Leibniz Transactions on Embedded Systems

Volume 10 | Issue 2 | December 2025

Special Issue on Industrial Real-Time Systems

Edited by
Daniel Casini
Qingxu Deng
Zhishan Guo
Wei Jiang
Haibo Zeng

ISSN 2199-2002

LITES Special Issue Editors

Daniel Casini
Scuola Superiore Sant'Anna, Pisa, Italy
daniel.casini@santannapisa.it

Qingxu Deng
Northeastern University, Shenyang, China
dengqx@mail.neu.edu.cn

Zhishan Guo
North Carolina State University, Raleigh, NC, USA
zguo32@ncsu.edu

Wei Jiang
University of Electronic Science and Technology of China, Chengdu, China
weijiang@uestc.edu.cn

Haibo Zeng
Virginia Tech, Virginia, USA
hbzeng@vt.edu

ACM Classification 2012
Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Real-time systems; Computing methodologies → Planning and scheduling

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
Online available at
https://www.dagstuhl.de/dagpub/2199-2002.

Publication date
December 2025

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://dnb.d-nb.de.

Aims and Scope
LITES aims at the publication of high-quality scholarly articles, ensuring efficient submission, reviewing, and publishing procedures. All articles are published open access, i.e., accessible online without any costs. The rights are retained by the author(s).

LITES publishes original articles on all aspects of embedded computer systems, in particular: the design, the implementation, the verification, and the testing of embedded hardware and software systems; the theoretical foundations; single-core, multi-processor, and networked architectures and their energy consumption and predictability properties; reliability and fault tolerance; security properties; and on applications in the avionics, the automotive, the telecommunication, the medical, and the production domains.

Editorial Office
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
LITES, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
lites@dagstuhl.de

Digital Object Identifier
10.4230/LITES.10.2.0

https://www.dagstuhl.de/lites

Contents

Leibniz Transactions on Embedded Systems, Vol. 10, Issue 2, Article No. 0, pp. 0:v–0:viii
Special Issue: Special Issue on Industrial Real-Time Systems.
Editor: Daniel Casini, Qingxu Deng, Zhishan Guo, Wei Jiang, and Haibo Zeng
Leibniz Transactions on Embedded Systems
LITES Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

List of Authors

Sanjoy Baruah (2)
Washington University in St. Louis, MO, USA

Franco Cordeiro (1)
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Christopher Gill (2)
Washington University in St. Louis, MO, USA

Laurent Pautet (1)
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Marion Sudvarg (2)
Washington University in St. Louis, MO, USA

Samuel Tardieu (1)
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Leibniz Transactions on Embedded Systems, Vol. 10, Issue 2, Article No. 0, pp. 0:vii–0:viii
Special Issue: Special Issue on Industrial Real-Time Systems.
Editor: Daniel Casini, Qingxu Deng, Zhishan Guo, Wei Jiang, and Haibo Zeng
Leibniz Transactions on Embedded Systems
LITES Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Integrating Multi-Level Mixed-Criticality into MCTS for Robust Resource Management

Franco Cordeiro
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Samuel Tardieu
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Laurent Pautet
LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Abstract

Managing actions with uncertain resource costs is a complex challenge, particularly in autonomous robot mission planning. Robots are often assigned multiple objectives with varying criticality levels, ranging from catastrophic to minor impacts, where failures can significantly affect system safety. Uncertainties in worst-case costs of resources, such as energy and operating time – the time it takes to carry out an action – further complicate mission planning and execution.

Monte Carlo Tree Search (MCTS) is a powerful tool for online planning, yet it struggles to account for uncertainty in worst-case cost estimations. Optimistic estimates risk resource shortages, while pessimistic ones lead to inefficient allocation.

The Mixed-Criticality (MC) approach, originally developed for real-time systems to schedule critical tasks by allocating processing resources under Worst-Case Execution Time (WCET) uncertainty, provides a framework of rules, models and design principles. We claim this framework can be adapted to autonomous robot mission planning, where critical objectives are met through analogous allocation of different kinds of resources such as energy and operating time despite uncertainties.

We propose enhancing MCTS with MC principles to handle uncertainty in worst-case costs across multiple resources and criticality of objectives. High-critical objectives must always be completed, regardless of resource constraints, while low-critical objectives operate flexibly, consuming resources within optimistic estimates when possible or being discarded when resources become scarce. This ensures efficient resource reallocation and prioritization of high-critical objectives.

To implement this, we present $(MC)^2TS$, a novel variant of MCTS that integrates MC principles for dynamic resource management. It supports more than two criticality levels to ensure that the most critical components meet the most stringent safety and reliability requirements, while also enabling robust resource management. By enabling replanning and mode changes, $(MC)^2TS$ improves MCTS's efficiency and enhances MC systems' adaptability to both degrading and improving resource conditions.

We evaluate $(MC)^2TS$ in an active perception scenario, where a drone retrieves data from distributed sensors under unpredictable environmental conditions. $(MC)^2TS$ outperforms MCTS by achieving more objectives, adapting plans when costs drop. It explores more objective sequences, minimizes oversizing, and enhances efficiency. Balancing safety and performance, it monitors robot battery, mission and objective resource constraints such as deadlines. Its robustness ensures low-critical objectives do not compromise high-critical objectives, making it a reliable solution for complex systems characterized by uncertain resource costs and critical objectives.

2012 ACM Subject Classification Computer systems organization → Embedded and cyber-physical systems; Computer systems organization → Real-time systems; Computing methodologies → Planning and scheduling

Keywords and phrases Embedded Systems, Safety / Mixed-Critical Systems, Real-Time Systems, Energy Aware Systems

Digital Object Identifier 10.4230/LITES.10.2.1

Acknowledgements This material is based upon work supported by the CIEDS (Interdisciplinary Centre for Defence and Security of Institut Polytechnique de Paris) and by the FARO project (Heuristic Foundations of Robot Swarms).

Received 2025-03-14 **Accepted** 2025-09-03 **Published** 2025-12-19
Editor Daniel Casini, Qingxu Deng, Zhishan Guo, Wei Jiang, and Haibo Zeng
Special Issue Special Issue on Industrial Real-Time Systems

1 Introduction

Autonomous robot mission planning presents numerous challenges. In these missions, robots are assigned multiple objectives of varying criticality levels, which classify the impact of potential failures on system safety. More than two levels can be used to distinguish the failure impacts, ranging from catastrophic and hazardous to major and minor impacts [26]. The complexity increases due to uncertainties in worst-case resource cost estimations, such as energy, where rare factors can significantly affect expected resource consumption, impacting mission planning and execution [14]. *Energy* is not the only resource required to keep track of. The robot may have to meet strict constraints such as deadlines for mission completion or for highly critical objectives, making *operating time* another crucial resource to monitor. Under these conditions, the robot may allocate or reallocate the resources in order to prioritize critical objectives, potentially foregoing less critical ones.

In this work, we focus on operating time, the time to achieve an action or an objective (e.g., flight duration), which must be clearly distinguished from computational time (e.g., processing time). Unless explicitly stated, all references to *time* refer exclusively to *operating time*.

The case study we choose to illustrate our contribution concerns *active perception* missions. A drone actively gathers information about its surroundings. Multiple geographically distributed sensors collect data without being connected to a network. The robot retrieves data from each sensor via Bluetooth. It is crucial to complete this objective before the sensor runs out of battery or memory. It may also face uncertainties in resource costs, such as the energy or time needed to move under unpredictable environmental conditions, such as strong winds or heavy rain. The drone must complete as many high-critical objectives as possible, followed by less critical ones.

The real-time systems community has studied the challenge of cost uncertainty. The real-time scheduling algorithms are notably affected by the uncertainty in *Worst-Case Execution Time* (WCET) estimates [1]. Indeed, the WCET evaluation often cannot be carried out with precision (see Figure 1). The system designers may obtain optimistic WCET through measurement-based methods while certification authorities may require pessimistic WCET obtained through static code analysis techniques [31]. Therefore, the WCET can be bounded by either optimistic or pessimistic estimates depending on safety guarantees required by the function. Relying solely on pessimistic WCET estimations in system design may result in unnecessary oversizing. Conversely, leaning towards optimistic WCET estimations can lead to scenarios where execution budget constraints are exceeded before task completion.

This challenge led to the emergence of the concept of *Mixed-Criticality* (MC) in real-time systems [5]. In this paradigm, tasks are systematically classified according to their criticality level, *i.e.* by assessing the consequences of a task failure. Higher critical tasks are usually imperative to the survival of the system, while lower critical tasks are related to services. Suppose a high-critical task must guarantee a failure probability of 10^{-9}. Its pessimistic WCET estimates (or a pessimistic time budget) must ensure that the probability of exceeding this execution time is below 10^{-9}. Alternatively, an optimistic WCET estimate (or an optimistic time budget) can be assigned, with a higher overrun probability (e.g., 10^{-5}). If the task does not complete within its optimistic budget, a mode change is triggered and lower-criticality tasks are either discarded or degraded to release execution time resources. This reallocation must provide the high-criticality task with its pessimistic budget, restoring its failure probability to 10^{-9}.

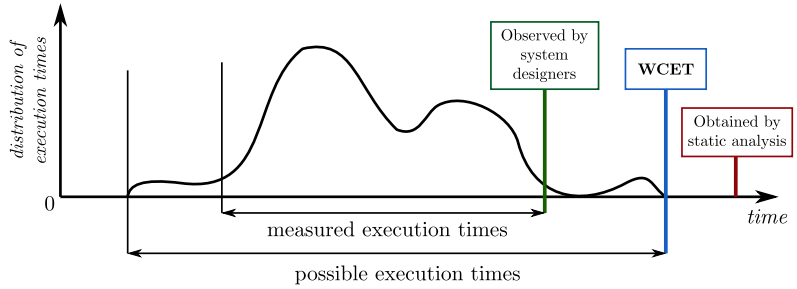


Figure 1 Optimistic and pessimistic WCET estimates.

In context of action planning, a variety of heuristics have been proposed to navigate the complexities of decision-making [14]. Amid these heuristics, *Monte Carlo Tree Search* (MCTS) emerges as a popular choice for online planning in which an agent needs to navigate through a tree-like search space to find an optimal path or make optimal decisions. In this context, MCTS is not inherently designed to balance guaranteeing execution of some objectives and maximizing overall objective completion. As it is based on the Monte Carlo method, its decision is strongly influenced by the model's most probable scenarios. Adapting MCTS to optimistic scenarios can lead to a resource shortage, while adapting to pessimistic scenarios can lead to oversizing the system.

To address these issues, our contribution is twofold. Firstly, we reformulate the MC approach in the context of uncertainty in resource costs. While the Mixed-Criticality (MC) approach was originally developed for real-time systems, where Worst-Case Execution Time (WCET) is the primary uncertain resource, we propose modeling the plan-building problem with both resource uncertainty and objective criticality as a generalized mixed-criticality problem. In this work, we broaden the scope of resource considerations to include energy and operating time requirements. Secondly, we design $(\text{MC})^2\text{TS}$, which integrates MC considerations into MCTS-based heuristics. It aims to anticipate changes in resource costs and adjust the decision-making process to optimize objective completion, resource utilization and system reliability in terms of criticality compliance. We accommodate an arbitrary number of criticality levels and incorporate both replanning and mode changes to enhance MCTS efficiency while also enabling MC systems to return to their initial mode after a mode change.

The rest of the paper is organized as follows. In section 2, we describe the system model and define some notations. Then we formulate the problem formally in Section 3. We describe our $(\text{MC})^2\text{TS}$ heuristic in Section 4. In Section 5, we evaluate our heuristic and demonstrate the improvement in performance brought by our approach.

2 System Model

We consider a robot which selects a sequence of actions $\boldsymbol{x} = (x_1, x_2, ..., x_k)$ to fulfill a set of objectives so that it maximizes the value of a global objective function $g(\boldsymbol{x})$. The objective function takes into account several parameters such as the reward r_i assigned to the completion of each objective x_i. Without loss of generality, we assume that only one (potentially complex) action is needed to achieve one objective. For example, this action might involve multiple movements followed by data loading to accomplish the objective. Thus, we use the terms *action* and *objective* interchangeably.

We define an ordered set of L criticality levels: $1 \prec 2 \prec \cdots \prec L$ based on the consequences of a failure. The system designer assigns a *criticality level* to an objective. At run-time, the system executes at a *criticality mode* ℓ allowing only future actions with criticality level in $\{\ell, \ldots, L\}$ to

execute. When a mode change occurs in mode ℓ the system makes a transition to mode $\ell + 1$. The system always starts its execution in the lowest mode 1. The set of actions of criticality ℓ is defined as $\boldsymbol{x}^\ell$.

In the following, we focus on the two-mode model. As noted by Burns and Davis [5], this abstraction captures essential system behavior under degraded conditions and provides a foundation for generalization to multiple criticality levels. In section 4.4, we generalize our contribution to an arbitrary number of criticality levels.

Each action x_i is associated with an actual cost $\bar{C}_i$, a tuple whose elements represent the different resources used by the robots. In our case, the cost tuple would be $\bar{C}_i = [\mathrm{d}_i, \mathrm{e}_i]$. $\bar{C}_i$[duration] (resp $\bar{C}_i$[energy]) designates the actual action duration d_i (resp energy e_i) it takes to run action x_i. The value of $\bar{C}_i$ is not known a priori; it is observed while performing x_i. Each sequence of actions $(x_1, x_2, \ldots, x_k)$ is associated with an actual accumulated cost $\bar{b}_k[r] = \sum_{h=1}^{k} \bar{C}_h[r]$ for resource r.

