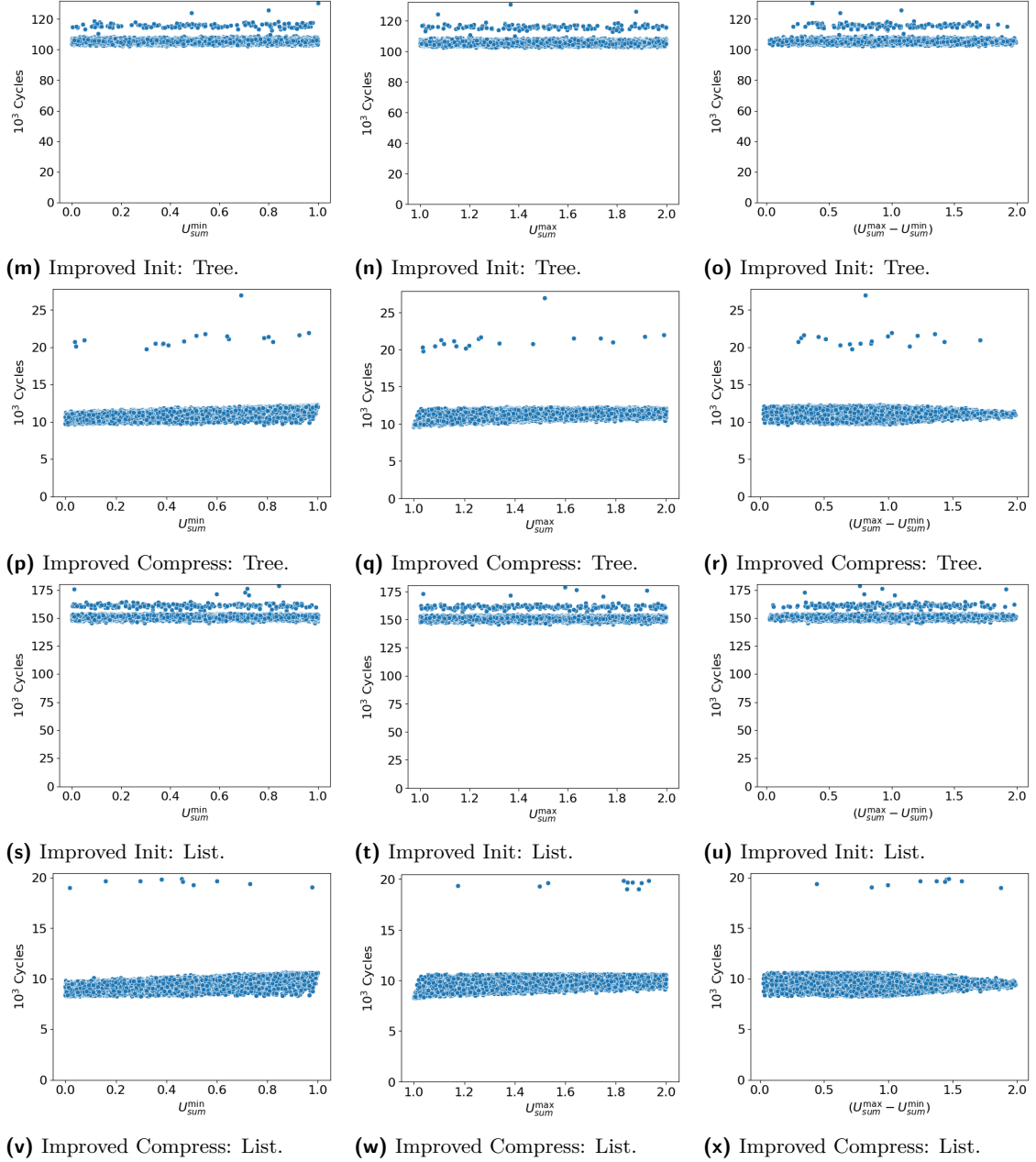


(k) Improved Compress: Array.



(l) Improved Compress: Array.

2:18 Improved Elastic Scheduling Algorithms for Implicit-Deadline Tasks



■ **Figure 5** Execution times by utilization metrics for 50 tasks.

Algorithm 3 Elastic_Partitioned_EDF(Γ, m)

Input: A list Γ of elastic tasks to schedule on m processor cores

Output: The value λ to obtain feasibility

```

1  $\lambda^{\max} \leftarrow 0$ 
2 forall  $\tau_i \in \Gamma$  do
3    $\lambda_i^{\max} \leftarrow \frac{U_i^{\max} - U_i^{\min}}{E_i}$ 
4    $\lambda^{\max} \leftarrow \max(\lambda^{\max}, \lambda_i^{\max})$ 
5 end
6 if  $\Gamma(0)$  is schedulable on  $m$  cores then return 0
7 if  $\Gamma(\lambda^{\max})$  is not schedulable on  $m$  cores then return INFEASIBLE
8  $\lambda_{\text{HI}} \leftarrow \lambda^{\max}$ 
9  $\lambda_{\text{LO}} \leftarrow 0$ 
10 do
11    $\lambda \leftarrow (\lambda_{\text{HI}} + \lambda_{\text{LO}}) / 2$ 
12   if  $\Gamma(\lambda)$  is schedulable on  $m$  cores then  $\lambda_{\text{HI}} \leftarrow \lambda$ 
13   else  $\lambda_{\text{LO}} \leftarrow \lambda$ 
14 while  $\lambda_{\text{HI}} - \lambda_{\text{LO}} > \epsilon$ 
15 return  $\lambda_{\text{HI}}$ 

```

Algorithm 3 uses the notation $\Gamma(\lambda)$ from Baruah [3], denoting the task system obtained from Γ by applying compression λ , i.e., with each task τ_i having a utilization $U_i(\lambda)$ according to Equation 8. The algorithm first checks if $\Gamma(0)$ — the uncompressed task set — is schedulable by partitioned EDF on m cores; schedulability may be determined according to the heuristics employed by Orr and Baruah [22]. If so, it returns the value $\lambda = 0$. It then checks if $\Gamma(\lambda^{\max})$ is schedulable; if not, the algorithm fails. Otherwise, it performs binary search over values of λ in the range $[0, \lambda^{\max}]$: λ_{HI} (initialized to λ^{\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. At each step, the algorithm checks schedulability of $\Gamma(\lambda)$; if feasibility is determined, λ_{HI} is decreased to the tested value of λ ; otherwise, λ_{LO} is increased to the tested value of λ . The algorithm terminates when the difference between λ_{HI} and λ_{LO} does not exceed ϵ .

4.1.1 Optimality of Search for λ

We now discuss and prove results about the optimality of iterative and binary searches for partitioned EDF scheduling. We begin by introducing the term $\lambda_{\Gamma, m}^*$, defined as the smallest value of λ for which $\Gamma(\lambda)$ is schedulable by partitioned EDF on m cores.

The first result is intuitive: it says that, once you compress a task system such that it is schedulable, it will remain schedulable when compressed more.

► **Theorem 2.** *Given a value of λ , if $\Gamma(\lambda)$ is partitioned EDF schedulable on m cores, then $\Gamma(\lambda')$ is also partitioned EDF schedulable for every value of $\lambda' \geq \lambda$.*

Proof. Consider a set Γ of n tasks τ_i . If $\Gamma(\lambda)$ is partitioned EDF schedulable on m cores, then there exists a partition $\{\Gamma_1, \dots, \Gamma_m\}$ of Γ such that the following condition holds:

$$\forall j \in 1..m, \quad \sum_{\tau_i \in \Gamma_j} U_i(\lambda) \leq 1$$

Consider a value $\lambda' \geq \lambda$. For each task τ_i ,

$$U_i(\lambda') = \max(U_i^{\max} - \lambda' E_i, U_i^{\min}) \leq \max(U_i^{\max} - \lambda E_i, U_i^{\min}) = U_i(\lambda)$$

Since $U_i(\lambda') < U_i(\lambda)$, it follows that:

$$\forall j \in 1..m, \quad \sum_{\tau_i \in \Gamma_j} U_i(\lambda') \leq \sum_{\tau_i \in \Gamma_j} U_i(\lambda) \leq 1$$

So there remains a partition of $\Gamma(\lambda')$ for which the condition holds. \blacktriangleleft

It follows that $\Gamma(\lambda)$ is partitioned EDF schedulable for every value of λ that exceeds $\lambda_{\Gamma, m}^*$. This allows us to say something about the optimality of the elastic algorithms for partitioned EDF scheduling.

