Improved Implicit-Deadline Elastic Scheduling

Marion Sudvarg
Washington University in St. Louis
msudvarg@wustl.edu

Chris Gill
Washington University in St. Louis
cdgill@wustl.edu

Sanjoy Baruah
Washington University in St. Louis
baruah@wustl.edu

***Abstract*—Elastic scheduling provides a framework under which the utilizations of recurrent tasks are reduced by increasing their periods in response to system overload. First proposed by Buttazzo et al. in 1998 for uniprocessor scheduling of implicit-deadline tasks, elastic scheduling was extended to multiprocessor scheduling algorithms by Orr and Baruah in 2019. In this paper, we propose and analyze improvements to elastic scheduling of implicit-deadline tasks. *(i)* We evaluate a new algorithm that we proposed as a short note in the Real-Time Systems journal, and demonstrate that it allows for faster admission control than Buttazzo's algorithm when applied to uniprocessor and fluid scheduling. *(ii)* We propose and analyze faster elastic scheduling algorithms for partitioned EDF scheduling. *(iii)* We provide an exact algorithm for elastic scheduling under global EDF.**

I. Introduction

Elastic real-time scheduling models provide a framework for dynamic task adaptation to guarantee schedulability even on a system that becomes overloaded. First proposed by Buttazzo et al. [1], [2], elastic scheduling allows tasks to reduce ("compress") their utilizations, typically by increasing their periods. Each task has a maximum utilization, representing the desired service level at which it nominally executes given sufficient computational resources. Each adaptable or "elastic" task is also assigned an elastic parameter representing its relative adaptability (e.g., based on its importance). If the system becomes overloaded, elastic task utilizations are reduced proportionally to their elasticities until schedulable. Each task is also assigned a minimum utilization representing the minimum service necessary to maintain correct or safe execution below which its utilization can no longer be compressed.

While elastic scheduling models are therefore useful for adjusting a predefined set of tasks for execution on a resource-constrained system, Buttazzo's original elastic scheduling model 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 [1], [2]. 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 implicit-deadline tasks scheduled on both uniprocessor and multiprocessor systems. Prior algorithms can be classified into two categories. For scheduling algorithms with a **utilization bound**, a quadratic-time algorithm proposed by Buttazzo et al. [1], [2] for uniprocessor elastic scheduling finds an exact solution; this same algorithm was applied to multiprocessor fluid scheduling by Orr and Baruah [3]. For multiprocessor scheduling algorithms where analysis does not simply check total utilization (e.g., global EDF and partitioned EDF), 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.

NSF grants CPS-2229290 and CNS-2141256 and Washington University seed grant CC0001285 (PJ000030737) supported this work.

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. We presented such an algorithm in a short note published in the Real-Time Systems journal [4]. In §III, we evaluate its performance in comparison to Buttazzo's original quadratic-time algorithm and discuss its advantages. In §IV, we also propose an application to partitioned EDF scheduling, for which partitioning heuristics provide an (albeit pessimistic) utilization bound, then evaluate the speedups gained versus pessimism in the amount of compression applied.

Second, for the inexact 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, §IV demonstrates that we can find a result more quickly (compared to linear search) for partitioned EDF scheduling.

Third, in §V we propose a new **exact algorithm for elastic scheduling under global EDF**. Rather than searching for an amount of compression with tunable precision, an exact solution can be obtained in time quadratic on the number of tasks by modifying our algorithm in [4].

II. Background

A. Elastic Scheduling with Utilization Bounds

The elastic model for implicit-deadline tasks [1], [2] characterizes each task $\tau_i{=}(C_i, U_i^{\min}, U_i^{\max}, U_i, E_i)$ by five non-negative parameters. C_i is the task's worst-case execution time. $U_i^{\max}$ is its maximum utilization when executing at the desired service level in an uncompressed state. $U_i^{\min}$ is its minimum utilization, i.e., a bound on the amount its service can degrade. U_i is the task's assigned utilization, constrained by $U_i^{\min} \leq U_i \leq U_i^{\max}$. E_i is an elastic constant, representing "the flexibility of the task to vary its utilization" [1].