With two criticality modes, $C^{\text{LO}}_{j\to i}[r]$ represents the optimistic worst-case cost in resource r of an action x_i executed after x_j when the system executes in low-criticality mode (LO-mode). Similarly to $\bar{b}_k$, we define the optimistic worst-case accumulated cost of a sequence $(x_1, x_2, \ldots, x_k)$ as $b^{\text{LO}}_k = \sum_{h=1}^{k} C^{\text{LO}}_h[r]$. If $\bar{b}_k$ exceeds b^{LO}_k, the system switches to high-criticality mode (HI-mode). $C^{\text{HI}}_{j\to i}[r]$ represents the pessimistic worst-case cost in resource r of an action x_i executed after x_j when the system executes in HI-mode. We assume that $\forall r, C^{\text{HI}}_{j\to i}[r] \geq C^{\text{LO}}_{j\to i}[r]$. Similarly to b^{LO}_k, the pessimistic worst-case accumulated cost b^{HI}_k takes into account the worst case accumulated cost possible when a mode change occurs during the execution of any action of a sequence. The formula for this accumulated cost will be detailed later.

The cost of an action x_i in LO-mode or HI-mode is determined not only by the optimistic or pessimistic estimates made by the system designer but also by the previously executed action x_j. Indeed, as we consider actions as the entire process of reaching an objective and completing it, the position of the robot at the start of an action changes the total cost of executing it. Therefore, we represent the worst case cost in resource r of executing action x_i after action x_j as $C_{j\to i}[r]$. In the MCTS literature, the dependency of the worst-case cost on the previous action is often implicit and the notation is typically shortened into $C_i[r]$; however, we explicitly highlight this dependency as it is required for analyzing the complexity.

With two criticality modes, the system runs in either high (HI-mode) or low (LO-mode) critical mode. It starts in LO-mode and when any resource consumption exceeds the LO-mode value, a switch to HI-mode occurs. A high-critical action (HI-action) is carried out whether the system is running in LO-mode or HI-mode. Its cost is the one of the current mode. Conversely, future LO-actions are discarded in order to free up resources for the HI-actions currently in the plan. However, if the system changes to HI-mode while a LO-action is ongoing, this action will continue until completion. This difference with the classical MC model will be explained in section 4.

At last, a seamless transition between modes is required: execution must continue running smoothly as far as HI-actions are concerned. The plan must therefore support a safe transition to HI-mode, even under the most pessimistic assumptions. This requires that the transition be anticipated beforehand, with resource budgets allocated to complete actions under HI-mode, even if it was initiated in LO-mode. Similarly, a return to LO-mode is allowed only when there is sufficient confidence that HI-criticality tasks can safely be executed within LO-mode.

This also means HI-actions must not depend on the execution of LO-actions. Regarding dependencies between LO-actions, we do not specify a method to check for dependencies between them. We only assume that it is possible to check them during planning, *i.e.* during the expansion and simulation phases of MCTS, and during execution to check whether every LO-action required for the current objective have been executed.

We can generalize the notation for any criticality mode ℓ. $C^{\ell}_{j\to i}[r]$ represents the worst-case cost in resource r of an action x_i executed after x_j when the system executes in mode ℓ. We define b^{ℓ}_{k} for the worst-case accumulated cost of a sequence $(x_1, x_2, \ldots, x_k)$ executed in mode ℓ. We also assume that $\forall r, \forall \ell, C^{\ell+1}_{j\to i}[r] \geq C^{\ell}_{j\to i}[r]$.

We introduce a last refinement in the model. So far, resources are directly related to the robot, such as the energy available in its battery or the time budget for completing the mission. But, each individual objective may also have its own resource constraints, such as a time budget or a deadline before an objective runs out of memory or energy. These resource constraints may be known to the robot prior to the mission start if the system is not connected to a network. For these objectives, the system designer might also explore alternative degradation strategies to the disard model, such as the *elastic task model* (to extend the objective deadline) [6] or *weakly-hard resource constraints* (to extend the objective lifetime) [10]. Naturally, extending an objective's survivability comes at the cost of degraded service quality.

3 Problem Statement

Our goal is to generate a robust plan that optimizes the objective function g, accounting for actions whose resource costs are uncertain. Traditionally, such planning problems rely on heuristics like *Monte Carlo Tree Search* (MCTS). However, MCTS is not well-suited to handling uncertainty in resource costs, which can result in failure to meet critical objectives at runtime.

The *Mixed-Criticality* (MC) approach was originally developed for real-time systems, using scheduling algorithms to allocate processing time under WCET uncertainty while enforcing safe mode transitions. While this framework of models and design principles effectively manages temporal uncertainties, we propose adapting it to autonomous robot mission planning - where multiple resources (energy, operational time, etc.) must be allocated under uncertainty. However, traditional MC systems handle only execution time through specific schedulers. To generalize this approach, decision-making heuristics like MCTS must be extended to: (1) support multiple resource types, and (2) maintain safe mode transitions as required by the MC approach.

To incorporate MC into the MCTS heuristic, the model used in the plan building must be adapted. For this purpose, it is essential to analyze how action costs influence the planning process. Among the four MCTS phases (outlined in Section 4.1), cost uncertainty primarily affects the selection and simulation phases. To achieve a more adaptive planning strategy, these phases must be improved to better handle changes between different worst-case cost estimates. Additionally, since MC systems are capable of supporting multiple criticality levels, a comprehensive mixed-criticality approach must account for an arbitrary number of worst-case cost estimates.

Some MC properties must also be present in our solution. Transitions between action sequences must be seamless, ensuring that sufficient resources are always available for both current and future critical actions. To improve robustness, the addition of actions of lower criticality should not affect the overall performance of the actions of higher criticality. This requires a certain isolation between the criticality levels to ensure that the resulting plan still accomplishes similar performance in high criticality modes as before the addition of the new objectives. Finally, in scenarios where costs are precisely known and all actions are critical, our approach should produce results similar to those achieved using a standard MCTS heuristic.

Research Objectives

Resilience: We aim to develop a mission planning heuristic that maximizes the completion of critical objectives even under conditions where actual resource cost matches pessimistic assumptions. The resource budgets must be strictly enforced, thus guaranteeing safety.

Efficiency: We aim to maximize the completion of non-critical objectives under conditions where actual resource cost matches optimistic assumptions, thus minimizing overdimensioning the system. Under stable conditions (whether optimistic or pessimistic), the planning process should produce results comparable with those achieved using a standard MCTS heuristic.

Robustness: We aim to minimize the effect of adding new objectives on the number of higher-criticality objectives achieved. In particular, adding a new lowermost criticality level with new actions should not result in fewer higher-criticality objectives being achieved, even though the set of those higher-criticality objectives might change.

4 Approach

In this section, we present our $(MC)^2TS$ heuristic, an extension of MCTS heuristics to deal with uncertain constraints, such as the energy costs. Our $(MC)^2TS$ proposal is part of a larger mission control software:

- The *planner* builds a *plan* by selecting a sequence of actions that the robot can safely perform in the current environment while optimizing the reward (Subsection 4.2, Subsection 4.4) and respecting global and objective-specific budget constraints such as deadlines (Subsection 4.6).
- The *scheduler* executes the *plan* (Subsection 4.3) while constantly monitoring resource levels and switches to a higher criticality mode if resource consumption exceeds the allocated budget for the current mode.
- Periodically, *replanning* occurs, which prompts the planner to design a new plan that is better suited to the current conditions; this plan will then be transmitted to the scheduler.

Our solution involves running the MCTS heuristic (Subsection 4.1) while incorporating the MC approach during the selection and simulation phases. We show in Subsection 4.5 that the addition of MC to MCTS does not increase the complexity.

4.1 MCTS: Phases Description

MCTS is composed of four phases: *selection, expansion, simulation* and *backpropagation* [3]. These four phases are executed iteratively to incrementally grow a tree, as shown in Figure 2. Each node k of the tree represents a sequence of actions $\boldsymbol{x}_k = (x_1, x_2, \ldots, x_k)$ and contains data about the expected reward of the subsequent sequences of actions. During each iteration, a new leaf node is added to the tree, and the statistics within the ancestor nodes are updated accordingly. This iterative process repeats until a *computation budget* is reached. Note that a computation budget is a standard notation in MCTS literature to designate processing limits to which the heuristic expands the tree. It should not be misunderstood with operating time budgets or worst-case execution time budgets.

In the selection phase, the heuristic selects an expandable and rewarding node. An expandable node is defined as a node that has at least one child that has not yet been visited during the search. The heuristic begins at the root node of the tree and recursively traverses child nodes until an expandable node is reached. In order to check whether an action x_i is feasible at node k, we verify conditions such as whether the accumulated cost after executing the sequence $(x_1, x_2, \ldots, x_k, x_i)$ does not surpass the budget. Therefore, cost uncertainties affect the result of this phase.

During the expansion phase, a leaf node $k+1$ is added to the selected node k by choosing an action x_{k+1} among the actions previously computed as possible to execute after $\boldsymbol{x}_k$. This phase is thus indirectly affected by the uncertainty of costs.

In the simulation phase, the expected value of the reward of the new node is computed by simulating possible scenarios after the execution of $\boldsymbol{x}_{k+1}$ using a *rollout policy*. This policy can be a random policy or a heuristic tailored to the problem. A maximum horizon is defined to limit the

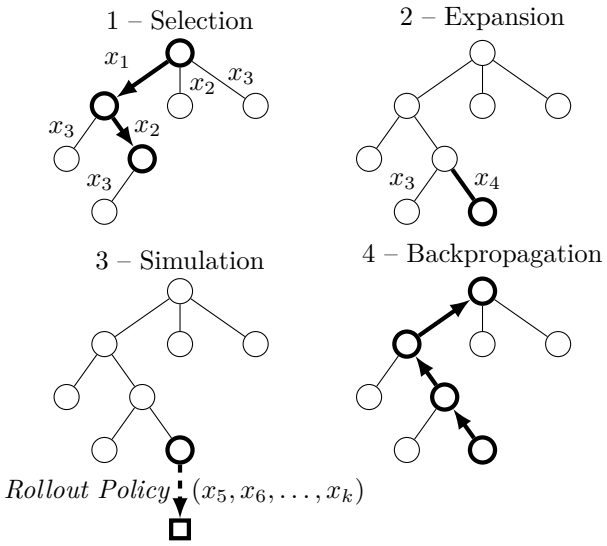


Figure 2 MCTS phases.

length of the action sequences simulated within this phase. During the execution of the rollout, it is important to know which sequences of actions are possible after $\boldsymbol{x}_{k+1}$, considering environment and budget. Therefore, cost uncertainties affect the result of this phase.

In the backpropagation phase, the result of the simulation phase for the new node is added to the statistics of all nodes along its path up to the root of the tree. These statistics are usually unbiased estimators of the rollout evaluations for the objective function. In this phase, the expected reward of each of the ancestor nodes is updated (backpropagated) with a more precise value, as we have more data about the child nodes resulting from the simulation. This phase is not impacted by the uncertainty of costs.

MCTS is an algorithm that can be greatly optimized by rebuilding the plan periodically during the mission execution, a process known as replanning. Usually, a replan occurs after an action has been completed and MCTS is executed to determine a new set of actions. As the system state is updated, the new plan is better adapted to the new possible outcomes of the mission execution. Replanning thus leads to a more efficient use of the remaining resources by adapting to the new reality of the situation. Thus, our approach can also benefit from replanning, since a change of mode is no substitute for updating the plan to reflect the current system state.

4.2 (MC)^{2}TS: Plan Building for Dual-Criticality Systems

We first examine the simpler scenario involving two criticality levels. In our approach, each action has two costs, one in HI-mode, one in LO-mode. Thus, each node k in the MCTS action tree has two associated accumulated costs b_k^{HI} and b_k^{LO}, with the accumulated costs for the root node b_0^{HI} and b_0^{LO} being zero. The computation of these accumulated costs depend on the action sequence $\boldsymbol{x} = (x_1, x_2, \ldots x_k)$ assigned to nodes $1, 2 \ldots k$.

For LO-objectives, the accumulated costs for any resource r are calculated as

$$b_k^{\text{LO}} = b_{k-1}^{\text{LO}} + C_{(k-1)\to k}^{\text{LO}} \tag{1}$$

$$b_k^{\text{HI}} = b_{k-1}^{\text{LO}} + C_{(k-1)\to k}^{\text{HI}} \tag{2}$$

whereas for HI-objectives, the accumulated costs are

$$b_k^{\text{LO}} = b_{k-1}^{\text{LO}} + C_{(k-1)\to k}^{\text{LO}} \tag{3}$$

$$b_k^{\text{HI}} = \max_{h_k \le j < k} \left(b_j^{\text{HI}} + C_{j\to k}^{\text{HI}} \right) \tag{4}$$

where $h_k = \max\{n \in [1, k-1] \mid x_n \in \boldsymbol{x}^{\text{HI}}\}$ with the convention $\max\{\varnothing\} = 0$

In LO-mode, as all actions are executed, the accumulated costs for HI and LO actions are computed the same way. This means the accumulated cost is simply the sum of $C_{(k-1)\to k}^{\text{LO}}$ of every node k in the branch, as shown in Equation 1 and Equation 3.

In case of a budget overrun, the system may change modes during the execution of a LO-objective and must proceed to execute the next HI-objective, if one exists. However, the cost of executing the HI-objective cannot be predetermined from an unknown intermediate system state, as the system state is only fully known at the end of each action. To address this, we ensure the current action completes by allocating it a budget in HI-mode, even if it is a LO-objective.

In HI-mode, since a LO-objective must complete its execution in HI-mode during a mode change, its cumulative cost in HI-mode is calculated by adding the cost of executing x_i in HI-mode to the cumulative cost of executing the previous action in LO-mode (as outlined in Equation 2).