► **Theorem 3.** *The values of λ obtained by using the iterative approach of Orr and Baruah [22] or the binary search in Algorithm 3 will be within ϵ of λ^* if an exact test of partitioned EDF schedulability is performed for $\Gamma(\lambda)$ at each considered value of λ . In other words, $\lambda - \lambda^* < \epsilon$.*

Proof.

- **Iterative Approach:** The algorithm tests $\lambda = 0$ first; if $\lambda^* = 0$, then the algorithm returns this value. Otherwise, consider the value λ returned by the algorithm: $\Gamma(\lambda)$ is feasible, but $\Gamma(\lambda - \epsilon)$ is *not* feasible. It follows from Theorem 2 that $\lambda^* > \lambda - \epsilon$, which implies $\lambda - \lambda^* < \epsilon$.
- **Binary Search Approach:** The algorithm again tests $\lambda = 0$ first; if $\lambda^* = 0$, then the algorithm returns this value. Otherwise, consider the value λ_{HI} returned by the algorithm: $\Gamma(\lambda_{\text{HI}})$ is feasible, but $\Gamma(\lambda_{\text{LO}})$ is not; thus, by Theorem 2, $\lambda^* > \lambda_{\text{LO}}$. Due to the algorithm's termination condition, we know that $\lambda_{\text{HI}} - \lambda_{\text{LO}} \leq \epsilon$, and so $\lambda - \lambda^* < \epsilon$.

► **Corollary 4.** *The values λ_{LIN} obtained by the iterative approach of Orr and Baruah [22] and λ_{BIN} obtained by Algorithm 3 will be within ϵ of each other if an exact test of partitioned EDF schedulability is performed for $\Gamma(\lambda)$ at each considered value of λ . In other words, $|\lambda_{\text{LIN}} - \lambda_{\text{BIN}}| < \epsilon$.*

Proof. From Theorem 3, $\lambda_{\text{LIN}} - \lambda^* < \epsilon$ and $\lambda_{\text{BIN}} - \lambda^* < \epsilon$, so $|\lambda_{\text{LIN}} - \lambda_{\text{BIN}}| < \epsilon$. \blacktriangleleft

This tells us that, given an exact schedulability test for partitioned EDF, both algorithms will find values for λ that are within ϵ of the optimal value λ^* and are within ϵ of each other. However, no such guarantee can be made if schedulability is determined by heuristic, as illustrated in Figure 7. It follows that:

► **Theorem 5.** *The difference between the values λ_{LIN} obtained by the iterative approach of Orr and Baruah [22] and λ_{BIN} obtained by Algorithm 3 might exceed ϵ if heuristic tests of EDF schedulability are used.*

This has a surprising implication, which follows from the above results.

► **Corollary 6.** *Given a value of λ , if $\Gamma(\lambda)$ is identified by heuristic to be partitioned EDF schedulable on m cores, then $\Gamma(\lambda')$ might not be identifiable as such for some value of $\lambda' > \lambda$.*

The implication, then, is that while binary search is faster, it might overcompress a set of tasks by more than ϵ when applying heuristic partitioning (of course, the iterative search might overcompress instead). However, as we show in Section 4.3, binary search compresses, on average, only $0.262 \times \epsilon$ more than iterative search for the sets of tasks we evaluated.

4.2 Application of Algorithm 2

In [2], it is observed that under the first-fit and best-fit heuristics, a set Γ of tasks τ_i is schedulable on m processor cores if the total utilization $\sum_i U_i$ does not exceed $\frac{m+1}{2}$ and if no single task's utilization exceeds 1. Thus, the efficient procedure outlined in Algorithm 2 can be adopted by compressing to a desired utilization $U_D = \frac{m+1}{2}$, achieving compression in $\mathcal{O}(n \log n)$ time.

We note that $\frac{m+1}{2}$ is an *upper-bound* on the utilization required by the first-fit and best-fit heuristics. Thus, the amount of compression resulting from an application of this approach might be more than necessary to achieve partitioned EDF schedulability, even under the above-listed heuristics. It follows that the approach of Orr and Baruah [22], while slower, might achieve better results — both in terms of compressing utilizations less aggressively, and by identifying more schedulable task sets. We evaluate these tradeoffs in Section 4.3.

4.3 Evaluation

We now compare the approaches to elastic partitioned EDF scheduling proposed in this section to the original approach of Orr and Baruah [22].

4.3.1 Implementation

In this section, we compare the following three approaches:

1. ITER: The iterative approach from [22]. For each value of λ tested, we attempt to find a schedulable partition by employing the best-fit decreasing then first-fit decreasing bin packing heuristics. The algorithm terminates if either is successful.
2. BIN: Our proposed binary search approach in Algorithm 3, again using the best-fit then first-fit decreasing bin packing heuristics.
3. UTIL: The utilization-based approach of Algorithm 2, with $U_D = \frac{m+1}{2}$. We use the array-based implementation, as this was observed to perform best in our evaluations in Section 3.2.

We implement each approach in C++, compiling and measuring execution times with the same settings as described in Section 3.2.1, and running them on the same Raspberry Pi 3 Model B+. All tasks are also represented using the same data structure.

4.3.2 Generating Task Sets

We generate sets Γ of tasks τ_i according to Orr and Baruah's methodology in [22]:

- We consider multiprocessor platforms with $m = 4, 8$, and 16 identical processor cores.
- For each number of cores m , we consider sets of n tasks, with $n = 2m, 4m$, and $8m$.
- The maximum utilization U_i^{\max} assigned to each task τ_i is selected at random, but we constrain these values to be no more than a parameter α . We separately consider values of $\alpha \in \{0.6, 0.8, 1.0\}$.
- Each set of tasks has a total maximum utilization U_{SUM}^{\max} of $u \cdot m \cdot \alpha$. We separately consider values of $u \in \{1.1, 1.5, 1.9\}$.
- For each combination of values m, n, α , and u , we generate 1000 sets of tasks.
- We use the DRS algorithm [16] to distribute the total maximum utilization U_{SUM}^{\max} across individual U_i^{\max} values. DRS allows us to select these values uniformly from the space of selections satisfying the conditions that (i) the total $\sum_i U_i^{\max}$ equals the specified U_{SUM}^{\max} and (ii) each value U_i^{\max} does not exceed the constraint imposed by the chosen value of α .

- 553 ■ Individual minimum utilizations U_i^{\min} are assigned at random, selected uniformly from the
- 554 range $(0, U_i^{\max}]$.
- 555 ■ Elastic coefficients E_i are assigned at random, selected uniformly from the range $(1, 5]$.

556 4.3.3 Linear versus Binary Search

557 We begin by comparing the ITER and BIN approaches. For each set of tasks, we compute λ^{\max}
 558 per Equation 9, then search for the optimal λ with granularity $\epsilon = \frac{\lambda^{\max}}{1000}$ (the same value tested by
 559 Orr and Baruah in [22]).

560 Figure 6 shows — for each combination of m , n , α , and u — the median and maximum speedup
 561 achieved by binary search over linear search. As before, task sets not requiring compression
 562 ($\lambda = 0$) are excluded, as are those deemed infeasible under any amount of compression. Binary
 563 search achieves significant speedups, especially for larger values of α and u . These task sets
 564 have larger total maximum utilizations, and therefore tend to need more compression to achieve
 565 schedulability. In such cases, the linear search takes longer to reach the higher λ value, so binary
 566 search is significantly faster. Median speedups for each combination were as high as $51\times$, while
 567 the maximum speedup observed was $86\times$.