Under the original model proposed by Buttazzo et al. [1], [2], elastic scheduling was applied to uniprocessor scheduling algorithms with utilization bounds, e.g., EDF with its bound of 1, or rate-monotonic (RM) scheduling under Liu and Layland's bound [5]. It was since extended by Orr and Baruah [3] to multiprocessor fluid scheduling [6] where the utilization bound is equal to the number m of processors. Under these models, a task system $\Gamma = \{\tau_1, \ldots, \tau_n\}$ has a total uncompressed utilization $U_{\text{SUM}}^{\max} = \sum_{i=1}^{n} U_i^{\max}$ and a desired utilization U_D representing the utilization bound allowed by the scheduling algorithm in use. In the event of system overload, i.e., if $U_{\text{SUM}}^{\max} > U_D$, the model assigns a utilization U_i to each elastic task τ_i such that *(i)* $\sum_i U_i = U_D$, i.e., total utilization equals the bound; and *(ii)* if $U_i > U_i^{\min}$ and $U_j > U_j^{\min}$, then U_i and U_j must satisfy the relationship:

$$\left(\frac{U_i^{\max} - U_i}{E_i}\right) = \left(\frac{U_j^{\max} - U_j}{E_j}\right) \tag{1}$$

A task system Γ for which such U_i exist for all tasks is said to be *feasible*. Compression is realized by adjusting each task's period T_i according to its new utilization, i.e., $T_i = C_i/U_i$.

Buttazzo's Algorithm: Let Γ denote a feasible task system with $E_i > 0$ for all tasks $\tau_i \in \Gamma$, and consider the U_i values that satisfy the above conditions. The tasks in Γ may be partitioned into two classes — Γ_{VAR} (those tasks for which $U_i > U_i^{\min}$, so their utilizations can be compressed further if necessary) and Γ_{FIX} (those for which $U_i = U_i^{\min}$; i.e., their utilizations are now fixed). It has been shown [1, Eqn. 8] that for each $\tau_i \in \Gamma_{\text{VAR}}$, the utilization U_i takes the value

$$U_i = U_i^{\max} - \left(\frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}}\right) \times E_i \tag{2}$$

where $U_{\text{SUM}} = \sum_{\tau_i \in \Gamma_{\text{VAR}}} U_i^{\max}$, $E_{\text{SUM}} = \sum_{\tau_i \in \Gamma_{\text{VAR}}} E_i$, and $\Delta = \sum_{\tau_i \in \Gamma_{\text{FIX}}} U_i^{\min}$. Given a set of elastic tasks Γ, the algorithm of [1, Figure 3] starts out computing U_i values for the tasks assuming that they are all in Γ_{VAR} — i.e., their U_i values are computed according to Eqn. 2. 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_{VAR} to Γ_{FIX}; 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 $\mathcal{O}(n)$ time. Since an iteration is followed by another only if some task is moved from Γ_{VAR} to Γ_{FIX} and there are n tasks, the number of iterations is bounded from above by n. The overall running time for the algorithm is therefore $\mathcal{O}(n^2)$.

Our Improved Algorithm: In a short note in the Real-Time Systems journal [4], 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:

$$\phi_i \overset{\text{def}}{=} \left(\frac{U_i^{\max} - U_i^{\min}}{E_i}\right) \tag{3}$$

We proved in [4, Theorem 1] that in Buttazzo's algorithm of [1, Figure 3], tasks may be "moved" from Γ_{VAR} to Γ_{FIX} in order of their ϕ_i parameters. Assuming that the tasks are indexed such that $\phi_i \le \phi_{i+1}$ for all $i, 1 \le i < n$, 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 the ones in Γ_{FIX} before any of the ones in Γ_{VAR}. This can all be done in a single pass in $\mathcal{O}(n)$ time with the procedure in [4, Algorithm 1]. 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.

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

Though we proved better asymptotic time complexity in [4], we did not evaluate the algorithm's performance for realistic task sets. In §III, we perform this evaluation and extend the algorithm to fluid scheduling.

B. Scheduling Without a Utilization Bound

In addition to fluid scheduling, in [3], Orr and Baruah also extended elastic models to multiprocessor scheduling with partitioned EDF and global EDF. Each of these algorithms involves schedulability analysis that is more involved than simply checking total utilization against a bound that is constant in the number of tasks. To deal with this, they observed that the degree by which compression is applied to a task system can be quantified by the relationship in Eqn. 1. In doing so, they introduce a term λ that is representative of this relationship, and express the utilization U_i of each task τ_i as:

$$U_i(\lambda) \overset{\text{def}}{=} \max\left(U_i^{\max} - \lambda E_i, U_i^{\min}\right) \tag{4}$$

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:

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

which is equal to the value ϕ_i in Eqn. 3. As such, we may hereafter refer to ϕ_i interchangeably as $\lambda_i^{\max}$. For a complete set of tasks Γ we also denote the maximum compression that may be applied to the task system as:

$$\lambda^{\max} \overset{\text{def}}{=} \max_{\tau_i}\left(\lambda_i^{\max}\right) \tag{5}$$

The problem of elastic scheduling under Buttazzo's model [1], [2] can therefore be reduced to the problem of finding the minimum value of λ for which a set of tasks are schedulable. For partitioned EDF, global EDF, and algorithm PriD, Orr and Baruah propose an approximate search technique that iterates over values of λ in the interval $[0, \lambda^{\max}]$ with some "granularity" ϵ. For each value of λ, they assess schedulability, terminating the search once the compressed task system is deemed schedulable.