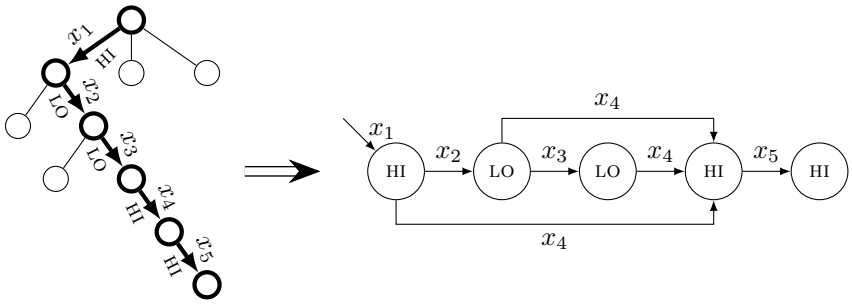


Figure 3 Possible mode changes in one branch. The criticality of each node is derived from the criticality of the action leading to that state.

In HI-mode, the accumulated cost associated with a HI-objective x_i must account for the worst-case scenario among several possible situations. Figure 3 show the possible executions for one branch: the system might have operated exclusively in HI-mode during the execution of the current action x_i (x_4); it could have already been running in HI-mode while executing the previous HI-objective x_{h_k} (x_1); or it might have transitioned from LO to HI-mode during the execution of a prior LO-objective x_l, where $h_k < l < i$ (x_2 or x_3). When node i is added to the tree, its accumulated cost is determined by taking the maximum cost among these potential scenarios (as described in Equation 4).

Note that in the cases where there are no uncertainties, *i.e.* $\forall i, C_{j\to i}^{\text{LO}} = C_{j\to i}^{\text{HI}} = C_{j\to i}$, then $\forall k, b_k^{\text{LO}} = b_k^{\text{HI}} = \sum_i C_{(k-1)\to k}$. This means that (MC)^{2}TS behaves exactly as MCTS when the action costs do not change between LO and HI mode assumptions.

MCTS or (MC)^{2}TS cannot guarantee that all HI-objectives will be achieved due to resource limits. To prevent LO-objectives from overriding HI-objectives, the rewards for HI-objectives should be greater than the sum of LO-objectives ones: $\forall x_i \in \boldsymbol{x}^{\text{HI}}, \sum_{x_j \in \boldsymbol{x}^{\text{LO}}} r_j < r_i$. Thus, HI-objective rewards become the greatest contributors to the objective function.

4.3 (MC)^{2}TS: Plan Execution for Dual-Criticality Systems

Figure 4 illustrates how the outcome from the (MC)^{2}TS step is mapped into instructions for the scheduler. In this example, the selected action sequence is (x_0, x_1, x_2, x_3). x_3 is a HI-objective, while x_0, x_1, x_2 are LO-objectives.

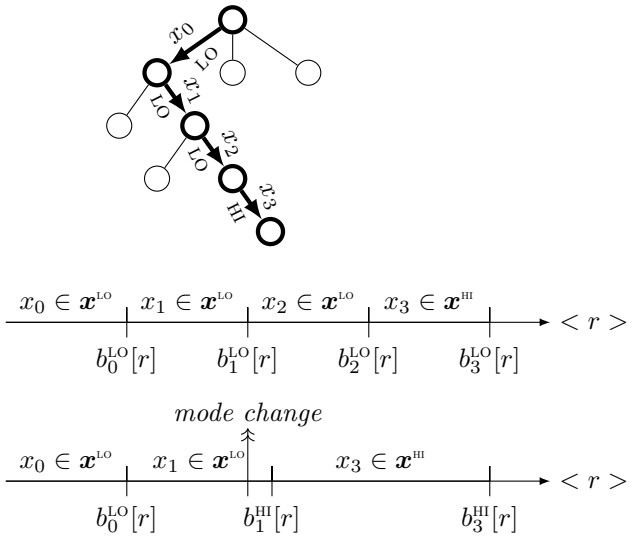


Figure 4 $(\text{MC})^2\text{TS}$ to scheduler instructions and constraints.

For every action x_k, the accumulated cost in LO mode $b_k^{\text{LO}}[r]$ from the $(\text{MC})^2\text{TS}$ tree is used as the budget corresponding to resource r. The systems starts in LO-mode. After executing any action x_k, the actual accumulated cost of each resource r (*e.g.*,, duration, or energy) is compared against $b_k^{\text{LO}}[r]$. If the budget is exceeded for at least one resource, the system switches (or stays) in HI-mode.

When the system is running in HI-mode, it discards the next action x_k if it is a LO-objective. This ensures that the resources consumption stays below the computed HI-mode accumulated cost $b_k^{\text{HI}}[r]$. In the setup shown on Figure 4, only x_0, x_1, x_2 can be dropped as x_3 is a HI-objective.

Figure 4 represents an example of a mode change. The first line shows the consumption of resource r in LO-mode, while the second one is a possible scenario where an action surpasses its LO-mode cost. Line $< r >$ represents the progressive usage of the available resources. When the mode change occurs, we let LO-objective x_1 complete its execution in HI-mode, LO-objective x_2 is discarded and HI-objective x_3 is executed after x_1 completion.

After transitioning to HI-mode, the system may incur costs similar to those in LO-mode, which can cause the actual accumulated cost for each resource r to fall below $b_k^{\text{LO}}[r]$ after executing action x_k. In this case, the system reverts to LO-mode and executes the upcoming LO-objectives as usual. Although this allows for potential frequent mode changes if actual accumulated costs fluctuate around the accumulated cost in LO-mode, mode changes introduce no overhead: the mode is only consulted before each action to determine if the next action can be executed. Furthermore, HI-actions will be completed as planned, and LO-actions are scheduled whenever it is safe.

However, the system needs to check whether any previously dropped LO-objective x_i is a dependency of the LO-objective x_k it is about to execute. If this is the case, then action x_k must be dropped as well. This ensures a seamless mode change to LO-mode, since LO-actions will have every required dependency before their execution. HI-actions do not have this issue as only LO-objectives can depend on LO-objectives.

Replanning, which involves rebuilding the plan during mission execution, often after completing an action, is a key process for optimizing the MCTS algorithm by generating new action sequences. When a replanning occurs, the resources are reallocated completely, and the system restarts in LO-mode after a new plan has been adopted. Note that replanning is no substitute for a mode change. Indeed, the mode change inherently adapts to new execution conditions by integrating

this adaptation during its plan building phase, whereas replanning requires creating a completely new plan to address the new conditions. It is important to note that the robot must have the knowledge of which actions were already completed in order to replan accordingly, both to know which actions are still available and which dependencies of future actions were already completed.

The frequency of replanning is a trade-off determined during system design. On the one hand, frequent replanning enhances robot autonomy and reduces data exchanges but increases the risk of skipping multiple LO-objectives when transitioning to HI-mode. This conservative approach may lead to overly cautious actions, as the robot will prioritize safety based on assumed worst-case resource consumption for HI-objectives.

On the other hand, frequent replanning facilitates better adaptation to real conditions. If an objective requires fewer resources than initially estimated, a more ambitious plan can be formulated during the replanning process. However, it is crucial that the replanning executes promptly to avoid delays that consume additional resources, which could otherwise be used for executing objectives.

Finally, if $b_k^{\text{HI}}[r]$ is exceeded for any resource r during or by the end of x_k execution, it implies either wrong system design or accidental operation outside safe conditions, such as due to a hardware failure. In such scenarios, $(\text{MC})^2$TS outperforms MCTS in safety by eliminating LO-mode actions, enabling the system to recover and return to its nominal operating conditions.

4.4 $(\text{MC})^2$TS: Plan Building for Multiple Criticality Systems

$(\text{MC})^2$TS can be adapted to $L > 2$ levels of criticality, with the lowest being the least critical. Each action x_k, assigned a criticality level χ_k, will have L worst-case costs and accumulated worst-case costs, one for each mode, even for criticality levels higher than χ_k. Indeed, actions x_k must be allocated a sufficient budget to guarantee their completion without interruption in the event of a mode change to a higher criticality level $\ell > \chi_k$.

To ensure that lower criticality actions do not take priority over actions of higher criticality levels, the system designer must ensure that the action rewards enforce $\forall x_i \in \boldsymbol{x^\ell}, \sum_{x_j \in \boldsymbol{x}(\chi_j < \ell)} r_j < r_i$.

▶ **Lemma 1.** *Transitioning from mode ℓ to a higher mode $m > \ell$, and then returning to mode ℓ has no impact on the worst-case accumulated cost b_k^ℓ.*

Proof. After transitioning to mode m to execute action x_i, the system may encounter costs to those of mode ℓ. As a result, the actual accumulated cost for each resource r may fall below the expected worst-case accumulated cost $b_j^\ell[r]$ after executing the subsequent action x_j (with $x_i \prec x_j \prec x_k$). In such cases, the system reverts to mode ℓ and continues executing subsequent objectives in that mode.

Therefore, for every resource r, the accumulated cost after executing x_j is less than or equal to the worst-case accumulated cost b_j^ℓ that would have been incurred without any transition to mode m. Consequently, transitioning to mode $m > \ell$ and returning to mode ℓ has no impact on the worst-case accumulated cost b_k^ℓ. ◀

▶ **Lemma 2.** *In a system where costs for any individual action increase with the criticality level, the worst-case accumulated cost is increasing with the criticality level:*

$$\forall j, \forall i, \forall \ell, C_{j\to i}^{\ell+1} \geq C_{j\to i}^{\ell} \implies \forall k, \forall \ell, \forall m \in \{1, \ldots, \ell\}, \quad b_k^m \leq b_k^\ell$$

This is true even if changes of the criticality mode led to skip some earlier actions in the plan.

Proof. Thanks to Lemma 1, we only need to consider and guarantee higher criticality modes during plan building.

Let $x_j^{\chi^j}$ denote an action x_j executed in mode chi^j. We consider the action sequence $(x_1^{\chi^1}, x_2^{\chi^2}, \ldots, x_k^{\ell})$, which results in b_k^{ℓ}, the worst-case accumulated cost in mode ℓ for node k. Note that the final action x_k is executed in mode ℓ.

From this sequence, we can derive another valid sequence $(x_1^{\chi^1}, x_2^{\chi^2}, \ldots, x_k^{\ell+1})$. This new sequence is valid because, if one or more mode changes occur during the execution of the final action x_k, our model enforces that the action must still be completed at any criticality level. In this case, we assume the system reaches level $\ell + 1$.

If $C_{k-1\to k}^{\ell+1} \geq C_{k-1\to k}^{\ell}$, then the accumulated cost of this valid sequence in mode $\ell + 1$ is greater than or equal to the worst-case accumulated cost in mode ℓ, i.e., b_k^{ℓ}. Thus, the worst-case accumulated cost in mode $\ell + 1$, denoted $b_k^{\ell+1}$, is also greater than or equal to b_k^{ℓ}. This result implies that the worst-case accumulated cost increases with the criticality level. ◀

▶ **Theorem 3.** *In a system with a number of criticality levels $L \geq 2$, the worst-case accumulated cost b_k^{ℓ} for node k in mode ℓ is:*

$$b_k^{\ell} = \begin{cases} b_{k-1}^{1} + C_{(k-1)\to k}^{1} & \text{if } \ell = 1 \quad (5) \\ \max\limits_{h_k^{\ell} \leq j < k} \left(b_j^{\ell} + C_{j\to k}^{\ell} \right), & \text{if } 1 < \ell \leq \chi_k \quad (6) \\ \max\limits_{h_k^{\chi_k} \leq j < k} \left(b_j^{\chi_k} + C_{j\to k}^{\ell} \right), & \text{if } \ell > \chi_k \quad (7) \end{cases}$$

where $h_k^{\ell} = \max\{n \in [1, k-1] \mid \chi_n \geq \ell\}$ with the convention $\max\{\varnothing\} = 0$, and $b_0^{\ell} = 0$ for all ℓ.

Proof. In mode 1, there are no prior mode transitions to consider, neither at the lowest criticality level (since mode 1 is the least critical) nor at higher levels (as ensured by Lemma 1). As a result, the accumulated worst-case cost corresponds to both Equation 1 and Equation 5.

In modes $\ell \leq \chi_k$, multiple possible mode changes must be considered for the computation of the worst-case accumulated cost, similar to Equation 2, and as described in Figure 3. Since any action with criticality level greater than ℓ will always execute, we define h_k^{ℓ} the earliest possible node to skip to node k as the last node with an action of criticality $\chi_h \geq \ell$. Indeed, nodes j before h_k^{ℓ} can at most skip to other nodes $n \leq h_k^{\ell}$ in mode ℓ. Every action x_j such that $j \geq h_k^{\ell}$ may trigger a mode change and skip to x_k. The maximum value of their accumulated cost must be considered, as well as the cost $C_{j\to k}^{\ell}$ to execute node k at mode ℓ. According to Lemma 2, the worst case accumulated cost for node j will be at the highest possible level ℓ. Thus, the worst-case accumulated cost at node k must consider the worst-case accumulated costs in nodes j at mode ℓ.

In modes $\ell > \chi_k$, the only possible mode change to the current mode ℓ occurs during execution, similar to the mode changes considered in Equation 2. Indeed, action x_k can only execute in that criticality mode if it was already executing when the mode change occurred. Therefore, the most resource-consuming case would be would be transitioning from a mode $\chi_j \leq \chi_k$ to ℓ. According to Lemma 2, the greatest accumulated cost will be at mode $\chi_j = \chi_k$. ◀

4.5 (MC)^{2}TS: Complexity Analysis

An important consideration is how much (MC)^{2}TS impacts the computation time of MCTS. MCTS has been previously successfully applied to real-time environments [8, 15, 21]. Therefore, if (MC)^{2}TS has a similar execution time, then its applicability is guaranteed. We first evaluate the time complexity of MCTS in our application. MCTS executes I iterations, each one executing the four phases. Consider the current depth of the tree is d, the number of possible actions is n, the

simulation horizon is h, the number of different scenarios simulated at each simulation phase is m, and the complexity of computing the accumulated worst-case costs for one node is c.