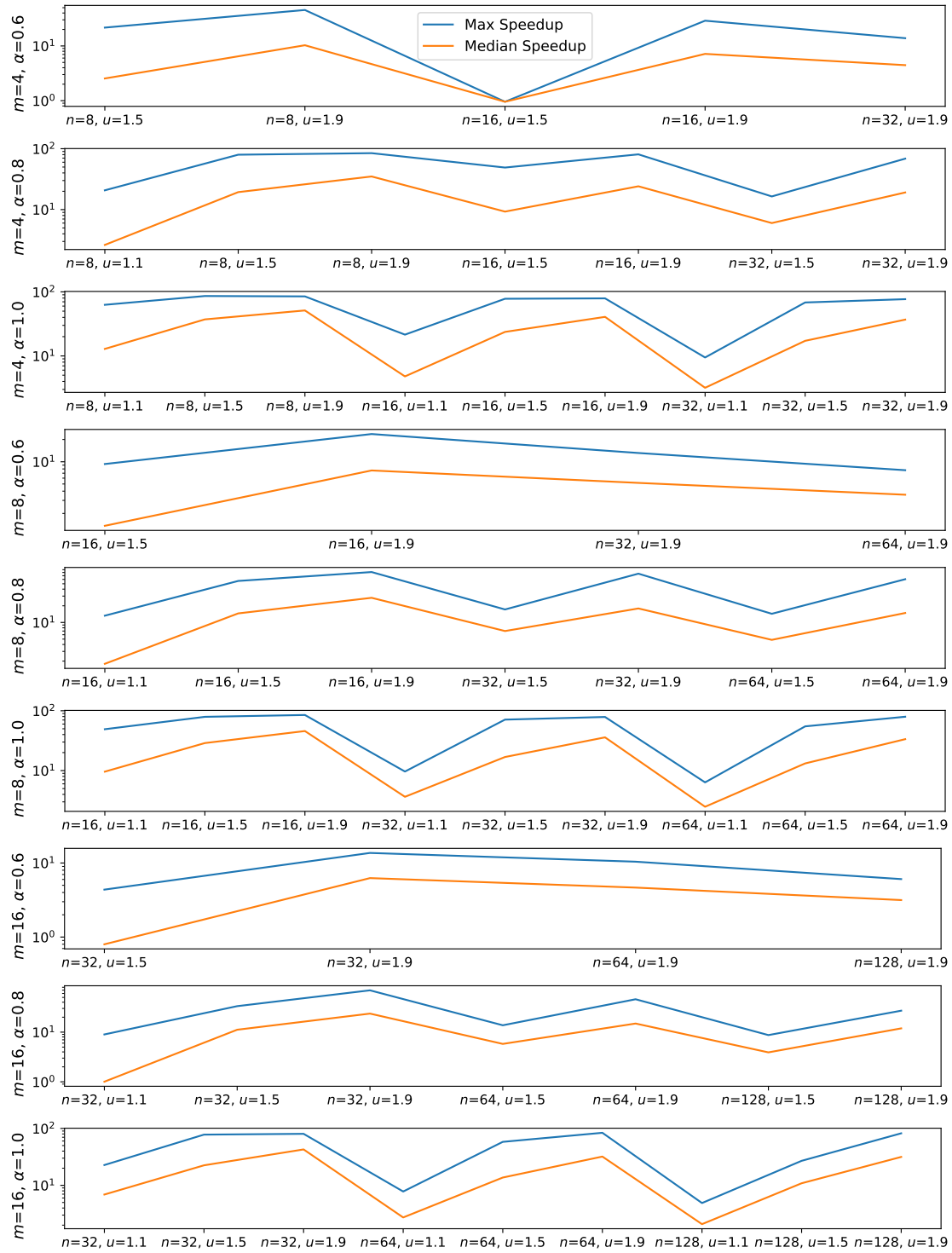
568 Figure 7 shows the distribution of differences $\frac{(\lambda_{\text{BIN}} - \lambda_{\text{LIN}})}{\epsilon}$ between the value λ_{BIN} achieved by
 569 binary search and λ_{LIN} returned by linear search, normalized by ϵ . We again exclude trivial or
 570 infeasible task sets. Where outliers extend beyond the plotted boundaries, the y-axis labels denote
 571 the maximum value. We observe that, although the values λ_{BIN} and λ_{LIN} typically do not differ
 572 by more than ϵ , there are cases where they differ by much more. For 16 tasks on 4 cores, with
 573 $\alpha = 1.0$ and $u = 1.9$, λ_{BIN} exceeds λ_{LIN} by more than $200\times\epsilon$. Generally, we see that outliers occur
 574 where λ_{BIN} is larger than λ_{LIN} . This makes sense due to the behavior of binary search: search
 575 proceeds downward in factors of 2 from larger values, and if $\Gamma(\lambda)$ is deemed unschedulable for
 576 some tested value of λ greater than the optimal λ^* , the binary search will continue to test larger
 577 values. Despite these outliers, **the average compression values agree closely**, differing by less
 578 than $0.262\times\epsilon$ for every considered combination. Therefore, given the significant speedups gained,
 579 binary search is an attractive approach.

580 4.3.4 Fast Compression

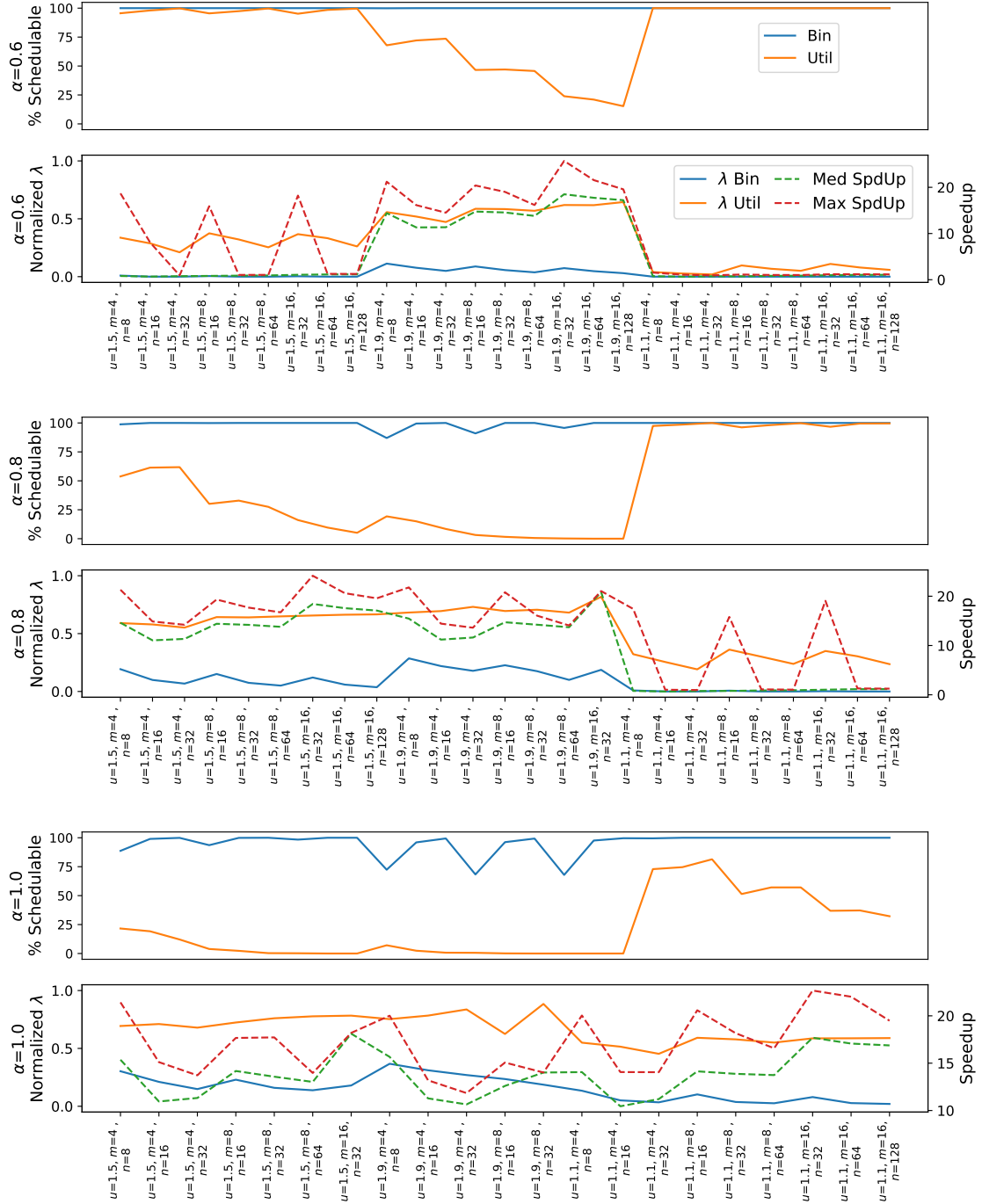
581 Although the binary search approach already improves execution time significantly while having
 582 minimal impact on the quality of our solution, we expect that our UTIL approach, which applies
 583 our quasilinear-time algorithm, will be even faster, though we also expect it to be more pessimistic.
 584 We evaluate this hypothesis by comparing the number of task sets each approach deems feasible,
 585 the resulting values of λ necessary to achieve schedulability, and the time to find those values
 586 of λ . As in [22], to ensure a consistent comparison, we only compare λ values (and, in our case,
 587 execution times — measuring execution times was outside the scope of the work in [22]) for those
 588 tasks deemed feasible. Also as in [22], we separately consider each value of m , α , n , and u .