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

> 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 [7]. 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 $\left(\left\lfloor \frac{\lambda^{\max}}{\epsilon} \right\rfloor + 1\right)$ — the overall complexity is $\Theta\left(\frac{\lambda^{\max}}{\epsilon} \cdot (n \log n + n \cdot m)\right)$.

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 [8, Theorem 5] that a set Γ of implicit-deadline tasks is schedulable on m processors if:

$$\sum_{\tau_i \in \Gamma} U_i \leq m - (m-1) \cdot \max_{\tau_i \in \Gamma} \{U_i\} \tag{6}$$

Because the utilization bound includes the maximum among task utilizations, and because that maximum may change (indeed, the *task* with the maximum utilization may change) as utilizations are compressed, Buttazzo's algorithm cannot be applied directly. Orr and Baruah instead perform a linear search over the space of possible values of λ, terminating when Eqn. 6 holds true [3]. In §V, we present a polynomial-time algorithm that finds an exact solution, if one exists.

III. Utilization Bounds

This section considers elastic scheduling with utilization bounds; in particular, we consider EDF and RM scheduling on a uniprocessor and fluid scheduling on a multiprocessor.

A. Performance Evaluation

We begin by comparing the performance of our improved algorithm for elastic scheduling of implicit-deadline tasks from [4] to that of Buttazzo's algorithm in [1], [2].

Complexity of Buttazzo's Algorithm: As noted in §II-A, Buttazzo's elastic scheduling algorithm [1], [2] has worst-case execution time complexity that is quadratic in the number of tasks. Buttazzo et al. note in [1] 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 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

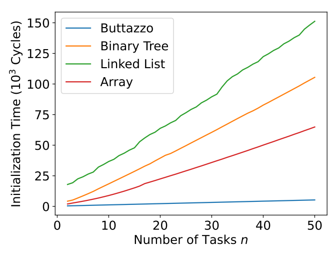
The total uncompressed utilization $U_{\text{SUM}}^{\max}$ is 2.0, but the desired utilization is $U_D = 1.0$. Then, in the absence of a constraint $U_i^{\min}$, the utilization U_i of each task τ_i will be assigned according to Eqn. 2:

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

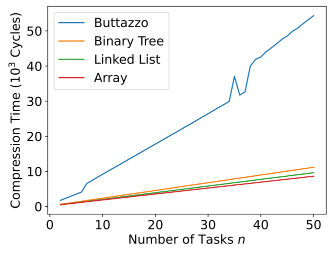
Computing for each task, we obtain $U_1 = U_2 = 0.8$ and $U_3 = -0.6$. While this set of assignments does achieve a total utilization of 1.0, these assignments are not valid: a negative utilization does not have semantic meaning.

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, Buttazzo's algorithm cannot be guaranteed to have better than quadratic time complexity in the number of tasks. On the other hand, our algorithm in [4] 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.

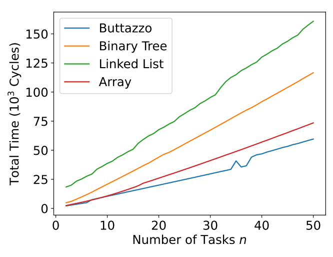
Implementation: Evaluations are performed on a Raspberry Pi 3 Model B+ with a 4-core ARMv8 Cortex-A53 running at 700 MHz (to prevent throttling — see [10], [11]) and 1GB of RAM. We compile Linux kernel 6.1.21 for the ARMv7l 32-bit ISA. We implement both algorithms in C++ and quantify execution time performance by measuring elapsed processor cycles, reading 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 GCC version 10.2.1 at optimization level `O0`, allowing 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



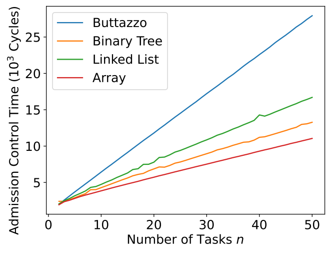
(a) Median Init Times.



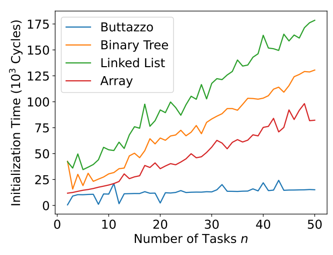
(b) Median Compress Times.



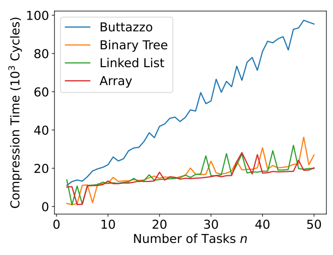
(c) Median Total Times.