In the selection phase, we traverse d nodes in the tree, making $n-i$ comparisons at each node i, having a time complexity of $O^{slc}_{MCTS} = \mathcal{O}(\sum_{0}^{d-1}(n-i)) = \mathcal{O}(\frac{d^2}{2} + d(n-\frac{1}{2}))$. The expansion phase has a constant time complexity. In the simulation phase, we run one simulation traversing d nodes while computing costs, then m simulations traversing h nodes, having a time complexity of $O^{sim}_{MCTS} = \mathcal{O}((d+hm)c)$. Moreover, the rollout can be trivially parallelized, making its complexity in a processor with C cores become $\frac{hm}{C}c$ and $O^{sim}_{MCTS} = \mathcal{O}((d+\frac{hm}{C})c)$. In the backpropagation phase, we traverse the tree backwards by d nodes, having a complexity of $O^{bck}_{MCTS} = \mathcal{O}(d)$. Since d is bound by the number of actions n, as a completed objective will not be executed again, the complexity of one MCTS iteration in this application is:

$$O^{sim}_{MCTS} = \mathcal{O}((n+hm)c) \tag{8}$$

$$O_{MCTS} = \mathcal{O}(2n^2 + cn + chm/C) \tag{9}$$

The difference in complexity between MCTS and (MC)^{2}TS is the complexity c of computing the worst-case accumulated costs for a node, and thus the complexity of the simulation phase. For MCTS, we can consider c as $\mathcal{O}(1)$, and O^{sim}_{MCTS} becomes $\mathcal{O}(n+\frac{hm}{C})$. As for (MC)^{2}TS, we have to compute every possible cost $C^{\ell}_{j\to i}$ to compute the accumulated costs. Each action x_i of criticality χ_i in an action sequence has L possible completions. Following each completion in mode ℓ, the next action to be executed will be a different action x_j with a criticality $\chi_j \geq \ell$. The costs in modes $(\ell, \ell+1, \ldots, L)$ must be computed. Modes $\chi_j < \ell$ are not concerned, since when an action completes in mode ℓ, it will skip actions of criticality inferior to ℓ, and completing in lower modes will inevitably be less costly. Therefore, at each node of the chosen branch, there are $\sum_{\ell=0}^{L-1}(L-\ell) = \frac{L^2+L}{2}$ costs that will be computed throughout the execution, and the time complexity $O^{sim}_{(MC)^2TS}$ becomes:

$$O^{sim}_{(MC)^2TS} = \mathcal{O}\left(n\frac{L^2+L}{2} + \frac{hm}{C}\frac{(L^2+L)}{2}\right) = \mathcal{O}\left(n\frac{L^2}{2} + \frac{hm}{C}\frac{L^2}{2}\right) \tag{10}$$

$$O_{(MC)^2TS} = \mathcal{O}\left(2n^2 + n\frac{L^2}{2} + \frac{hm}{C}\frac{L^2}{2}\right) \tag{11}$$

This analysis leads to three conclusions. First, the complexity of both MCTS and (MC)^{2}TS depends quadratically on the number of objectives, n^2. Indeed, L typically ranges between 2 and 3. While some systems, such as those complying to the DO-178C standard for avionics [2], may use up to 5 criticality levels, larger values are uncommon. As a result, in most systems, n will be significantly larger than L^2, making $\mathcal{O}(2n^2)$ the dominant factor in the time complexity of the algorithm. Second, the complexity of (MC)^{2}TS depends quadratically on the number of criticality levels, L^2, meaning the increase in time complexity is impacted by the value of L. At last, given the existence of efficient real-time MCTS implementations, our complexity analysis demonstrates the feasibility of a real-time (MC)^{2}TS implementation.

4.6 (MC)^{2}TS: Resource Consumption, Mission and Objective Constraints

Our model integrate budget constraints at both the robot level (e.g., mission deadline) and the objective level (e.g., objective deadlines). Our analysis has so far focused on robot-centric resource budget constraints, particularly battery capacity and mission deadline. This timing constraint is enforced during planning by ensuring the accumulated worst-case time costs ($b^{\text{LO}}_k[time]$) never exceeds the mission deadline, effectively treating it as a resource budget constraint.

However, objective-level timing budgets are equally crucial. For example, a sensor may exhaust its energy or storage before the mission deadline. To address this, system designers can impose per-objective time budget constraints, requiring each objective to be completed by its specified deadline. Missing this deadline could result in the loss or degradation of the reward associated with achieving that objective.

Within the (MC)^{2}TS framework, objective timing constraints follow the same budget overrun management approach as mission timing constraints. For each objective x_i, the robot operates within its allocated worst-case time budget ($b_k^{\text{LO}}[time]$). When this budget is exhausted – indicating a deadline miss – the system transitions to HI mode. In this state, LO-objectives are automatically discarded while HI-objectives remain active, with the system ensuring their completion by reallocating time originally designated for lower-priority objectives.

However, discarding objectives because they are classified as LO-objectives might be seen as a disproportionate response, while categorising them as HI-objectives could be viewed as an inappropriate design choice. The MC approach proposes alternatives to the discard model, such as the elastic task model [6] or the weakly hard degradation model [10]. In the case of the elastic task model, when the system switches modes, LO-tasks in HI-mode have their period or deadline lengthened, freeing up resources for HI-tasks. In the same way, a weakly hard system will harvest fewer results in order to use fewer resources.

While our approach can be extended to incorporate mechanisms such as elastic deadlines or weakly hard constraints, this extension remains outside the scope of the current paper. Future work could explore its integration and assess its impact on system performance. In the next paragraph, we provide to the interested reader the main ideas on how to incorporate these alternative approaches into our model.

Since robots and objectives are not inherently connected in our framework, a robot detecting a resource overrun cannot directly instruct an objective to modify its behavior. Instead, the objective itself may transition to a degraded mode when approaching or missing its deadline. For example, upon reaching its initial deadline, an objective could autonomously adopt weakly hard constraints or extend its deadline. The robot can be preconfigured with knowledge of this degradation mechanism, eliminating any need for real-time communication between the robot and objectives. To prevent degraded behavior, any objective that transitions to this behaviour must receive a corresponding reduction in reward. Importantly, the revised reward value must still satisfy the criticality hierarchy constraint: remaining strictly greater than the sum of rewards for all lower-criticality objectives (Section 4.2).

5 Evaluation

In this section, we evaluate the performance of our algorithm in a data collection scenario with regard to the research objectives listed in Section 3:

- How much does (MC)^{2}TS improve the resilience of the system by increasing the number of completed HI-objectives when actual resource costs match those expected in HI-mode while guaranteeing resource constraints?
- How much does (MC)^{2}TS reduce system oversizing by increasing the number of completed LO-objectives when actual costs match those expected in LO-mode while guaranteeing resource constraints?
- How much does (MC)^{2}TS improve the robustness of the system by not altering the results of high critical levels when lower levels change.

We evaluated our solution regarding the research objectives in 6 simulations:

- *Maximize objectives under pessimistic assumptions* (Section 5.3): we compare our solution to an MCTS implementation that makes optimistic assumptions about costs, maximizing the number of objectives included in the plan and increasing efficiency. This experiment should demonstrate how considering only one possible cost per action can lead to the system being unsafe when actual costs match the pessimistic estimates.
- *Maximize objectives under optimistic assumptions* (Section 5.4): we compare our solution to an MCTS implementation that makes pessimistic assumptions about costs during the plan building. Indeed, due to the safety of the system, every LO or HI-objective in the plan can be executed during the mission even when actual costs match those expected in HI-mode. This experiment should demonstrate how considering only one possible cost per action can lead to the system being oversized when actual costs match the optimistic estimates.
- *Maximize objectives under middle case assumptions* (Section 5.5): we compare our solution to MCTS implementations that consider costs in a range between the optimistic costs and the pessimistic costs.
- *Enforce objectives timing constraints* (Section 5.6): we evaluate our solution's ability to ensure the deadline of HI-objectives and the impact of this constraint in the efficiency of the plan.
- *Minimize impact of low criticality objectives on higher critical ones* (Section 5.7): we compare our solution in two situations, one considering 2 and another considering 4 criticality levels. We evaluate the influence of adding intermediary levels of criticality with new objectives.
- *Computational budget influence* (Section 5.8): we evaluate whether our solution's performance, when compared to MCTS, is affected by the computational budget.

5.1 Problem specification

Let us consider a scenario in which a drone is deployed to collect data from a set of sensors distributed across an agricultural field. The drone operates under constraints related to both energy consumption and operating time (flight duration or mission deadline). Each of the sensors can be visited independently of the others. Since some sensors are approaching battery depletion, it is crucial to prioritize retrieving their data before they lose power. However, the energy and time required for navigation are subject to stochastic variations due to potential environmental disturbances, such as wind conditions or unforeseen obstacles that may impact the drone's motor efficiency.

The energy and time consumption associated with movement exhibit a lower bound under ideal conditions but increase as uncertainties arise. Specifically, the more adverse the conditions, the greater the expected cost. To model this variability, we represent the movement cost as a random variable following a half-normal distribution. The minimum possible cost is defined as half of the worst-case cost in LO-mode, denoted by C^{LO}. The standard deviation of this distribution is environment-dependent: under optimistic operating conditions, it is set to $\frac{C^{\text{LO}}}{10}$, whereas in more challenging scenarios, it is increased to $\frac{C^{\text{LO}}}{3}$, thereby increasing the probability of incurring higher movement costs.

5.2 Experiment setup

The environment is represented as a 100×100 grid, within which the robot can navigate freely. The movement and data retrieval costs considered in this study are presented in Table 1. A unit of movement corresponds to the side length of a single tile in the grid. In HI-mode, each HI-objective is permitted to consume up to twice the previously allocated budget, while LO-objectives are discarded. Energy costs are expressed as a percentage of the total battery capacity. To investigate

Table 1 Action costs.

Action	C^{LO} [duration]	C^{LO} [energy]	C^{HI} [duration]	C^{HI} [energy]
Move (one unit)	2.0	0.1%	4.0	0.2%
Retrieve data	5.0	1.0%	10.0	2.0%

the impact of resource constraints, the maximum energy budget is set to 60% of the total battery level. The decision-making process employs a random rollout policy in conjunction with the Upper Confidence Bound for Trees (UCT) [20] selection policy – a best-first approach that computes an upper confidence bound to evaluate the optimality of selected nodes.

The implemented action set consists of navigating to an objective location and retrieving data from the sensor at that location. The *computation budget*, defined as the number of times the Monte Carlo Tree Search (MCTS) executes the selection phase, is set to 600. The *horizon*, representing the maximum sequence length of actions tested during rollouts, is fixed at 5. Furthermore, the *C parameter* in UCT, which governs the balance between exploration of alternative paths and exploitation of the best current path, is set to 0.5. The cost of an action is calculated based on the distance between the current objective location and the next objective.

The robot initiates its trajectory from one corner of the grid and is expected to reach the diagonally opposite corner. To demonstrate the effects of a mode change, replanning occurs every two actions, allowing the robot operating under $(\text{MC})^2\text{TS}$ to execute portions of the mission in HI-mode. This prevents an immediate return to LO-mode after each mode transition, which would occur if replanning were performed after every action. A total of 50 distinct scenarios were generated, each comprising 15 randomly distributed objective locations, 4 of which are designated as high-critical objectives and 11 as low-critical objectives. Both MCTS and $(\text{MC})^2\text{TS}$ were executed 100 times per scenario, and performance was analyzed under varying time budget constraints.

The objective function considered is

$$g(\boldsymbol{x}, t) = \frac{\sum\limits_{i \in \boldsymbol{x}} r_i}{\sum\limits_{i} r_i} - \frac{t}{B[\text{time}]} \times 10^{-4} \tag{12}$$

where $\boldsymbol{x}$ is the set of actions already completed, r_i is the reward for completing action x_i, t is the time consumed from the beginning of the mission until the end of the last action and $B[time]$ is the total time budget available (or the mission deadline).

The $\frac{t}{B[time]}$ expression is used to prioritize plans that achieve the same objectives in less time. The 10^{-4} weight is used to ensure this value is always below the reward of achieving an objective. The $\sum\limits_{i} r_i$ factor ensures the total value is always below 1, which is necessary for UCT. For $(\text{MC})^2\text{TS}$, we use $b^{\text{LO}}[time]$ as t, as it is the accumulated cost in LO-mode. The reward for reaching the recharge site is 1.0, while the one for performing any other HI-action is 0.2 and for completing any LO-objective 0.0166. Note that $\forall x_i \in \boldsymbol{x}^{\text{HI}}, \sum_{x_j \in \boldsymbol{x}^{\text{LO}}} r_j < r_i$ as HI-objectives are our priority. As reaching the recharge site is the main HI-objective we want to ensure is in the plan, its reward is greater than the sum of the rewards of the other HI-objectives.

We want to assess each algorithm's ability to find a plan where the robot retrieves data from objective points and reaches the charging site before exhausting its allocated budget. A robot can fail to reach the recharging site due to unexpected high action costs. In this situation, the number of achieved objectives will be counted as zero.