589 Results are shown in Figure 8, which alternates between two types of plots. The first type
 590 shows the percentage of task sets identified as feasible by BIN and UTIL. The second type compares
 591 the median and maximum speedup gained by UTIL over BIN to the mean λ values achieved by
 592 each approach. As in [22], λ values are normalized by λ^{\max} to give a value in the interval $[0,1]$;
 593 this is necessary for comparing λ values across task sets.

594 We observe that UTIL identifies fewer schedulable task sets, and for those it does identify
 595 as schedulable, it typically requires more compression. This is especially true for larger values
 596 of α and u for which the total maximum utilization U_{SUM}^{\max} is larger. Nonetheless, at the cost
 597 of more pessimism, UTIL achieves speedups observed to reach over $20\times$; this may be desirable



■ **Figure 6** Speedups achieved by BIN over ITER.



■ **Figure 8** Speed and Schedulability Tradeoffs Between BIN and UTIL.

where online decisions must be made rapidly. For example, in mixed-criticality systems [30, 10], elastic frameworks have been proposed to extend the periods of low-criticality tasks, rather than suspending them, in response to critical task overruns [24, 29]. The transition must be made as quickly as possible, and so overcompression is acceptable (and is no worse than the alternative of dropping all low-criticality jobs).

5 Global EDF and RM

In this section, we present and evaluate an exact polynomial-time algorithm for elastic utilization compression under global earliest deadline first (EDF) and global rate monotonic (RM) scheduling.

5.1 The Global EDF Algorithm

Recall the condition for global EDF schedulability on m cores [15, Theorem 5] from Equation 10 in Section 2.2.2, which we restate here:

$$\sum_{\tau_i \in \Gamma} U_i \leq m - (m - 1) \cdot \max_{\tau_i \in \Gamma} \{U_i\}$$

Without loss of generality, let's say that τ_j is the task with the maximum utilization, i.e., $U_j = \max_{\tau_i \in \Gamma} \{U_i\}$. Then we can restate the schedulability condition as

$$\sum_{\tau_i, i \neq j} U_i + mU_j \leq m \quad (13)$$

► **Theorem 7.** *For a set Γ of elastic tasks, the amount of compression λ needed to satisfy the schedulability condition in Equation 13 can be found by finding λ for the condition $\sum_{\tau_i \in \Gamma^*} U_i \leq m$ for a set of tasks Γ^* where the task $\tau_j = (U_j^{\min}, U_j^{\max}, E_j)$ in Γ has been replaced by a task $\tau_j^* = (mU_j^{\min}, mU_j^{\max}, mE_j)$ in Γ^* .*

Proof. Consider the value λ satisfying $\sum_{\tau_i \in \Gamma^*} U_i = m$, i.e.,

$$\sum_{\tau_i \in \Gamma^*} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} = m$$

Assume that $\tau_j^* \in \Gamma^*$ is parameterized as $\tau_j^* = (mU_j^{\min}, mU_j^{\max}, mE_j)$. Then:

$$\sum_{\tau_i \in \Gamma^*, i \neq j} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} + \max \{mU_j^{\max} - \lambda mE_j, mU_j^{\min}\} = m$$

Equivalently,

$$\sum_{\tau_i \in \Gamma^*, i \neq j} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} + m \times \max \{U_j^{\max} - \lambda E_j, U_j^{\min}\} = m$$

So $\sum_{\tau_i \in \Gamma, i \neq j} U_i(\lambda) + mU_j(\lambda) = m$ and Γ is global EDF schedulable. ◀

Intuitively, this says we can replace τ_j with a task τ_j^* with utilization and elasticity values scaled by m ; schedulability is then based on a utilization bound of m and the system can be compressed using our algorithm [26, Algorithm 1]. However, as the task with the maximum utilization can change during compression, the utilization bound (the RHS of Equation 10) might no longer hold. Therefore, we must assume *every* task may take the role of τ_j after compression, so we repeat this procedure for each task. We then take the result for which (i) the task with the maximum utilization after compression matches the one taking the role of τ_j ; and (ii) if there are multiple such consistent results, we take the one that applies the least compression. This procedure is outlined in Algorithm 4.

■ **Algorithm 4** Elastic_Global_EDF(Γ, m)

Input: A list Γ of elastic tasks to schedule on m processor cores
Output: The value λ to obtain feasibility

```

1 if  $\Gamma(0)$  is schedulable on  $m$  cores then return 0
2 if  $\Gamma(\lambda^{\max})$  is not schedulable on  $m$  cores then return INFEASIBLE
3 Sort  $\Gamma$  in non-decreasing order of  $\phi_i$  (see Equation 7)
4  $\lambda \leftarrow \lambda^{\max}$ 
5 forall  $\tau_i \in \Gamma$  do
6    $\tau_j^* \leftarrow (U_j^{\max} : mU_i^{\max}, U_j^{\min} : mU_i^{\min}, E_j : mE_i)$ 
7    $\Gamma^* \leftarrow \Gamma$ , Remove  $\tau_i$  and insert  $\tau_j^*$  into  $\Gamma^*$ 
8   ▷ Invoke Algorithm 2
9    $\lambda^* \leftarrow \text{ELASTIC\_COMPRESSION}(\Gamma^*, m)$ 
10  if  $\frac{U_j^*}{m}$  is the maximum compressed utilization and  $\lambda^* < \lambda$  then  $\lambda \leftarrow \lambda^*$ 
11 end
12 return  $\lambda$ 

```

5.1.1 Description

The algorithm takes a set Γ of elastic tasks and checks whether compression is necessary, and whether feasibility can be achieved. It then sorts the tasks in non-decreasing order of their ϕ_i values (see Equation 7). For each task τ_i in Γ , a representative task τ_j^* is constructed with $U_j^{\min} = mU_i^{\min}$, $U_j^{\max} = mU_i^{\max}$, and $E_j = mE_i$ per Theorem 7. Then, τ_i is replaced in Γ with τ_j^* to form the temporary set Γ^* ; from Equation 7 we can see that

$$\phi_j = \left(\frac{mU_i^{\max} - mU_i^{\min}}{mE_i} \right) = \left(\frac{U_i^{\max} - U_i^{\min}}{E_i} \right) = \phi_i$$

so tasks $\tau_i^* \in \Gamma^*$ remain sorted by their ϕ_i values.

Our linear-time procedure listed in Algorithm 2 is then invoked, and the compression value λ^* is retrieved. We note that although our compression algorithm, as written in [26, Algorithm 1], does not return a value of λ , it can be easily modified to do so. From Equations 3 and 8 we have

$$\lambda = \left(\frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}} \right)$$

Then from Line 16 of its listing in Algorithm 2, we see that this value is computed and tracked by our algorithm, and so it can be retrieved in constant time for use in Line 9 of Algorithm 4. If, by checking $U_i = U_j^*/m$, it is determined that τ_i would have the maximum compressed utilization, then the task set is schedulable under global EDF. Since multiple feasible configurations might be found, a variable λ is used to track the best value (i.e., the minimum compression needed); this is initialized to λ^{\max} as the algorithm has already deemed feasibility.