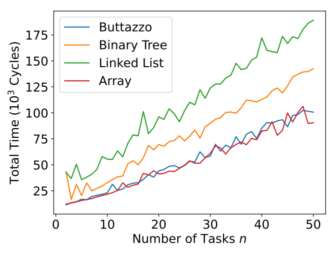
(d) Median Admission Times.



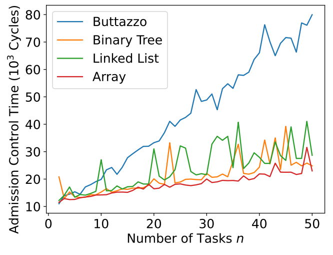
(e) Max Init Times.



(f) Max Compress Times.



(g) Max Total Times.



(h) Max Admission Times.

Fig. 1: Performance comparison between Buttazzo's algorithm [9, Figure 9.29] and our algorithm [4, Algorithm 1].

on that core at the highest real-time priority under Linux's `SCHED_FIFO` scheduling class.

We implement the version of Buttazzo's algorithm presented in [9, Figure 9.29], modified slightly to assign utilizations without updating periods. We compare this to three implementations of our algorithm [4, Algorithm 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).
- The set of tasks Γ is implemented as a balanced binary tree (`std::set`), sorted by ϕ_i. Constructing the set takes quasi-linear time, but subsequent insertion and removal requires only logarithmic time, while enabling sequential iteration over tasks in sorted order.
- 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 to maintain sorted order.

Generating Task Sets: We generate sets Γ of n tasks τ_i, generating 10 000 sets for each n in 2–50. Each set of tasks has a total maximum utilization $U_{\text{SUM}}^{\max}$ selected at random uniformly from $(1.0, 2.0]$ and a total minimum utilization $U_{\text{SUM}}^{\min}$ selected at random uniformly from $(0.0, 1.0]$. We apply the Dirichlet Rescale (DRS) algorithm [12] to distribute the total maximum utilization $U_{\text{SUM}}^{\max}$ in an unbiased random fashion across the $U_i^{\max}$ values for each individual task. We then apply the DRS algorithm to distribute the total minimum utilization $U_{\text{SUM}}^{\min}$ 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 $U_{\text{SUM}}^{\min}$ and *(ii)* each value $U_i^{\min}$ does not exceed the corresponding $U_i^{\max}$. Each task τ_i is then assigned an elasticity E_i at random, selected uniformly from the range $(0, 1]$.

Compression Time: We compress each set of tasks to a total utilization of 1.0 (to be EDF-schedulable on a single processor). We measure execution time by reading directly from the cycle counter, reporting elapsed CPU cycles. We separately measure the initialization ("Init") and compression ("Compress") times for each algorithm. For Buttazzo's algorithm, initialization only involves computing the $U_{\text{SUM}}^{\min}$ value and checking whether it exceeds U_D. The dominant contribution to our algorithm's execution time complexity is the sorting of tasks by their ϕ_i values; we therefore include in initialization time both the computation of U_{SUM} and E_{SUM} as well as the total time to calculate each task's ϕ_i value and establish the sorted order. For the array and list, the sort is performed over the complete data structure; for the binary tree, we insert tasks individually as their ϕ_i values are calculated.

The median and maximum times for the 10 000 sets of tasks generated for each size n from 2–50 are reported in Fig. 1. As expected, the time to initialize Buttazzo's algorithm is much faster than our algorithm, which has to sort tasks by their ϕ_i values. Of our three implementations of our algorithm, the linked list was the slowest to initialize, while the array was the fastest; we assume that this was due to the data locality and simplicity of managing the data structure.

Also, as expected, the compression time for Buttazzo's algorithm was much longer than for our algorithm. On average, the array tended to be the fastest, followed by the linked list, followed by the binary tree. This makes sense; while all three data structures enable linear time traversal, the array is the simplest to iterate and has the best data locality; the linked list is still simple, but requires following pointers between nodes, and does not have as good of locality; and the binary tree requires even more complex pointer chasing.

Most interesting, we observe that our algorithm does *not* strictly dominate Buttazzo's algorithm in total running time. In fact, in the average case, Buttazzo's algorithm performs better because of the low initialization overhead. In the worst case, both Buttazzo's algorithm and the array-based implementation of our algorithm dominate the other two implementations, but neither clearly dominates the other.

Nonetheless, we argue that our algorithm is better in practice. While there is not a clear advantage to using our algorithm to perform compression over a complete set of tasks, there is no clear *disadvantage* either. Furthermore, our algorithm performs better in situations where initialization has already happened, e.g. for online adjustment in response to changes in available utilization. The worst execution times that we observed for the array-based implementation of Sudvarg's algorithm were $3.45\times$ faster than those of Buttazzo's algorithm when just compressing tasks.