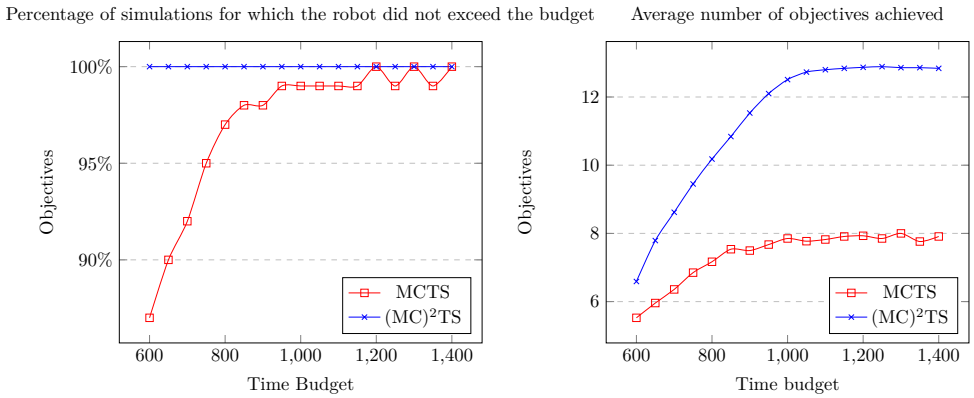


Figure 5 MCTS plans built with optimistic cost assumptions but executed under pessimistic costs.

5.3 Maximize objectives under pessimistic assumptions

We evaluate the performance of $(MC)^2TS$ in comparison to MCTS, where the latter is configured to generate plans that optimize the number of actions under optimistic assumptions. As the plans do not guarantee resource constraints in the most pessimistic scenarios, these plans may fail to ensure a safe return to the charging site.

If we consider our efficiency goal, for which we try to optimize the optimistic case, this MCTS configuration produces plans achieving a greater number of objectives. Consequently, if we compare its performance with that of $(MC)^2TS$ under conditions where the costs in mode LO are never exceeded, it will often achieve more objectives. However, if we consider conditions where the costs in mode LO are exceeded, the MCTS configuration generates unsafe plans that require more resources than are available. This is relevant for our resilience goal. Consequently, we evaluate how often MCTS plans fail to handle such situations. We also evaluate the number of total objectives achieved by $(MC)^2TS$ versus MCTS during these missions.

The experimental results are presented in Figure 5 .The first graph shows that optimistic MCTS frequently fails midway through mission execution under lower budgets, significantly reducing its effectiveness. In contrast, as illustrated in the second graph, $(MC)^2TS$ achieves more objectives even with higher budgets. This is attributed to its capability to revert to LO-mode once cumulative costs fall below optimistic assumptions. Consequently, $(MC)^2TS$ provides a safer and more reliable execution under pessimistic cost conditions.

5.4 Maximize objectives under optimistic assumptions

We evaluate the performance of $(MC)^2TS$ in comparison to MCTS, where the latter is configured to exclusively generate plans that never exceed the allocated resource budget.

When focusing our safety objective where we only consider conditions where actual costs are pessimistic, this MCTS configuration is optimal as it is designed to never surpass the allocated budget. Indeed, every action in the plan will always be executed. In the most pessimistic scenario, $(MC)^2TS$ will drop every LO-objective and execute a plan that only contains HI-objectives. It will compute the accumulated cost as the sum of $C^{\text{HI}}[r]$, just as the pessimistic MCTS configuration. Therefore, under conditions where actual costs are pessimistic, both approaches will execute similar amounts of HI-objectives, making them equal when it comes to safety.

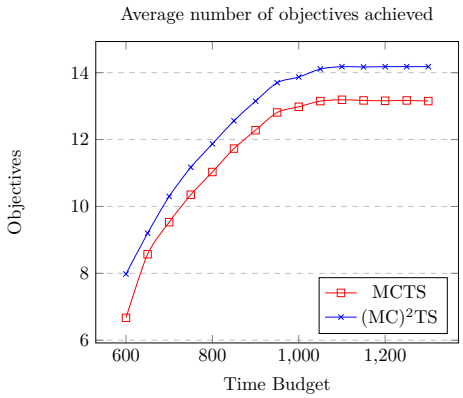


Figure 6 MCTS plans built on pessimistic costs executed under optimistic costs.

When considering an optimistic situation and our efficiency objective, MCTS may execute less LO-objectives than (MC)^{2}TS due to its pessimism. Thus, we evaluate the number of objectives achieved by (MC)^{2}TS compared with that of MCTS by simulating situations where the budget in LO mode is never exceeded.

The experimental results, shown in Figure 6, reach a peak in values due to the constrained energy budget. In this scenario, where MCTS is configured to generate plans exclusively based on pessimistic costs while operating under optimistic resource costs, (MC)^{2}TS outperforms MCTS when the budget is insufficient to achieve all objectives. This is because (MC)^{2}TS maintains an accumulated cost closer to that of execution under optimistic actual costs, allowing it to explore more action sequences by leveraging the previously conserved budget. As a result, (MC)^{2}TS makes less pessimistic assumptions about the environment, effectively reducing system oversizing and improving overall efficiency.

5.5 Maximize objectives under middle case assumptions

As shown previously, (MC)^{2}TS outperforms MCTS when MCTS uses either optimistic or pessimistic costs. In this subsection, we evaluate whether MCTS can outperform (MC)^{2}TS when considering costs that lie between these two extremes. Intermediate costs may offer a balanced trade-off, ensuring safety while maintaining efficiency. Such costs would be less pessimistic, preserving a degree of efficiency while providing the robot with a safety margin sufficient for most scenarios. To cover this possibility, we conducted the same simulations as those used for optimistic MCTS, but compared them with an MCTS implementation that incorporates higher costs for movement and data retrieval. We implemented the same simulation of a mode change used in the optimistic MCTS simulation to give these MCTS implementations a higher chance of being safe.

We simulated 4 costs considered by MCTS in the range between C^{LO} and C^{HI} for both time and energy resources. The costs were rounded for the time cost to be always an integer.

The experimental results are displayed in Figure 7. While increasing the considered cost reduces the unsafety of MCTS, no cost configuration below the pessimistic threshold makes it entirely safe. Additionally, for budgets exceeding 800, (MC)^{2}TS consistently achieves more objectives than MCTS. Thus, even MCTS implementations with less pessimistic assumptions fail to provide a better solution in terms of balancing safety and performance.

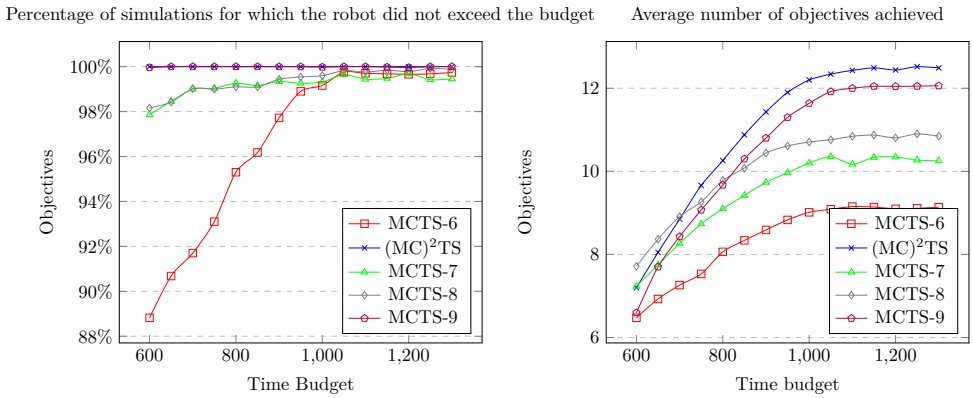


Figure 7 MCTS plans built on middle-case costs executed under pessimistic costs. Each MCTS number represents the cost for retrieving data considered.

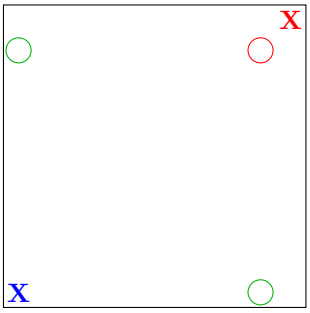


Figure 8 Objective distribution in the deadline scenario. The blue X represents the starting point, while the red X represents the finishing point. HI-objectives are represented as red circles while LO-objectives are green circles.

5.6 Enforce objective timing constraints

To assess the impact of respecting the deadlines for HI-objectives, increasing the resilience on (MC)^{2}TS's plan building, we simulate a scenario in which each HI-objective has a deadline of 500 time units. We compare it to the plan generated by (MC)^{2}TS for a scenario where no objective deadlines are imposed.

Figure 8 shows the distribution of the objectives in the map used. The HI-objectives are agglomerated close to the charging station. The robot must then haste to complete these objectives first to avoid missing their deadline in the pessimistic case. The LO-objectives are separated in two groups at opposite corners of the grid. In this scenario, the robot needs a higher budget to visit all LO-objectives than in the scenario where HI-objectives do not have any deadlines.

The experimental results are shown in Figure 9. In lower budgets, the robot only completes the HI-objectives, as it does not have enough budget to return and complete LO-objectives after. As the budget increases, the robot is then capable of visiting one LO-objectives group after completing the HI-objectives, and both cases are capable of completing every objective given enough budget. However, from a time budget of 800 onward, the robot in the scenario with deadlines completes less objectives. This is due to the robot having to traverse a longer path to respect the HI-objective deadlines.

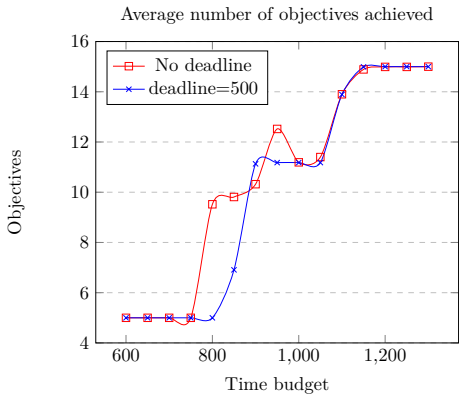


Figure 9 Simulation with HI-objective deadlines. The simulations considered an optimistic environment.

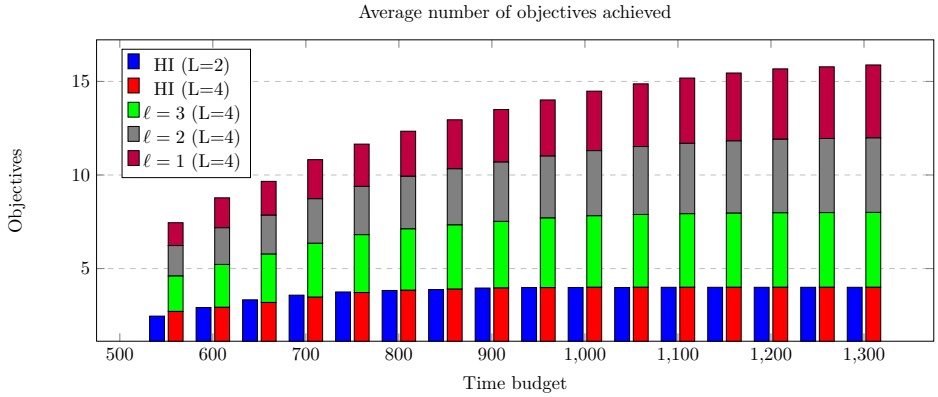


Figure 10 Simulation with 2 and 4 levels of criticality. HI represents actions of the highest level of criticality.

Therefore, we successfully guaranteed the completion of objectives before a deadline, increasing the resilience of the system. However, respecting objective deadlines even in HI-mode during planning may produce worse results under conditions where the execution is always in LO-mode. Thus, respecting deadlines in HI-mode should be reserved for HI-actions. For LO-actions, it is more coherent to respect deadlines only in LO-mode, even though they may execute in HI-mode.

5.7 Minimize impact of low criticality objectives on higher critical ones

To evaluate the effects of adding objectives of lower criticality levels, we compared $(MC)^2TS$ with $L = 4$ and with $L = 2$ criticality levels. To achieve our robustness goal, we expect that the addition of actions of criticality levels $\ell < 3$ should not change the average results in level 4.

To verify this result, we simulated both $(MC)^2TS$ implementations in random scenarios where each criticality level has 4 objectives. The simulations were conducted under conditions where resource costs are optimistic estimates and the battery budget was set to 100%.

The results are shown in Figure 10. When considering the different criticality levels, increasing the budget never reduces the average number of objectives achieved at any criticality level. The average number of objectives of highest criticality achieved are shown side by side. As the time budget increases, more objectives are achieved in both configuration. Therefore, the possibility of completing more objectives of lower criticality does not reduce the number of objectives completed

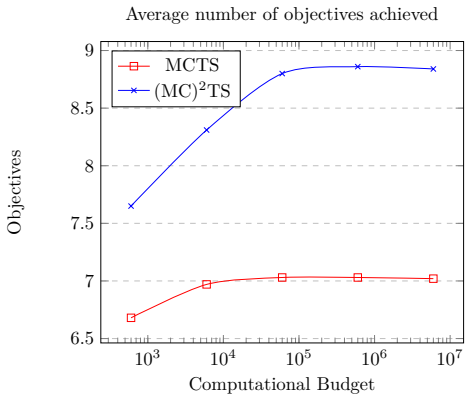


Figure 11 Simulation varying the computational budget under optimistic actual costs.

at the highest level. Moreover, this is similar for both implementations of (MC)^{2}TS with every time budget. This demonstrates the robustness of our algorithm, as adding actions of lower criticality does not influence the results for higher criticality levels.

However, the computational overhead increases with the number of criticality levels. Therefore, if replanning is done often, this might impact on the overall performance of the robot considering L=4, especially if computing costs is computationally heavy. Thus, adding more levels of criticality should be reserved for systems where the computation of extra costs will not significantly impact on the planning time.

5.8 Computational budget influence

An essential consideration is whether the results from previous experiments remain consistent across different computational budgets and how this value impacts the comparison between the heuristics. To evaluate possible degradation, we simulate the scenario using pessimistic plans with a fixed time budget of 600 while varying the computational budget.