5.1.2 Execution Time Complexity

For a set Γ of n tasks, sorting in order of ϕ_i values takes time $\mathcal{O}(n \log n)$. Inside the **forall** loop in Algorithm 4, constructing τ_j^* from τ_i takes constant time. Our compression algorithm runs in quasilinear time, but this time is dominated by sorting the tasks [26]. Since Γ has already been sorted, compression takes time linear in the number of tasks. Checking whether U_j^*/m is the maximum compressed utilization also takes linear time. Since each iteration of the loop takes time $\mathcal{O}(n)$ and it runs once for each of the n tasks, the total execution time complexity is $\mathcal{O}(n^2)$.

5.2 Extension to Global RM

We now apply this same approach to tasks scheduled in global rate monotonic (RM) fashion. Recall the condition for global RM schedulability on m cores [7, Theorem 5] from Equation 11 in Section 2.2.3, which we restate here:

$$\sum_{\tau_i \in \Gamma} U_i \leq \frac{m}{2} \left(1 - \max_{\tau_i \in \Gamma} \{U_i\} \right) + \max_{\tau_i \in \Gamma} \{U_i\}$$

As with global EDF, without loss of generality we can say that τ_j is the task with the maximum utilization, i.e., $U_j = \max_{\tau_i \in \Gamma} \{U_i\}$. Then we can restate the schedulability condition as:

$$\sum_{\tau_i, i \neq j} U_i + \frac{m}{2} U_j \leq \frac{m}{2} \quad (14)$$

► **Theorem 8.** *For a set Γ of elastic tasks, the amount of compression λ needed to satisfy the schedulability condition in Equation 14 can be found by finding λ for the condition $\sum_{\tau_i \in \Gamma^*} U_i \leq \frac{m}{2}$ for a set of tasks Γ^* where the task $\tau_j = (U_j^{\min}, U_j^{\max}, E_j)$ in Γ has been replaced by a task $\tau_j^* = (\frac{m}{2} U_j^{\min}, \frac{m}{2} U_j^{\max}, \frac{m}{2} E_j)$ in Γ^* .*

Proof. Our reasoning is the same as that of Theorem 7, but with terms m replaced by terms $\frac{m}{2}$. Consider the value λ satisfying $\sum_{\tau_i \in \Gamma^*} U_i = \frac{m}{2}$, i.e.,

$$\sum_{\tau_i \in \Gamma^*} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} = \frac{m}{2}$$

Assume that $\tau_j^* \in \Gamma^*$ is parameterized as $\tau_j^* = (\frac{m}{2} U_j^{\min}, \frac{m}{2} U_j^{\max}, \frac{m}{2} E_j)$. Then:

$$\sum_{\tau_i \in \Gamma^*, i \neq j} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} + \max \left\{ \frac{m}{2} U_j^{\max} - \lambda \frac{m}{2} E_j, \frac{m}{2} U_j^{\min} \right\} = \frac{m}{2}$$

Equivalently,

$$\sum_{\tau_i \in \Gamma^*, i \neq j} \max \{U_i^{\max} - \lambda E_i, U_i^{\min}\} + \frac{m}{2} \times \max \{U_j^{\max} - \lambda E_j, U_j^{\min}\} = \frac{m}{2}$$

So $\sum_{\tau_i \in \Gamma, i \neq j} U_i(\lambda) + \frac{m}{2} U_j(\lambda) = \frac{m}{2}$ and Γ is global RM schedulable. ◀

Algorithm 4 can therefore be adapted to global RM scheduling by simply changing the transformation of τ_i into τ_j^* and compressing to a utilization bound of $\frac{m}{2}$. For completeness, we outline this procedure as Algorithm 5.

5.3 Evaluation

To quantify the improvements offered by our proposed algorithms for global EDF and RM, we compare their execution time to the original algorithms of Orr and Baruah in [22, 21].

We use the same approach to compile and measure execution times as described in Section 4.3, and run on a Raspberry Pi 3 Model B+ as before. We also use the same task sets generated in Section 4.3 to match Orr and Baruah's methodology in [22]. We use the array-based implementation of Algorithm 2, as this performed best in the experiments of Section 3.2.

Algorithm 5 Elastic_Global_RM(Γ, m)

Input: A list Γ of elastic tasks to schedule on m processor cores
Output: The value λ to obtain feasibility

```

1 if  $\Gamma(0)$  is schedulable on  $m$  cores then return 0
2 if  $\Gamma(\lambda^{\max})$  is not schedulable on  $m$  cores then return INFEASIBLE
3 Sort  $\Gamma$  in non-decreasing order of  $\phi_i$  (see Equation 7)
4  $\lambda \leftarrow \lambda^{\max}$ 
5 forall  $\tau_i \in \Gamma$  do
6    $\tau_j^* \leftarrow (U_j^{\max} : \frac{m}{2}U_i^{\max}, U_j^{\min} : \frac{m}{2}U_i^{\min}, E_j : \frac{m}{2}E_i)$ 
7    $\Gamma^* \leftarrow \Gamma$ , Remove  $\tau_i$  and insert  $\tau_j^*$  into  $\Gamma^*$ 
8   ▷ Invoke Algorithm 2
9    $\lambda^* \leftarrow \text{ELASTIC\_COMPRESSION}(\Gamma^*, \frac{m}{2})$ 
10  if  $\frac{2U_j^*}{m}$  is the maximum compressed utilization and  $\lambda^* < \lambda$  then  $\lambda \leftarrow \lambda^*$ 
11 end
12 return  $\lambda$ 

```

5.3.1 Global EDF

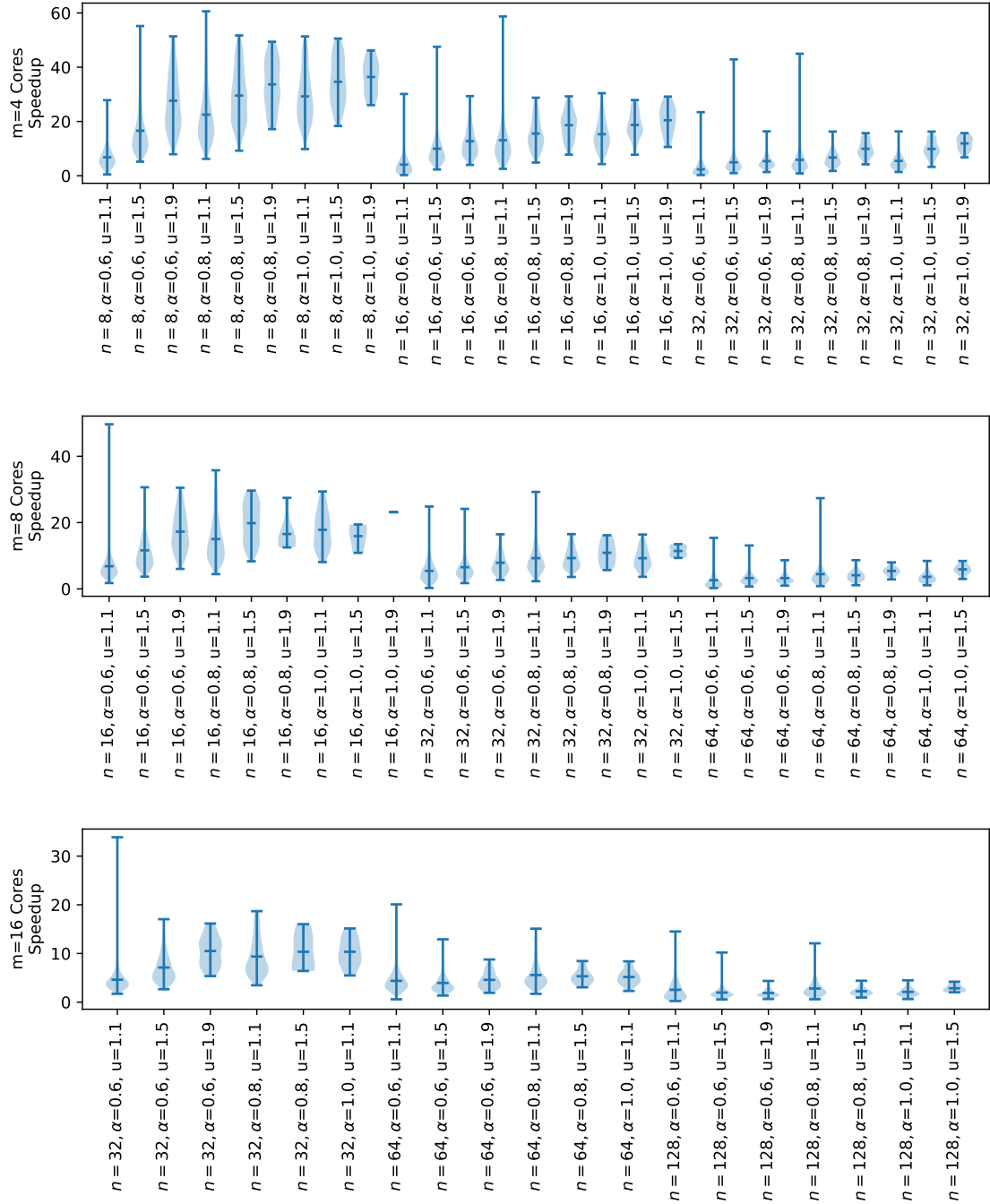
We begin by comparing the execution time of Algorithm 4 to Orr and Baruah’s elastic compression algorithm for global EDF [22, Algorithm 2], which we summarize in Section 2.2.2. As before we search for λ with granularity $\epsilon = \frac{\lambda^{\max}}{1000}$.