Task Admission: We modify our implementations of each algorithm to perform 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, then measure the time to compress after admitting the n^{th} task. Results are also illustrated in Fig. 1. We observe that, when admitting a new task, all implementations of our algorithm dominate Buttazzo's algorithm 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). The array-based implementation of our algorithm admits tasks $2.53\times$ faster than Buttazzo's algorithm in the worst case.

B. Extension to Fluid Scheduling

A set of tasks are fluid schedulable on m identical processor cores if and only if *(i)* their total utilization does not exceed m, and *(ii)* no individual task's utilization exceeds 1 [6]. Orr and Baruah therefore argued that, so long as each elastic task's maximum utilization $U_i^{\max} \leq 1$, Buttazzo's 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: Sudvarg's algorithm [4, Algorithm 1] may be used in place of Buttazzo's algorithm [9, Figure 9.29] to achieve faster compression (once initialized) and admission of new tasks. Evaluations show that the execution times of the tested implementations of both algorithms do not depend on $U_{\text{SUM}}^{\min}$ or $U_{\text{SUM}}^{\max}$ — the total minimum or maximum utilizations — nor the difference between them.[1] Therefore, the performance results illustrated in Fig. 1 should also extend to fluid scheduling.

IV. 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

[1]Plots omitted for length, but are available at https://sudvarg.com/SIES24.

Algorithm 1: Elastic_Partitioned_EDF(Γ, m)

```
Input: A list $\Gamma$ of elastic tasks to schedule on $m$ processor cores
Output: The value $\lambda$ 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      $\lambda^{\max} \leftarrow \max\left(\lambda^{\max}, \lambda_i^{\max}\right)$
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  $\lambda_{\text{HI}} \leftarrow \lambda^{\max}$, $\lambda_{\text{LO}} \leftarrow 0$
9  do
10     $\lambda \leftarrow (\lambda_{\text{HI}} - \lambda_{\text{LO}})/2$
11     if $\Gamma(\lambda)$ is schedulable on $m$ cores then $\lambda_{\text{HI}} \leftarrow \lambda$
12     else $\lambda_{\text{LO}} \leftarrow \lambda$
13 while $\lambda_{\text{HI}} - \lambda_{\text{LO}} > \epsilon$
14 return $\lambda_{\text{HI}}$
```

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 algorithm from [4, Algorithm 1] for compressing to this utilization bound.

A. Binary Search

We observe that a straightforward optimization may be applied to the approach of Orr and Baruah [3] summarized in §II. 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 $\Theta\left(\log \frac{\lambda^{\max}}{\epsilon}\right)$, as outlined in Alg. 1. Total time complexity is reduced to $\boldsymbol{\Theta\left((n \log n + n \cdot m) \cdot \log\left(\frac{\lambda^{\max}}{\epsilon}\right)\right)}$.

Alg. 1 uses the notation $\Gamma(\lambda)$ from Baruah [13], 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 Eqn. 4. 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 [3]. 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 $\lambda^{\max}$) tracks the smallest value of λ tested for which $\Gamma(\lambda)$ is schedulable, while λ_{LO} (initialized to 0) tracks the largest tested value for which $\Gamma(\lambda)$ is *not* schedulable. 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.

Optimality: We now discuss and prove results about the optimality of linear 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 remains schedulable when compressed more.

Lemma 1. *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 $\{\Gamma_1, \ldots, \Gamma_m\}$ 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, $U_i(\lambda') \leq U_i(\lambda)$, so

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

So there remains a partition where the condition holds. □

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.

Theorem 1. *The values of λ obtained by using the linear approach of Orr and Baruah [3] or the binary search in Alg. 1 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. **Linear Search**: 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 1 that $\lambda^* > \lambda - \epsilon$, which implies $\lambda - \lambda^* < \epsilon$.

Binary Search: 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 1, $\lambda^* > \lambda_{\text{LO}}$. Due to the algorithm's termination condition, we know that $\lambda_{\text{HI}} - \lambda_{\text{LO}} \leq \epsilon$, and so $\lambda - \lambda^* < \epsilon$. □

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 therefore within ϵ of each other. However, no such guarantee can be made if schedulability is determined by heuristic, as illustrated in Fig. 3. A corollary then follows from the above results.

Corollary 1. *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 $\lambda' > \lambda$.*

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 linear search might overcompress as well). However, as we show in §IV-C, binary search compresses, on average, only $0.054 \times \epsilon$ more than linear search for the sets of tasks we evaluated.