The experimental results are shown in Figure 11. As the computational budget increases, the algorithms are capable of finding more optimal plans. With higher budgets, both algorithms degenerate into a standard tree search, and they reach a peak number of objectives. As (MC)^{2}TS cost assumptions are more optimistic, it consistently achieves more objectives on average than pessimistic MCTS. Therefore, (MC)^{2}TS benefits are achievable with any computational budget.

5.9 Conclusions

(MC)^{2}TS outperforms optimistic MCTS by achieving more objectives across varying budgets, leveraging its ability to revert to LO-mode when costs match those of LO-mode, ensuring safer and more reliable execution under conditions where resource costs match those of HI-mode. (MC)^{2}TS also outperforms pessimistic MCTS by maintaining costs closer to the optimistic ones, enabling exploration of more action sequences through conserved resources, reducing execution under HI-mode conditions, minimizing oversizing, and enhancing efficiency. Moreover, even under less pessimistic assumptions, (MC)^{2}TS successfully provides a better solution by effectively balancing safety and performance. Therefore, our solution is better adapted to uncertain environments than MCTS, whether it is configured with an adjusted objective function or built only on pessimistic assumptions. (MC)^{2}TS also monitors multiple uncertain resources, such as energy and action duration, while ensuring that critical objective deadlines are respected. Finally, (MC)^{2}TS demonstrates its robustness, as incorporating lower-criticality actions does not compromise the results for higher-criticality levels.

6 Related Works

In this section, we explore the literature on MCS and MCTS, as these two domains play a key role in our proposed framework for optimizing robot operations.

6.1 Mixed-Criticality Systems

In mixed-criticality systems, researchers have focused on Real-Time Scheduling [5]. As we consider Decision Making Heuristics, we rather focus on the different system models than on the MC scheduling algorithms themselves.

The majority of the mixed-criticality models consider a system with two criticality levels, even though the original paper by Vestal [30] generalizes the approach to a system with multiple levels of criticality. However, many papers extend their approaches to multiple levels, as it is important for common multi-criticality avionics and automotive standards [4]. Fleming extended the Adaptive Mixed Criticality scheduling methods (AMC-rtb and AMC-max) to an arbitrary number of criticality levels [9]. When tasks are non-preemptive, similarly to our model, Novak et al. proposed a static scheduling approach to minimize the schedule makespan when the system has an arbitrary number of criticality levels [24].

Two mechanisms are usually considered to address a resource failure event: either discard the LO-tasks in HI-mode, or degrade the quality of LO-tasks [4]. Gu et al. have criticized the strategy of discarding LO-tasks in HI-mode [12] and propose an unused budget reclamation scheme. Though resource-efficient to some extent, it adopts a pessimistic outlook, degrading the overall efficiency potential. A few studies have proposed to attempt the minimum service level of LO-tasks in HI-mode. Liu et al. propose continuing to execute LO-tasks in HI-mode but with a smaller WCET [23]. However, such an alternative is outside the scope of our case study. Several works also propose increasing the tasks' period in HI-mode [18, 28, 6], but our system is not periodic.

In the context of energy-aware mixed-criticality systems, some studies propose Dynamic Voltage and Frequency Scaling (DVFS) or DVFS with Earliest Deadline First (EDF-VD) scheduling [23, 17]. While they aim to reduce energy consumption by gracefully degrading LO-tasks in HI-mode, they lack inherent prioritization of HI-tasks over LO-tasks. In these studies, DVFS serves merely a mechanism for degrading the execution of a LO-task. Our approach differs in that it considers energy as an uncertain parameter of the problem, and not just as a resource to be managed. Therefore, we treat energy (as well as action duration) as a first-class citizen within the MC system, equally important as execution time.

6.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) has proven to be pivotal in addressing complex decision-making problems, such as gaming, planning, optimization, and scheduling [16, 3]. Its exploration-exploitation properties make it particularly valuable in path planning applications. Dam et al. extended the application of Monte Carlo methods to improve exploration strategies for robot path planning [8]. Kartal et al. proposed hybrid approach, combining MCTS with Branch and Bound for multi-robot action allocation problem with time windows and capacity constraints [19]. These constraints and time windows only consider one lower and upper bound, which differs from the MC approach.

Most existing studies consider a known environment where cost is fixed. Given a dynamic environment, Patten et al. have proposed a method for using different samples of the robot's current knowledge to simulate future events [25]. However, it does not consider the uncertainties

associated with the robot's actions. Unlike other approaches that may consider the uncertainty of the rewards of an action [3, 29, 7, 22], we specifically focus on the uncertainties associated with the robot's resources, introducing a novel aspect of uncertain costs.

Additionally, we highlight mode-change and replanning in $(MC)^2TS$. We demonstrate how they are complementary techniques that increase the safety and efficiency. This emphasizes our adaptability even in the face of future cost changes.

7 Conclusions and Perspectives

In this article, we address the challenge of action planning under uncertain action costs and criticality of objectives, a common issue in complex critical systems where cost overestimation often leads to oversized systems, wasted resources, and reduced performance. To tackle this, we reformulate the Mixed-Criticality (MC) approach, originally from the real-time community, in two key ways. First, we generalize it to handle multiple resources, such as energy and action duration, beyond its initial focus on execution time. Second, we adapt it for use in Monte Carlo Tree Search (MCTS) rather than Real-Time Scheduling.

We introduce $(MC)^2TS$, an extension of MCTS tailored for mixed-criticality systems. This involves adapting the action planning system model to the MC framework and extending cost evaluation to align with the different criticality modes of MC systems. Our evaluation demonstrates significant advantages in reducing system oversizing while handling uncertain costs. Using an active perception example, we show that $(MC)^2TS$ enables robots to anticipate environmental changes and adapt cost estimates dynamically.

$(MC)^2TS$ outperforms both optimistic and pessimistic MCTS by achieving more objectives across varying budgets, leveraging its ability to revert to LO-mode when costs drop below optimistic assumptions and maintaining costs closer to the optimistic ones. This enables exploration of more action sequences through conserved resources, reduces pessimistic assumptions, minimizes oversizing, and enhances efficiency. $(MC)^2TS$ effectively balances safety and performance, even under less pessimistic assumptions, while monitoring uncertain resources like energy and action duration in particular by respecting critical deadlines. Additionally, its robustness ensures that incorporating lower-criticality actions does not compromise higher-criticality outcomes, making it a reliable solution for complex systems characterized by uncertain resource costs and critical objectives.

In future work, we plan to extend our model to robot fleets and other autonomous, self-aware systems [11], enhancing distributed planning with the MC approach. We aim to design $(MC)^2TS$ as a real-time service and validate its schedulability using tools like Cheddar [27], demonstrating its effectiveness in real-world environments. Beyond energy and time considerations, an interesting perspective would be to incorporate additional resource constraints such as memory usage for both robotic systems and sensor networks. Another promising investigation would be to extend our approach to other planning paradigms, such as dynamic adaptative policy [13], with mixed criticality approach. This work paves the way for more adaptable, reliable, and efficient solutions in safety-critical and resource-sensitive applications.

References

1 Sanjoy Baruah. Mixed-criticality scheduling theory: Scope, promise, and limitations. *IEEE Design & Test*, 35(2):31–37, 2018. doi:10.1109/MDAT.2017.2766571.

2 Benjamin Brosgol. DO-178C: the next avionics safety standard. *ACM SIGAda Ada Letters*, 31(3):5–6, November 2011. doi:10.1145/2070336.2070341.

3 Cameron B. Browne et al. A survey of MCTS. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

4 Alan Burns and Sanjoy Baruah. Towards a more practical model for mixed criticality systems. In *Workshop on Mixed-Criticality Systems (colocated with RTSS)*, 2013.

5 Alan Burns and Robert Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys*, 50:1–37, November 2017. doi:10.1145/3131347.

6 G.C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *19th IEEE Real-Time Systems Symposium*, pages 286–295, 1998.

7 Alberto Castellini, Enrico Marchesini, Alessandro Farinelli, et al. Online monte carlo planning for autonomous robots: Exploiting prior knowledge on task similarities. In *AIRO@ AI* IA*, pages 25–32, 2019.

8 Tuan Dam, Georgia Chalvatzaki, Jan Peters, and Joni Pajarinen. Monte-carlo robot path planning. *Robotics and Automation Letters*, 7(4):11213–11220, 2022. doi:10.1109/LRA.2022.3199674.

9 Thomas Fleming. *Extending mixed criticality scheduling*. PhD thesis, University of York, 2013. URL: https://etheses.whiterose.ac.uk/id/eprint/5688/1/ExtendingMixedCriticalityScheduling.pdf.

10 Oliver Gettings, Sophie Quinton, and Robert I. Davis. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, RTNS '15, pages 237–246, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2834848.2834850.

11 Holger Giese et al. *Architectural Concepts for Self-aware Computing Systems*, pages 109–147. Springer, 2017. doi:10.1007/978-3-319-47474-8_5.

12 Xiaozhe Gu and Arvind Easwaran. Dynamic budget management and budget reclamation for mixed-criticality systems. *Real-Time Systems*, 55, July 2019. doi:10.1007/S11241-019-09330-2.

13 Marjolijn Haasnoot, Jan H. Kwakkel, Warren E. Walker, and Judith ter Maat. Dynamic adaptive policy pathways: A method for crafting robust decisions for a deeply uncertain world. *Global Environmental Change*, 23(2):485–498, 2013. doi:10.1016/j.gloenvcha.2012.12.006.

14 Zaharuddeen Haruna, Muhammed Bashir Mu'azu, Abubakar Umar, and Glory Okpowodu Ufuoma. Path planning algorithms for mobile robots: A survey. In *Motion Planning for Dynamic Agents*. IntechOpen, 2023.

15 Todd Hester, Michael Quinlan, and Peter Stone. Rtmba: A real-time model-based reinforcement learning architecture for robot control. In *2012 IEEE International Conference on Robotics and Automation*, pages 85–90. IEEE, May 2012. doi:10.1109/icra.2012.6225072.

16 Constantin Hofmann, Xinhong Liu, Marvin May, and Gisela Lanza. Hybrid monte carlo tree search based multi-objective scheduling. *Production Engineering*, 17:133–144, 2022.

17 Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Energy efficient dvfs scheduling for mixed-criticality systems. In *International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2014. doi:10.1145/2656045.2656057.

18 Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Run and be safe: Mixed-criticality scheduling with temporary processor speedup. In *DATE*, pages 1329–1334, 2015. URL: http://dl.acm.org/citation.cfm?id=2757122.

19 B Kartal, E Nunes, J Godoy, and M Gini. Monte carlo tree search with branch and bound for multi-robot task allocation symposium conducted at the meeting of the the ijcai-16 workshop on autonomous mobile service robots. *AAAI16_MCTS.pdf*, 2016.

20 Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006. doi:10.1007/11871842_29.

21 Sahar Leisiazar, Edward J. Park, Angelica Lim, and Mo Chen. An mcts-drl based obstacle and occlusion avoidance methodology in robotic follow-ahead applications. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 221–228. IEEE, October 2023. doi:10.1109/iros55552.2023.10342150.

22 Wei Li et al. A self-learning monte carlo tree search algorithm for robot path planning. *Frontiers in Neurorobotics*, 17, 2023.

23 Di Liu et al. EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees. In *Real-Time Systems Symposium (RTSS)*, pages 35–46, 2016.

24 Antonin Novak, Premysl Sucha, and Zdenek Hanzalek. Scheduling with uncertain processing times in mixed-criticality systems. *European Journal of Operational Research*, 279(3):687–703, 2019. doi:10.1016/j.ejor.2019.05.038.

25 Timothy Patten, Wolfram Martens, and Robert Fitch. Monte carlo planning for active object classification. *Autonomous Robots*, 42(2):391–421, 2018. doi:10.1007/S10514-017-9626-0.

26 Wind River. Arinc 653–an avionics standard for safe, partitioned systems. In *IEEE Seminar*, 2008.

27 Stéphane Rubini et al. Scheduling analysis from architectural models of embedded multi-processor systems. *SIGBED Rev.*, 11(1):68–73, February 2014. doi:10.1145/2597457.2597467.

28 Hang Su and Dakai Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *DATE*, pages 147–152, 2013. doi:10.7873/DATE.2013.043.

29 Maciej Swiechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mandziuk. Monte carlo tree search: a review of recent modifications and applications. *Artif. Intell. Rev.*, 56(3):2497–2562, July 2022. doi:10.1007/S10462-022-10228-Y.

30 Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 239–243, 2007. doi:10.1109/RTSS.2007.47.

31 Reinhard Wilhelm et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):1–53, 2008. doi:10.1145/1347375.1347389.

Improved Elastic Scheduling Algorithms for Implicit-Deadline Tasks

Marion Sudvarg[1]
Washington University in St. Louis, MO, USA

Christopher Gill
Washington University in St. Louis, MO, USA

Sanjoy Baruah
Washington University in St. Louis, MO, USA

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

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 2025-04-17 **Accepted** 2025-11-26 **Published** 2025-12-19

Editor Daniel Casini, Qingxu Deng, Zhishan Guo, Wei Jiang, and Haibo Zeng

Special Issue Special Issue on Industrial Real-Time Systems

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 el 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).
- E_i: An elastic constant, representing "the flexibility of the task to vary its utilization" [12].

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

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

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

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

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

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

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

Intuitively, this model represents each task as a spring, with a length corresponding to the utilization U_i and an elasticity corresponding to E_i. The total length of the springs, placed end-to-end, represents U_{SUM}. If this exceeds U_D, the schedulable bound, a force is applied to the system that compresses each task's utilization U_i proportionally to its elasticity, subject to the constraint[3] that the utilization remains no less than the specified minimum $U_i^{\min}$. This physical analogy is illustrated in Figure 1.