Figure 9 shows — for each combination of m , n , α , and u — the distribution of speedups achieved by our exact algorithm over iterative search, with horizontal markers indicating median and maximum values. Figure 10 shows the distributions of each algorithm’s execution times. Task sets not requiring compression ($\lambda = 0$) are excluded, as are those deemed infeasible under any amount of compression. We see that our proposed algorithm achieves significant speedups, especially for larger values of α and u . These task sets have larger total maximum utilizations, and therefore tend to need more compression to achieve schedulability. In such cases, the linear search takes longer to reach the higher λ value, so our exact algorithm is consistently faster. Median speedups for each combination were as high as $37\times$, while the maximum speedup observed was $60\times$.

5.3.2 Global RM

We next compare the execution time of Algorithm 5 to Orr and Baruah’s elastic compression algorithm for global EDF from [21], which we summarize in Section 2.2.3. As before we search for λ with granularity $\epsilon = \frac{\lambda^{\max}}{1000}$.

Figure 11 shows the distributions of speedups achieved by our exact algorithm over iterative search and Figure 12 shows the distributions of their execution times. As in Figures 9 and 10, task sets not requiring compression ($\lambda = 0$) are excluded, as are those deemed infeasible under any amount of compression. We again see significant speedups, with similar patterns to the above results for global EDF scheduling. Median speedups were as high as $43\times$, while the maximum speedup observed was $51.8\times$.



■ **Figure 9** Speedups achieved by Algorithm 4 over iterative search.