B. Application of Our Algorithm in [4]

In [14], it is observed that under the first fit and best fit heuristics, a set Γ of tasks τ_i are schedulable on m processor cores if their total utilization does not exceed $(m+1)/2$ and if no single task's utilization exceeds 1. Thus, [4, Algorithm 1] can be adopted by compressing to a desired utilization $U_D = (m+1)/2$, achieving compression in $\mathcal{O}(n \log n)$ time.

We note that $(m+1)/2$ is an *upper-bound* on the utilization required by these heuristics. Thus, the amount of compression due to this approach might be more than necessary to achieve partitioned EDF schedulability. It follows that the approach of Orr and Baruah [3], though slower, might achieve better results — both in terms of compressing utilizations less aggressively, and by identifying more schedulable task sets.

C. Evaluation

Implementation: We implement Alg. 1 in C++, compiling and measuring execution times using the same settings as for the algorithms in §III-A, running them on the same Raspberry Pi 3B+. For each value of λ tested, we attempt to find a schedulable partition by employing the best fit decreasing then first fit decreasing bin backing heuristics. The algorithm terminates if either is successful.

Generating Task Sets: We generate synthetic task sets according to Orr and Baruah's methodology in [3]. We measure each implementation's time to compress the tasks to run on platforms with $m = 4$, 8, and 16 identical cores. For each value 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 constrained to be no more than a parameter $\alpha \in \{0.6, 0.8, 1.0\}$. Each set of tasks has a total maximum utilization $U_{\text{SUM}}^{\max}$ of $u \times m \times \alpha$, where $u \in \{1.1, 1.5, 1.9\}$. For each unique combination of m, n, α, and u, we generate 1000 sets of tasks.

We use the DRS algorithm [12] to distribute the total maximum utilization $U_{\text{SUM}}^{\max}$ across individual $U_i^{\max}$ values. Individual minimum utilizations $U_i^{\min}$ are assigned at random, selected uniformly from the range $(0, U_i^{\max}]$. Elastic coefficients E_i are selected at random uniformly from $(1, 5]$.

Linear versus Binary Search: We begin by comparing the linear algorithm from Orr and Baruah [3] to the binary search in Alg. 1. For each set of tasks, we compute $\lambda^{\max}$ using Eqn. 5, then search for the optimal λ with granularity $\epsilon = \lambda^{\max}/1000$ (the same value tested in [3]).

Fig. 2 shows, for each combination of u and α, the speedup achieved by binary over linear search. (Dependence on values of n and m was less significant.) Task sets not requiring compression or deemed infeasible are excluded. Binary search 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 binary search is significantly faster. Median speedups for each combination were as high as $38\times$, while the maximum speedup observed was $86\times$.

Fig. 3 shows, for each combination of m and n, the distribution of differences between the amount of compression achieved by binary search (λ_{BS}) and linear search (λ_{LIN}), normalized by ϵ. We again exclude trivial or infeasible task sets. Where outliers extend beyond the plotted boundaries, the x-axis labels denote the maximum value. We observe that, although the values λ_{BS} and λ_{LIN} typically do not differ

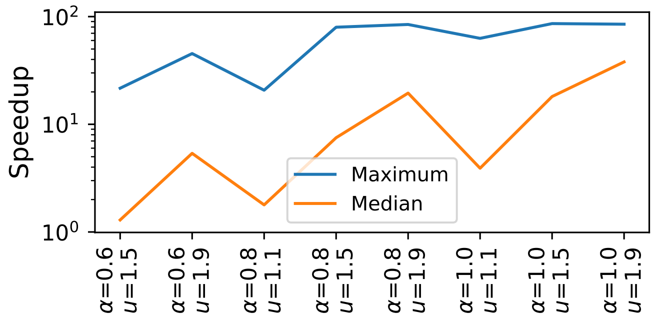


Fig. 2: Speedup achieved by binary versus linear search.

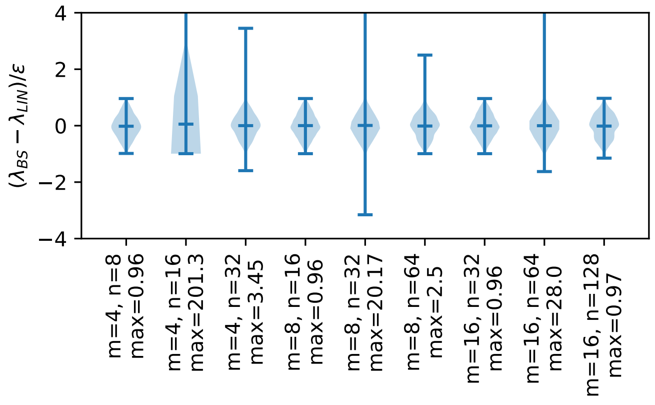


Fig. 3: Compression achieved by binary versus linear search.

by more than ϵ, there are cases where they differ by much more. For 16 tasks on 4 cores, with $\alpha = 1.0$ and $u = 1.9$, λ_{BS} exceeds λ_{LIN} by more than $200\times\epsilon$. Generally, we see that outliers occur where λ_{BS} is larger than λ_{LIN}. This makes sense due to the behavior of binary search: search proceeds downward in factors of 2 from larger values, and if $\Gamma(\lambda)$ is deemed unschedulable for some tested value of λ greater than the optimal λ^*, the binary search will continue to test larger values. Despite these outliers, **the average compression values agree closely**, differing by less than $0.06\times\epsilon$ for every considered combination. Therefore, given the significant speedups gained, binary search is an attractive approach.