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

2.1.1 An Overview of the Algorithm of Buttazzo et al.

Let Γ denote a feasible task system with $E_i > 0$ for all tasks[4] $\tau_i \in \Gamma$, and consider the U_i values that satisfy the above conditions. The tasks in Γ may be partitioned into two classes – Γ_{VARIABLE} (those tasks for which $U_i > U_i^{\min}$, and which can therefore have their utilizations compressed further if necessary) and Γ_{FIXED} (those for which $U_i = U_i^{\min}$; i.e., their utilizations are now fixed).

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

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

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

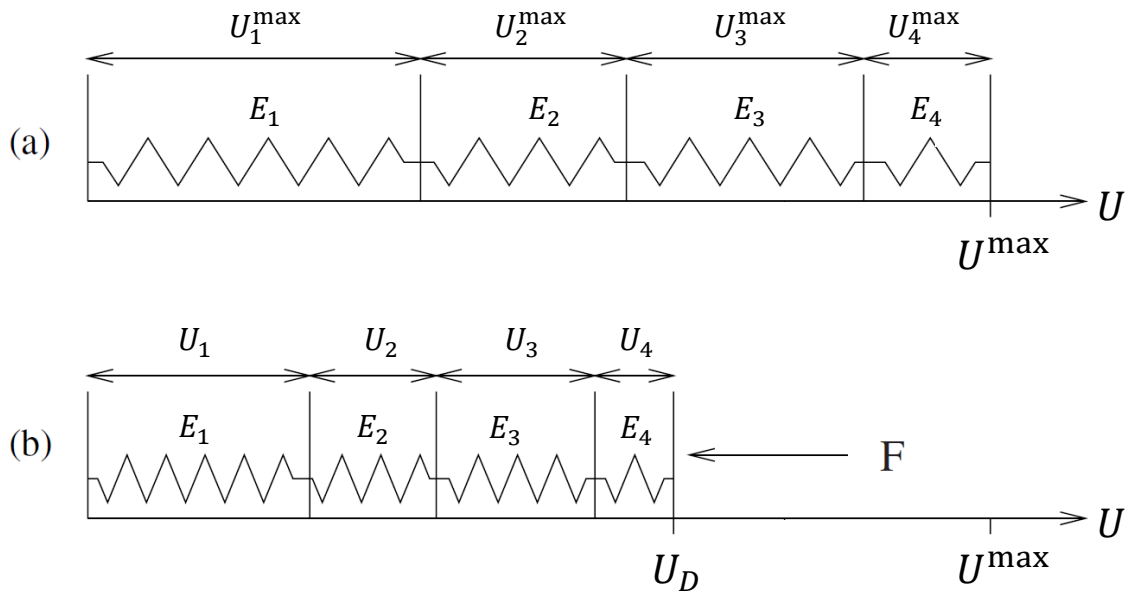


Figure 1 The physical spring analogy of the elastic model of Buttazzo et al., reproduced from [11, Figure 9.27] with modifications to the labels. **(a)** shows the uncompressed task set, while **(b)** illustrates the application of elastic compression to achieve schedulability.

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

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

where

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

and

$$E_{\text{SUM}} = \sum_{\tau_i \in \Gamma_{\text{VARIABLE}}} E_i \tag{5}$$

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

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

denotes the sum of the $U_i^{\min}$ parameters[5] 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.

[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 $\left(U_{\text{SUM}} - (U_D - \Delta)\right)$ 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}).

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_{VARIABLE} to Γ_{FIXED} and there are n tasks, the number of iterations is bounded from above by n. The overall running time for the algorithm is therefore $\mathcal{O}(n^2)$.

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

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

```
1  U_SUM^min ← Σ_{τ_i} C_i/T_i^max
2  if U_D < U_SUM^min then return INFEASIBLE
3  forall τ_i ∈ Γ do  U_i ← C_i/T_i^max
4  do
5      U_FIXED ← 0, U_VARIABLE ← 0, E_SUM ← 0
6      forall τ_i ∈ Γ do
7          if (E_i = 0) or (T_i = T_i^max) then
8              U_FIXED ← U_FIXED + U_i
9          else
10             E_SUM ← E_SUM + E_i
11             U_VARIABLE ← U_VARIABLE + U_i
12         end
13     end
14     OK ← TRUE
15     forall τ_i ∈ Γ do
16         if E_i > 0 or T_i < T_i^max then
17             U_i ← U_i^max − (U_variable − U_D + U_FIXED)E_i/E_SUM
18             T_i ← C_i/U_i
19             if T_i > T_i^max then
20                 T_i ← T_i^max
21                 OK ← FALSE
22             end
23         end
24     end
25 while OK = FALSE
26 return FEASIBLE
```

The same algorithm was also repurposed in [12] for admission control – i.e., for determining whether a new task seeking to join an already-executing system could be admitted without 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:

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

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 $\quad U_{\text{SUM}} = U_{\text{SUM}} + U_i^{\max}$
4 $\quad E_{\text{SUM}} = E_{\text{SUM}} + E_i$
5 **end**
6 **forall** $\tau_i \in \Gamma$ **do**
7 $\quad$ **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 $\quad\quad$ ▷ Task τ_i is no longer compressible – it's in Γ_{FIXED}
9 $\quad\quad U_i \leftarrow U_i^{\min}$ ▷ Since $\tau_i \in \Gamma_{\text{FIXED}}$
10 $\quad\quad \Delta \leftarrow \Delta + U_i^{\min}$ ▷ This additional amount of utilization is allocated to tasks in Γ_{FIXED}
11 $\quad\quad$ **if** $(\Delta > U_D)$ **then return** INFEASIBLE ▷ Cannot accommodate the minimum requirements
12 $\quad\quad U_{\text{SUM}} \leftarrow U_{\text{SUM}} - U_i^{\max}$ ▷ Since τ_i is removed from Γ_{VARIABLE}
13 $\quad\quad E_{\text{SUM}} = E_{\text{SUM}} - E_i$ ▷ As above – since τ_i is removed from Γ_{VARIABLE}
14 $\quad$ **else**
15 $\quad\quad$ ▷ Remaining tasks are all compressible (i.e., in Γ_{VARIABLE})
16 $\quad\quad U_i \leftarrow U_i^{\max} - \frac{U_{\text{SUM}} - (U_D - \Delta)}{E_{\text{SUM}}} \times E_i$ ▷ As per Equation 3
17 $\quad$ **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 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 [26], we did not evaluate the algorithm's performance for realistic task sets. In Section 3, we perform this evaluation and extend the algorithm to fluid scheduling.

2.2 Scheduling Without a Utilization Bound

In addition to fluid scheduling, in [22], Orr and Baruah also developed elastic models for multiprocessor partitioned EDF and global EDF scheduling. They later applied elastic scheduling to partitioned RM in a journal extension [21]. All of these scheduling policies have feasibility tests that are 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 Equation 2. 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:

$$U_i(\lambda) \stackrel{\text{def}}{=} \max\left(U_i^{\max} - \lambda E_i, U_i^{\min}\right) \tag{8}$$

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:

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

which is equal to the value ϕ_i in Equation 7. 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:

$$\lambda^{\max} \stackrel{\text{def}}{=} \max_{\tau_i}\left(\lambda_i^{\max}\right) \tag{9}$$

The problem of elastic scheduling under the model of Buttazzo et al. [12, 13] 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 and RM, global EDF and RM, 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 [22, 21]. In this paper, we explore and propose alternatives to this idea for partitioned and global EDF and global RM scheduling.

2.2.1 Partitioned EDF

Under partitioned earliest deadline first (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 is 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 [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 $\left(\left\lfloor \frac{\lambda^{\max}}{\epsilon} \right\rfloor + 1\right)$ – the overall complexity is $\Theta\left(\frac{\lambda^{\max}}{\epsilon} \cdot (n \log n + n \cdot m)\right)$.

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:

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

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:

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

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 $U_{\text{SUM}}^{\max}$ 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 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 Buttazo 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 running[6] 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 `O0`; 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 $U_{\text{SUM}}^{\max}$ selected at random uniformly from $(1.0, 2.0]$ and a total minimum utilization $U_{\text{SUM}}^{\min}$ selected at random uniformly from $(0.0, 1.0]$.
3. We apply the Dirichlet Rescale (DRS) algorithm [16] 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 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 $U_{\text{SUM}}^{\min}$ 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 $U_i^{\max}$.
5. Each task τ_i is assigned an elasticity E_i at random, selected uniformly from the range $(0, 1]$.

3.2.3 Execution Time of Utilization Compression for Schedulability

We execute our implementations of Algorithms 1 and 2 to 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 the original procedure in Algorithm 1, initialization involves computing the $U_{\text{SUM}}^{\min}$ value and checking whether it exceeds U_D, while compression is the **do ... while** loop in the algorithm. For our improved algorithm in Algorithm 2, compression is the **forall** loop that iterates over tasks in order of their ϕ_i parameters. The dominant contribution to the 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.

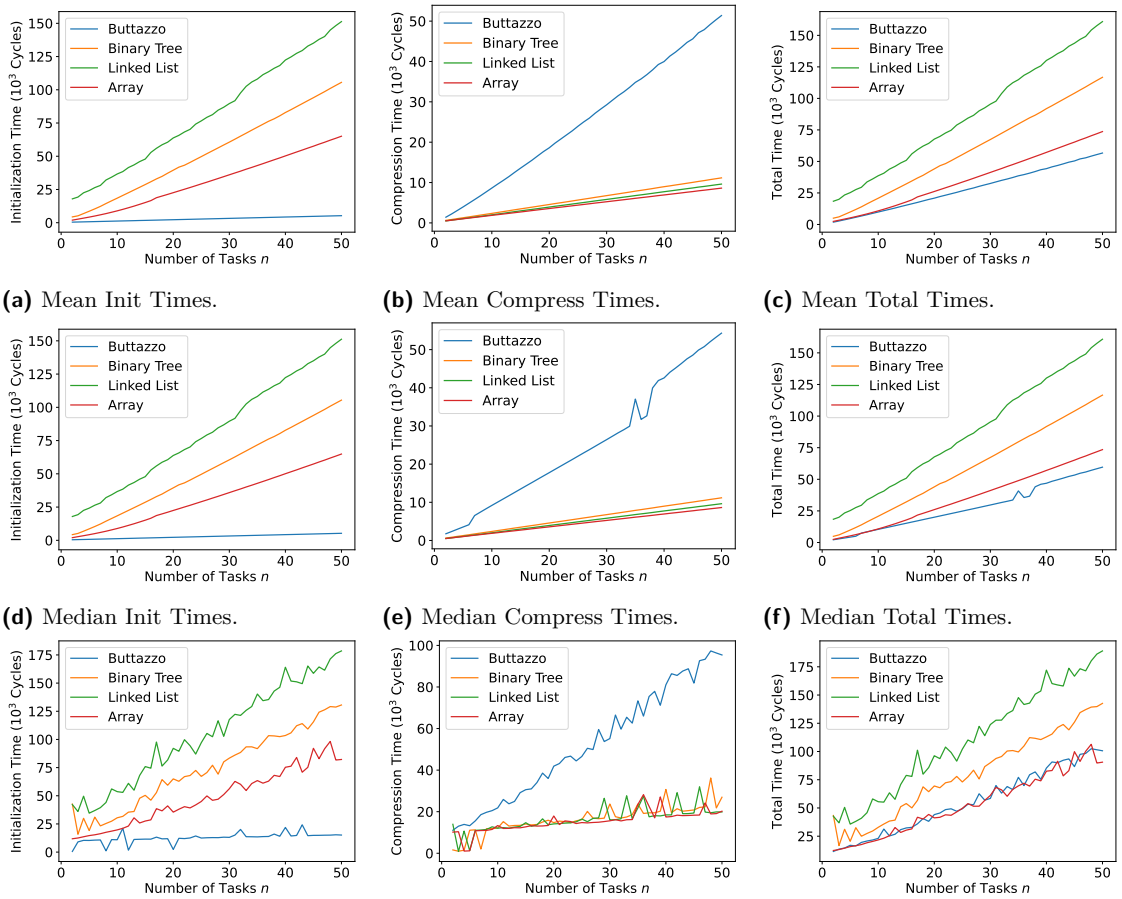
The mean, median, and maximum times for the 10 000 sets of tasks generated for each size n from 2–50 are reported in Figure 2. As expected, the time to initialize the algorithm of Buttazzo et al. is much faster than our improved algorithm, which has to sort tasks by their ϕ_i values. Of the three implementations 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 the algorithm of Buttazzo et al. 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.[7] This makes sense; while all three data structures enable linear time traversal, the array is the simplest to iterate (in fixed-size strides) and has the best data locality; the linked list is still simple, but requires following pointers between nodes, and does not have as good locality; and the binary tree requires even more complex pointer chasing.

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

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. In this case, when only compression needs to occur, associated overheads are summarized in Table 1. The worst execution times that we observed for the array-based implementation of our algorithm were 3.45× faster than those of the algorithm of Buttazzo et al. when just compressing tasks. Moreover, as we will demonstrate, there is a clear advantage to using our 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.



(a) Mean Init Times. **(b)** Mean Compress Times. **(c)** Mean Total Times.

(d) Median Init Times. **(e)** Median Compress Times. **(f)** Median Total Times.

(g) Max Init Times. **(h)** Max Compress Times. **(i)** Max Total Times.

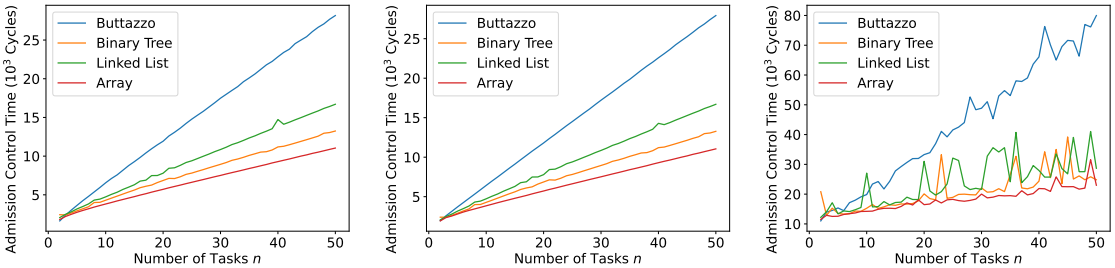
Figure 2 Performance scaling with number of tasks.