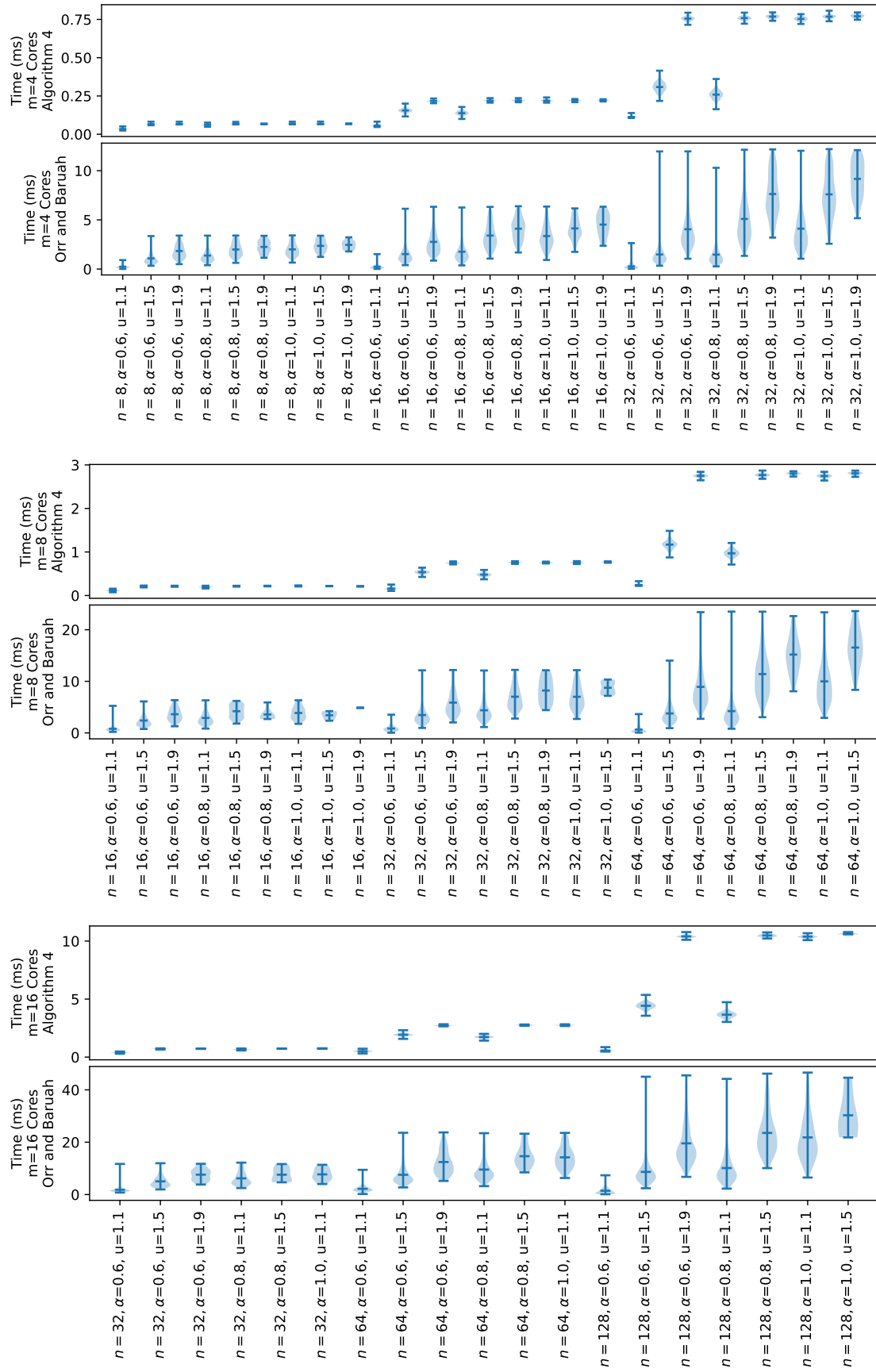
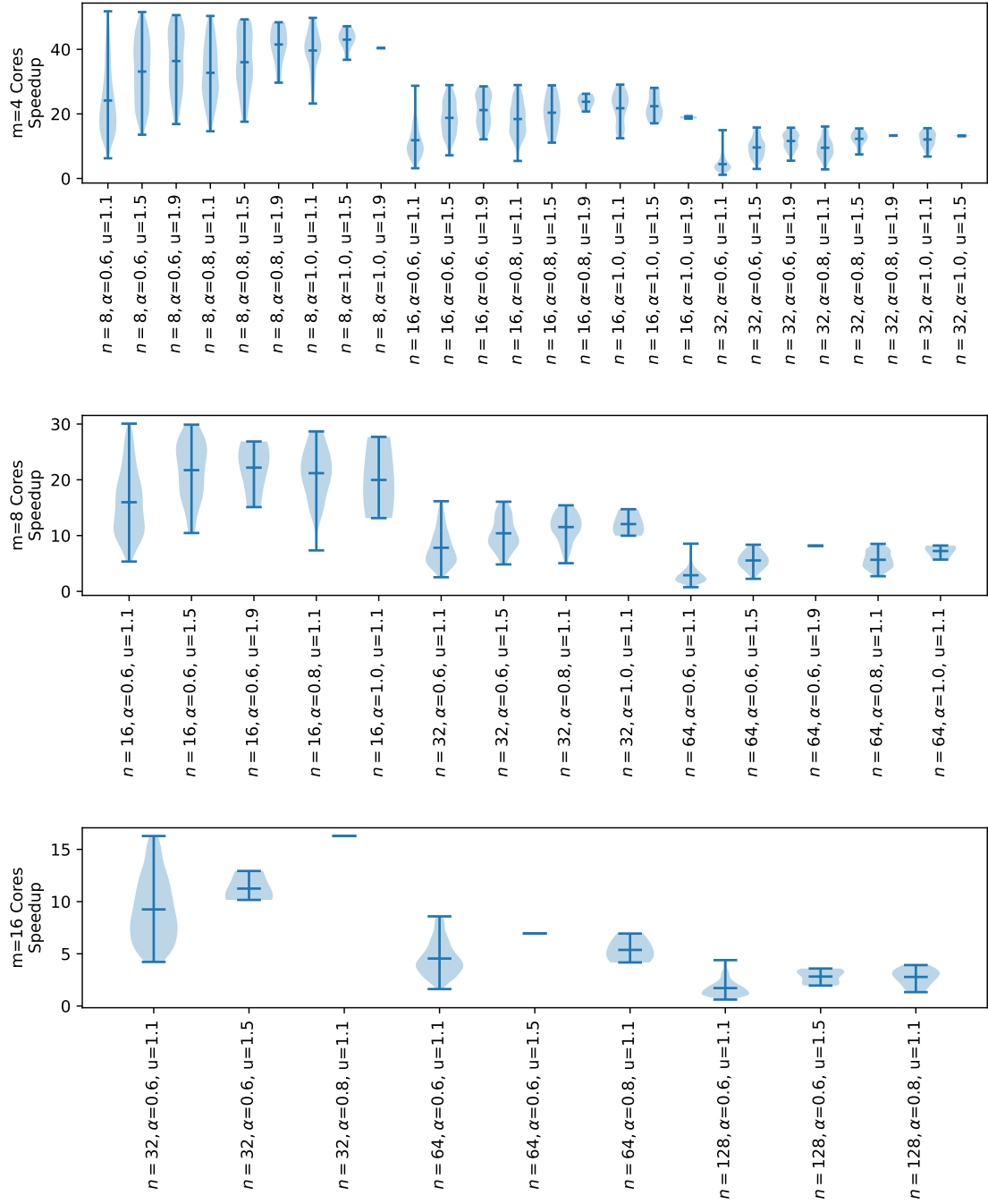
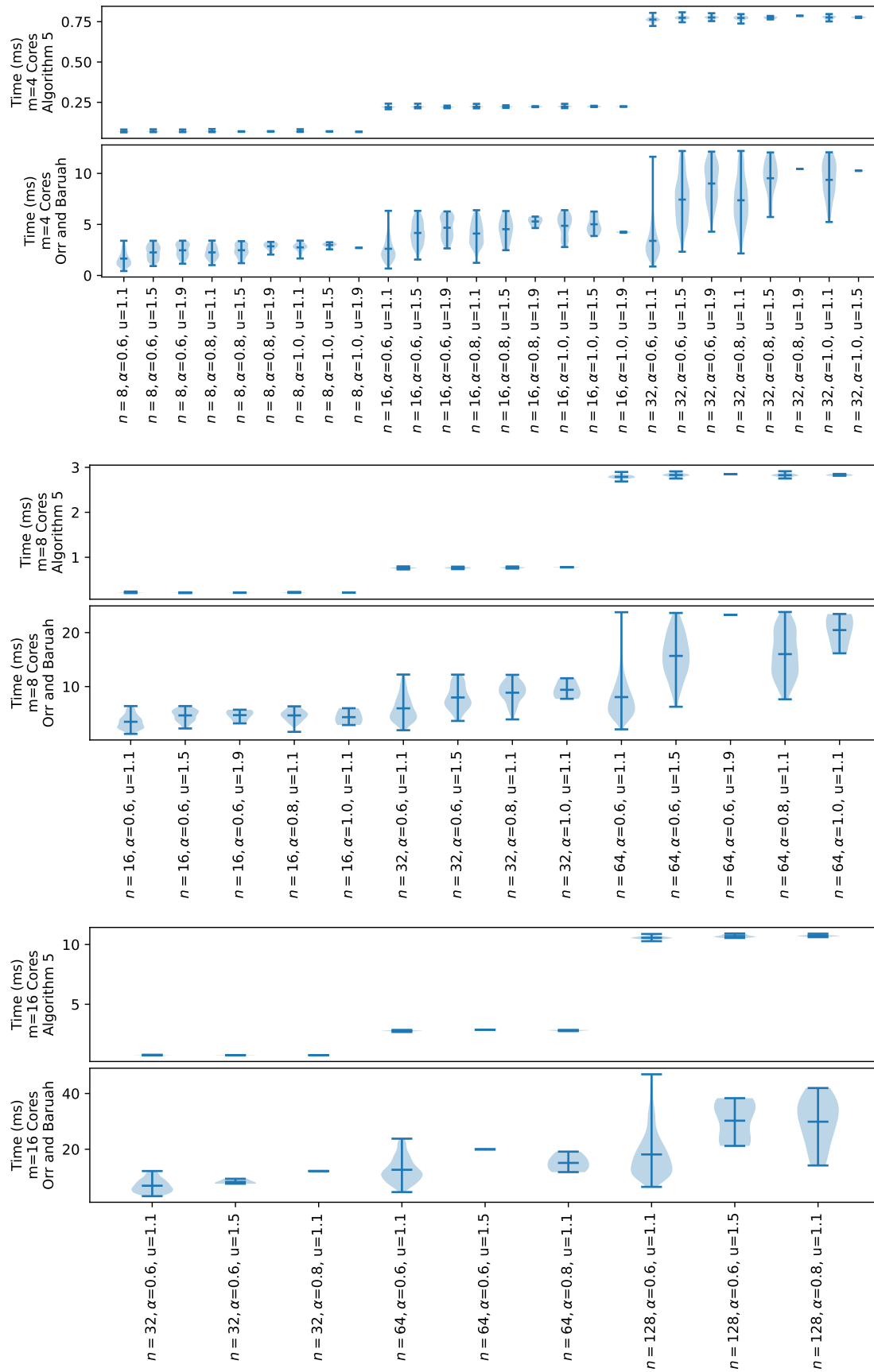


Figure 10 Execution times for global EDF compression algorithms.



■ **Figure 11** Speedups achieved by Algorithm 5 over iterative search.



■ **Figure 12** Execution times for global rm compression algorithms.

6 Conclusions and Future Work

This paper has presented and evaluated new approaches to elastic scheduling of implicit-deadline tasks. We began by considering the original model of Buttazzo et al. [12, 13] for utilization-bound scheduling on a uniprocessor. We compared the execution times of the original quadratic-time algorithm of Buttazzo et al. from [11, Figure 9.29] to the quasilinear time algorithm that we proposed in [26]. In practice, we demonstrated that the original algorithm is significantly slower to compress tasks compared to our proposed approach. However, initializing a sorted data structure dominates the execution time of our algorithm. As a result, for smaller numbers of tasks (up to 50), there is no clear advantage to using our algorithm over that of Buttazzo et al. (or vice versa) to compress complete task sets for schedulability. However, we observed that our algorithm achieves significant speedup during admission of new tasks. Because admission control involves online scheduling decisions, it is especially important that its overhead remains bounded. Therefore, in situations where initialization has already occurred — e.g., during admission of a new task or in response to changes in available utilization — our algorithm is significantly faster than the original approach. As these are more likely to be the scenarios encountered during dynamic online scenarios where the adaptation must have predictably low overheads, our proposed approach has clear advantages.

We then considered elastic scheduling for partitioned EDF. We observed that the approach of Orr and Baruah in [22, 21], which searches iteratively for the optimal “amount” of compression λ with some precision ϵ , may be improved by employing binary, rather than linear, search. We also demonstrated an application of our quasilinear time algorithm. While it runs even faster, it is often pessimistic, at times overcompressing or deeming a schedulable task set to be infeasible. Nonetheless, we can imagine scenarios where this may be desirable. For example, in mixed-criticality systems [30, 10], if a job of a safety-critical task overruns its expected execution time, jobs of less critical tasks are traditionally dropped. Elastic frameworks have been proposed instead where the periods of low-criticality tasks are extended to maintain the service-level guarantees required by critical jobs [24, 29]. In such a situation, the decision must be made as quickly as possible. If our faster approach overcompresses, this is no worse than the inelastic case where all low-criticality jobs are dropped anyway.