Fast Compression: Though the binary search already improves execution time significantly, we expect that applying our quasilinear-time algorithm from [4] to be even faster, though we also expect it to be more pessimistic. We evaluate this hypothesis by comparing the number of task sets each algorithm deems feasible, the corresponding values of λ necessary for schedulability, and the time to find those values of λ. As in [3], to ensure a consistent comparison, we only compare λ values (and, in our case, execution times — this was outside the scope of the work in [3]) for those tasks deemed feasible.

The top plot in Fig. 4 shows the percentage of schedulable task sets identified by the binary search (BS) and our quasilinear-time algorithm (SGB) to compress to the heuristic utilization bound for every combination of α and u. The bottom compares the median and maximum execution time speedup gained by SGB over BS to the mean λ values achieved by each implementation. As in [3], λ values are normalized by $\lambda^{\max}$ to give a value in the interval [0,1]; this is necessary for comparing λ values across task sets.

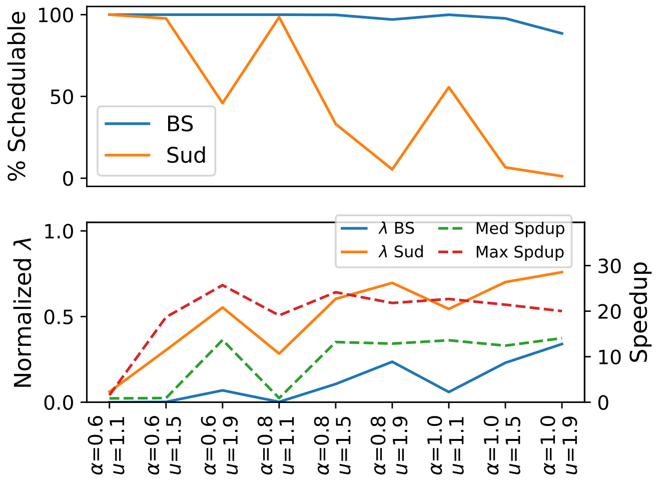


Fig. 4: Speed and schedulability tradeoffs.

We observe that SGB identifies fewer schedulable task sets, and for those it does identify as schedulable, it typically requires more compression. Nonetheless, at the cost of more pessimism, SGB achieves speedups observed to reach over 20×; this may be desirable where online decisions must be made rapidly. For example, in mixed-criticality systems [15], [16], elastic frameworks have been proposed to extend the periods of low-criticality tasks, rather than suspending them, in response to critical task overruns [17]. 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).

V. Global EDF

In this section, we present an exact polynomial-time algorithm for elastic scheduling under global EDF. Recall the schedulability condition [8, Theorem 5] from Eqn. 6 in §II. 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 $\sum_{\tau_i, i\neq j} U_i + mU_j \leq m$.

Theorem 2. *For a set Γ of elastic tasks, the amount of compression λ needed to satisfy the above schedulability condition can be found by finding λ for the condition $\sum_{\tau_i\in\Gamma^*} U_i \leq m$ for a set Γ^* where $\tau_j = (U_j^{\min}, U_j^{\max}, E_j)$ 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., $\sum_{\tau_i\in\Gamma^*} \max\left\{U_i^{\max} - \lambda E_i, U_i^{\min}\right\} = m$. Assume that $\tau_j^* \in \Gamma^*$ is parameterized as $\tau_j^* = (mU_j^{\min}, mU_j^{\max}, mE_j)$. Then:

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

Equivalently,

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

So $\sum_{\tau_i\in\Gamma, i\neq j} U_i(\lambda) + mU_j(\lambda) = m$. □

```
Algorithm 2: Elastic_Global_EDF($\Gamma, m$)
Input: A list $\Gamma$ of elastic tasks to schedule on $m$ processor cores
Output: The value $\lambda$ 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 Eqn. 3)
4  $\lambda \leftarrow \lambda^{\max}$
5  forall $\tau_i \in \Gamma$ do
6      $\tau_j \leftarrow (U_j^{\max} : mU_i^{\min}, U_j^{\min} : mU_i^{\min}, E_j : mE_i)$
7      $\Gamma^* \leftarrow \Gamma$, Remove $\tau_i$ and insert $\tau_j$ into $\Gamma^*$
8      ▷ Invoke our linear-time algorithm from [4]
9      $\lambda^* \leftarrow$ ELASTIC_COMPRESSION($\Gamma^*$,$m$)
10     if $U_j/m$ is the maximum compressed utilization and $\lambda^* < \lambda$
         then $\lambda \leftarrow \lambda^*$
11 end
12 return $\lambda$
```

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 [4, Algorithm 1]. However, as the task with the maximum utilization can change during compression, the utilization bound (the RHS of Eqn. 6) 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 Alg. 2.

We note that although our compression algorithm, as written in [4, Algorithm 1], does not return a value of λ, it can be easily modified to do so. From Equations 2 and 4 we can see that $\lambda = \left(\frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}}\right)$. This value is computed and tracked by our algorithm, and so it can be retrieved in constant time for use in Line 9 of Alg. 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 Alg. 2, constructing τ_j from τ_i takes constant time. From Eqn. 3 we can see that $\phi_j = \phi_i$, so τ_j can replace τ_i directly in constant time and Γ^* retains it sort order. Our compression algorithm runs in quasilinear time, but this time is dominated by sorting the tasks [4]. 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 $\boldsymbol{\mathcal{O}(n^2)}$.

VI. Conclusions and Future Work

We have evaluated the execution times of Buttazzo's and Sudvarg's elastic scheduling algorithms, demonstrating that Sudvarg's algorithm provides better performance after initialization, and is therefore more suitable for online adaptation. We have also proposed to use binary, rather than linear, search to find the "amount" of compression necessary for elastic task systems scheduled by algorithms without a simple utilization bound. We demonstrated significant speedups for heuristic schedulability analysis of partitioned EDF. Furthermore, we considered an application of Sudvarg's algorithm to partitioned EDF; though pessimistic, it enables even faster adaptation than binary search, and so may be appropriate where an online decision must be made rapidly (e.g., critical job overrun in mixed criticality systems). Finally, we have proposed a quadratic-time exact algorithm for elastic scheduling under global EDF. As future work, we will evaluate other applications of binary search for compression (e.g., to the hyperbolic bound for rate monotonic scheduling) and develop new polynomial-time algorithms (e.g., for elastic scheduling of algorithm PriD).

References

[1] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic Task Model for Adaptive Rate Control," in *IEEE Real-Time Systems Symposium*, 1998.

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

[3] J. Orr and S. Baruah, "Multiprocessor scheduling of elastic tasks," in *Proc. of 27th International Conference on Real-Time Networks and Systems*. ACM, 2019, pp. 133–142. [Online]. Available: https://doi.org/10.1145/3356401.3356403

[4] M. Sudvarg, C. Gill, and S. Baruah, "Linear-time admission control for elastic scheduling," *Real-Time Systems*, vol. 57, no. 4, pp. 485–490, 10 2021. [Online]. Available: https://doi.org/10.1007/s11241-021-09373-4

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun 1996. [Online]. Available: https://doi.org/10.1007/BF01940883

[7] J. M. López, J. L. Díaz, and D. F. García, "Utilization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems," *Real-Time Systems*, vol. 28, no. 1, pp. 39–68, Oct 2004. [Online]. Available: https://doi.org/10.1023/B:TIME.0000033378.56741.14

[8] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Systems*, vol. 25, no. 2, pp. 187–205, Sep 2003. [Online]. Available: https://doi.org/10.1023/A:1025120124771

[9] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. New York: Springer US, 2011, ch. Handling Overload Conditions, pp. 287–347.

[10] T. Blass, A. Hamann, R. Lange, D. Ziegenbein, and B. B. Brandenburg, "Automatic Latency Management for ROS 2: Benefits, Challenges, and Open Problems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021, pp. 264–277.

[11] M. Sudvarg and C. Gill, "A Concurrency Framework for Priority-Aware Intercomponent Requests in CAmkES on seL4," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2022.

[12] D. Griffin, I. Bate, and R. I. Davis, "Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests," in *2020 IEEE Real-Time Systems Symposium (RTSS)*, 2020, pp. 76–88.

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

[14] ——, "Partitioned EDF scheduling: a closer look," *Real-Time Systems*, vol. 49, no. 6, pp. 715–729, Nov 2013. [Online]. Available: https://doi.org/10.1007/s11241-013-9186-0

[15] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," in *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, 2007, pp. 239–243.

[16] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, 11 2017. [Online]. Available: https://doi.org/10.1145/3131347

[17] H. Su and D. Zhu, "An Elastic Mixed-Criticality Task Model and Its Scheduling Algorithm," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013, pp. 147–152.