Finally, we proposed exact compression algorithms for global EDF and RM scheduling, where Orr and Baruah also proposed iterative search for λ . We evaluated both algorithms and found them to be considerably faster, while providing a more precise result. This makes them more suitable for use in online systems where utilizations may have to be adjusted during runtime.

Elastic scheduling remains a promising approach for dynamic adaptation in overloaded real-time systems. As future work, we will continue to extend the framework to different classes of task systems and their corresponding scheduling policies, including algorithm PriD [15], for which Orr and Baruah again propose an approximate search technique [22]; global [6, 5] and partitioned [4] scheduling of constrained-deadline tasks; semi-federated, reservation-based federated, and bundled scheduling of parallel tasks; and mixed-criticality systems.

References

- 1 S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, Jun 1996. doi:10.1007/BF01940883.
- 2 Sanjoy K. Baruah. Partitioned EDF scheduling: a closer look. *Real Time Syst.*, 49(6):715–729, Nov 2013. doi:10.1007/s11241-013-9186-0.
- 3 Sanjoy K. 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, Dortmund, Germany, June 7-8, 2023*, pages 67–75, New York, NY, USA, 2023. ACM. doi:10.1145/3575757.3575759.

- 4 Sanjoy K. Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Trans. Computers*, 55(7):918–923, 2006. doi:10.1109/TC.2006.113.
- 5 Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, 3-6 December 2007, Tucson, Arizona, USA, pages 149–160. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.31.
- 6 Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. Improved schedulability analysis of EDF on multiprocessor platforms. In *17th Euromicro Conference on Real-Time Systems (ECRTS 2005)*, 6-8 July 2005, Palma de Mallorca, Spain, *Proceedings*, pages 209–218. IEEE Computer Society, 2005. doi:10.1109/ECRTS.2005.18.
- 7 Marko Bertogna, Michele Cirinei, and Giuseppe Lipari. New schedulability tests for real-time task sets scheduled by deadline monotonic on multiprocessors. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers*, volume 3974 of *Lecture Notes in Computer Science*, pages 306–321. Springer, Springer, 2005. URL: https://doi.org/10.1007/11795490_24, doi:10.1007/11795490_24.
- 8 Enrico Bini and Giorgio C. Buttazzo. Measuring the performance of schedulability tests. *Real Time Syst.*, 30(1-2):129–154, may 2005. doi:10.1007/s11241-005-0507-9.
- 9 Tobias Blaß, Arne Hamann, Ralph Lange, Dirk Ziegenbein, and Björn B. Brandenburg. Automatic latency management for ROS 2: Benefits, challenges, and open problems. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2021, Nashville, TN, USA, May 18-21, 2021*, pages 264–277. IEEE, 2021. doi:10.1109/RTAS52030.2021.00029.
- 10 Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, 11 2018. doi:10.1145/3131347.
- 11 Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, New York, 4th edition, 2024. doi:10.1007/978-3-031-45410-3.
- 12 Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic task model for adaptive rate control. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*, pages 286–295. IEEE Computer Society, 1998. doi:10.1109/REAL.1998.739754.
- 13 Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic scheduling for flexible workload management. *IEEE Trans. Computers*, 51(3):289–302, mar 2002. doi:10.1109/12.990127.
- 14 Thomas L. Dean and Mark S. Boddy. An analysis of time-dependent planning. In Howard E. Shrobe, Tom M. Mitchell, and Reid G. Smith, editors, *Proceedings of the 7th National Conference on Artificial Intelligence, St. Paul, MN, USA, August 21-26, 1988*, volume 88, pages 49–54. AAAI Press / The MIT Press, 1988. URL: <http://www.aaai.org/Library/AAAI/1988/aaai88-009.php>.
- 15 Joël Goossens, Shelby Funk, and Sanjoy Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2):187–205, Sep 2003. doi:10.1023/A:1025120124771.
- 16 David Griffin, Iain Bate, and Robert I. Davis. Generating utilization vectors for the systematic evaluation of schedulability tests. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*, pages 76–88. IEEE, 2020. doi:10.1109/RTSS49844.2020.00018.
- 17 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, volume 130, page 43, 2015.
- 18 Chung Laung Liu and James W Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)*, 20(1):46–61, 1973. doi:10.1145/321738.321743.
- 19 Jane W.-S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, Riccardo Bettati, and Jen-Yao Chung. Imprecise computations. *Proc. IEEE*, 82(1):83–94, 1994. doi:10.1109/5.259428.
- 20 J. M. López, J. L. Díaz, and D. F. García. Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems*, 28(1):39–68, Oct 2004. doi:10.1023/B:TIME.0000033378.56741.14.
- 21 James Orr and Sanjoy Baruah. Algorithms for implementing elastic tasks on multiprocessor platforms: a comparative evaluation. *Real-Time Systems*, 57(1):227–264, 2021. doi:10.1007/s11241-020-09358-9.
- 22 James Orr and Sanjoy K. Baruah. Multiprocessor scheduling of elastic tasks. In Jérôme Ermont, Ye-Qiong Song, and Christopher D. Gill, editors, *Proceedings of the 27th International Conference on Real-Time Networks and Systems, RTNS 2019, Toulouse, France, November 06-08, 2019*, pages 133–142. ACM, 2019. doi:10.1145/3356401.3356403.
- 23 James Orr, Christopher D. Gill, Kunal Agrawal, Sanjoy K. Baruah, Christian Cianfarani, Phyllis Ang, and Christopher Wong. Elasticity of workloads and periods of parallel real-time tasks. In Yassine Ouhammou, Frédéric Ridouard, Emmanuel Grolleau, Mathieu Jan, and Moris Behnam, editors, *Proceedings of the 26th International Conference on Real-Time Networks and Systems, RTNS 2018, Chasseneuil-du-Poitou, France, October 10-12, 2018*, pages 61–71. ACM, 2018. doi:10.1145/3273905.3273915.
- 24 Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In Enrico Macii, editor, *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 147–152. EDA Consortium San Jose, CA, USA / ACM DL, 2013. doi:10.7873/DATE.2013.043.
- 25 Marion Sudvarg, Jeremy Buhler, Roger D. Chamberlain, Christopher D. Gill, James H. Buckley,

- and Wenlei Chen. Parameterized workload adaptation for fork-join tasks with dynamic workloads and deadlines. In *29th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2023, Niigata, Japan, August 30 - Sept. 1, 2023*, pages 232–242. IEEE, IEEE, 2023. doi:10.1109/RTCSA58653.2023.00035.
- 26 Marion Sudvarg, Chris Gill, and Sanjoy Baruah. Linear-time admission control for elastic scheduling. *Real-Time Systems*, 57(4):485–490, 10 2021. doi:10.1007/s11241-021-09373-4.
 - 27 Marion Sudvarg, Chris Gill, and Sanjoy K. Baruah. Improved implicit-deadline elastic scheduling. In *14th IEEE International Symposium on Industrial Embedded Systems, SIES 2024, Chengdu, China, October 23-25, 2024*, pages 50–57. IEEE, IEEE, 2024. doi:10.1109/SIES62473.2024.10768003.
 - 28 Marion Sudvarg, Daisy Wang, Jeremy Buhler, and Chris Gill. Subtask-level elastic scheduling. In *IEEE Real-Time Systems Symposium, RTSS 2024, York, UK, December 10-13, 2024*, pages 388–401. IEEE, IEEE, 2024. doi:10.1109/RTSS62706.2024.00040.
 - 29 Zhuoran Sun, Marion Sudvarg, and Christopher D. Gill. Elastic scheduling for graceful degradation of mixed-criticality systems. In *Proceedings of the 32nd International Conference on Real-Time Networks and Systems, RTNS 2024, Porto, Portugal, November 6-8, 2024*, pages 218–228. ACM, 2024. doi:10.1145/3696355.3699701.
 - 30 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*, pages 239–243. IEEE Computer Society, 2007. doi:10.1109/RTSS.2007.47.