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.

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

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.



(a) Mean Admission Times. **(b)** Median Admission Times. **(c)** Max Admission Times.

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

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× faster than original algorithm in the worst case.

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.

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

3.2.5 Implications for Online Adaptation

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

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

Until now, we have considered only utilization compression in a general sense; thus, our 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.

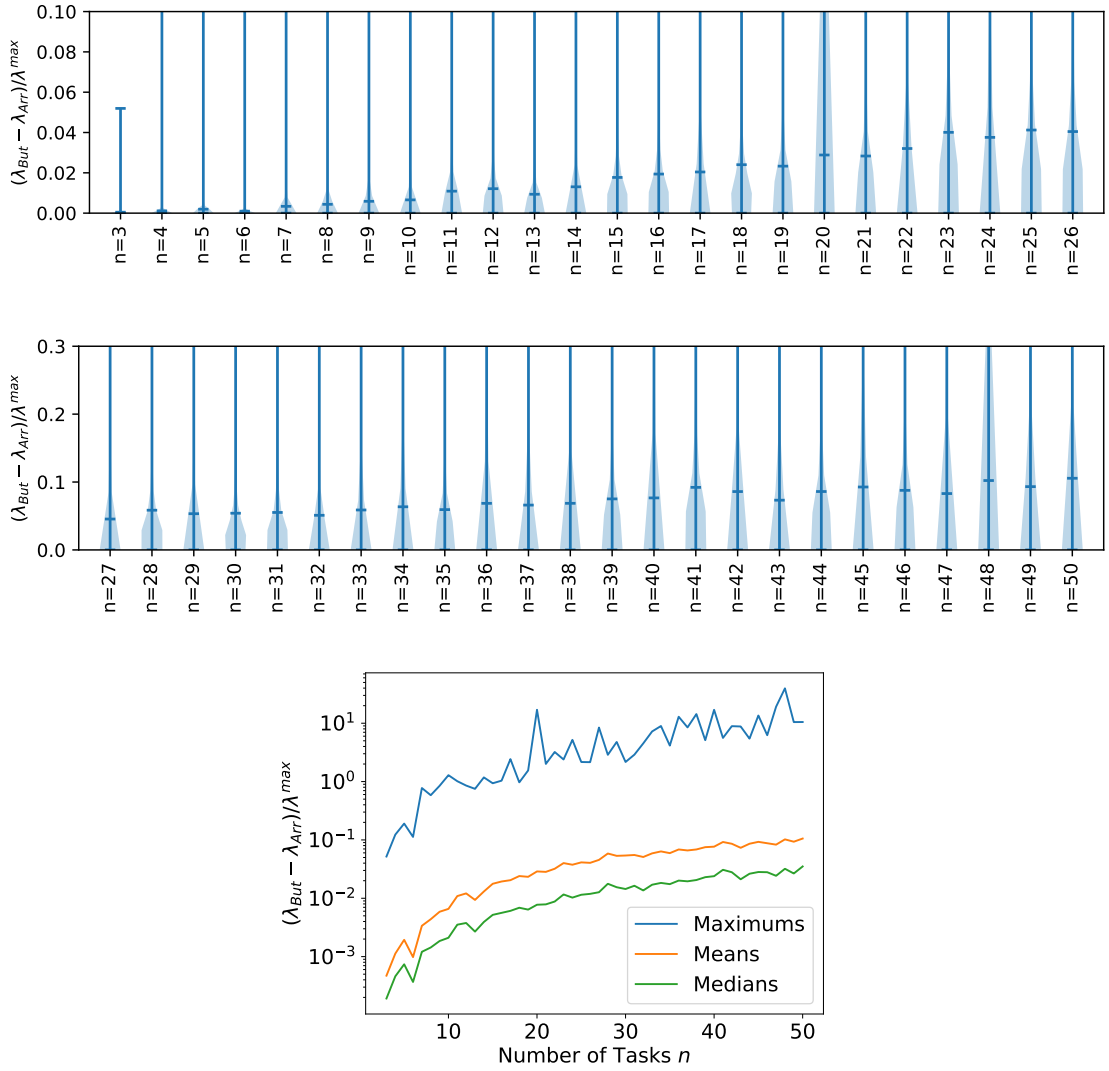


Figure 4 Difference between λ_{But} and λ_{Arr}, normalized by $\lambda^{\max}$.

Figures 4 (Top) and (Middle) show the distribution of differences $\frac{(\lambda_{\text{But}} - \lambda_{\text{Arr}})}{\lambda^{\max}}$ 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 $\lambda^{\max}$ taken from the original task set without overheads. Figure 4 (Bottom) shows the mean, median, and maximum of the normalized values 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.

[8] We ignore the "angle-synchronous" entry and renormalize probabilities over the remaining distribution.

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× more than the original $\lambda^{\max}$ 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 $U_{\text{SUM}}^{\max}$ or $U_{\text{SUM}}^{\min}$ 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 $U_{\text{SUM}}^{\min}$, $U_{\text{SUM}}^{\max}$, and the absolute distance between them, $\left(U_{\text{SUM}}^{\max} - U_{\text{SUM}}^{\min}\right)$, 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.

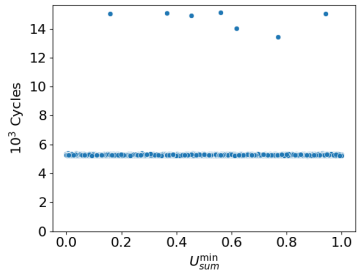
4 Partitioned EDF

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.

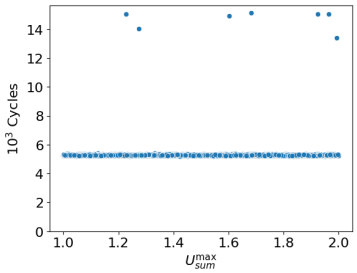
4.1 Binary Search

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 $\Theta\left(\log \frac{\lambda^{\max}}{\epsilon}\right)$, as outlined in Algorithm 3. Thus, total time complexity is reduced to

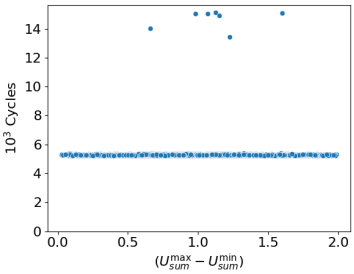
$$\boldsymbol{\Theta}\left((n \log n + n \cdot m) \cdot \log\left(\frac{\boldsymbol{\lambda^{\max}}}{\boldsymbol{\epsilon}}\right)\right) \tag{12}$$



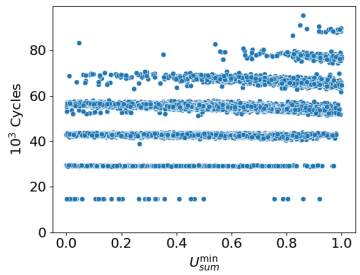
(a) Buttazzo Init.



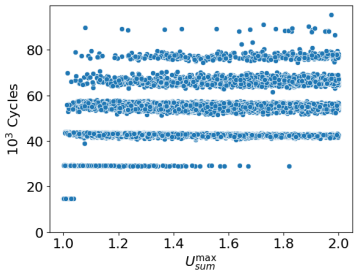
(b) Buttazzo Init.



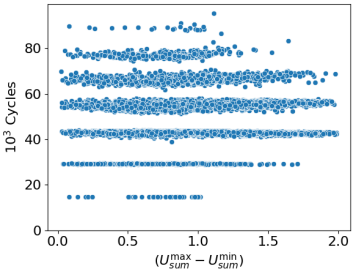
(c) Buttazzo Init.



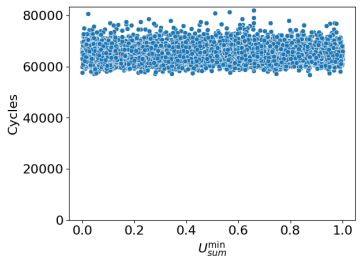
(d) Buttazzo Compress.



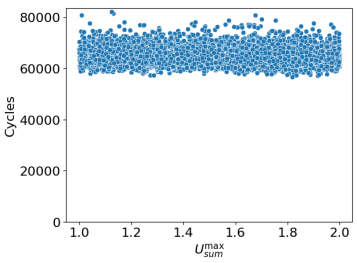
(e) Buttazzo Compress.



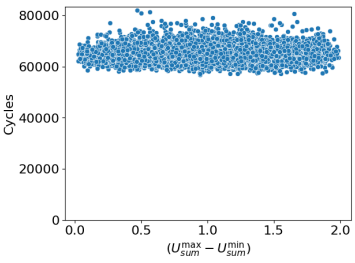
(f) Buttazzo Compress.



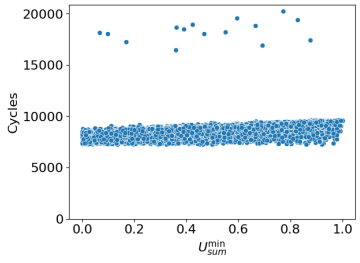
(g) Improved Init: Array.



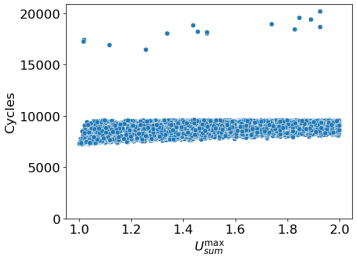
(h) Improved Init: Array.



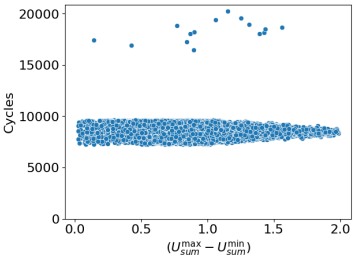
(i) Improved Init: Array.



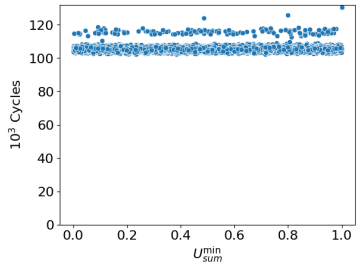
(j) Improved Compress: Array.



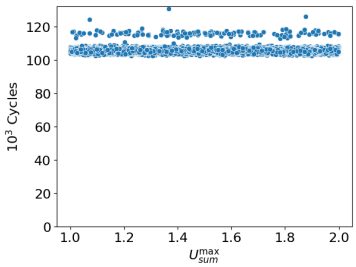
(k) Improved Compress: Array.



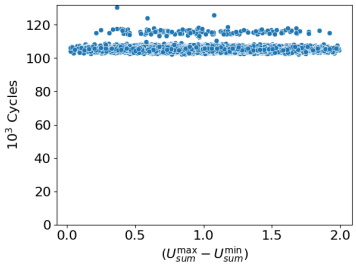
(l) Improved Compress: Array.



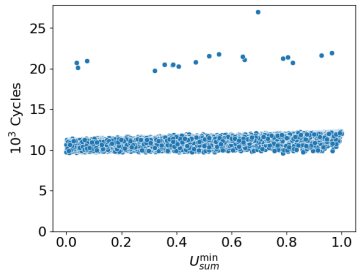
(m) Improved Init: Tree.



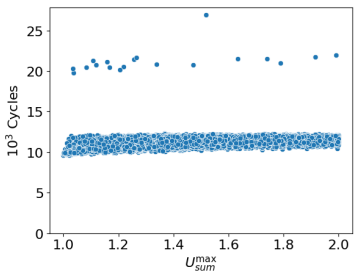
(n) Improved Init: Tree.



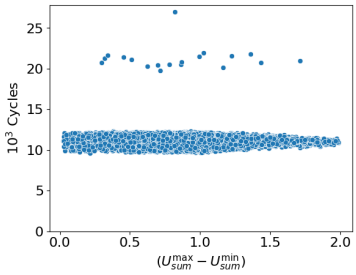
(o) Improved Init: Tree.



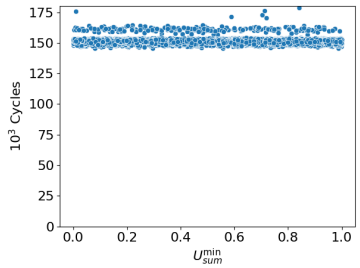
(p) Improved Compress: Tree.



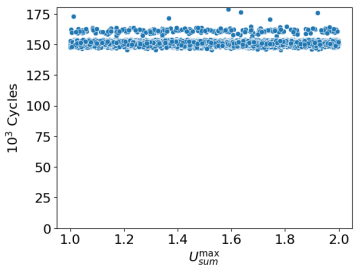
(q) Improved Compress: Tree.



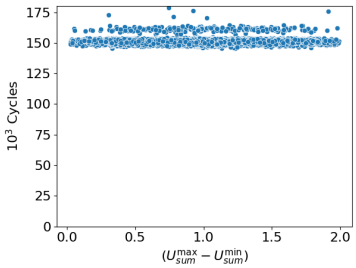
(r) Improved Compress: Tree.



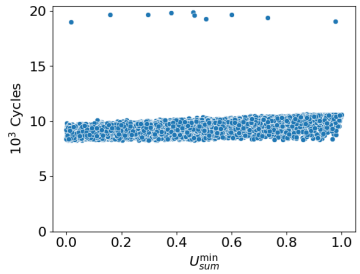
(s) Improved Init: List.



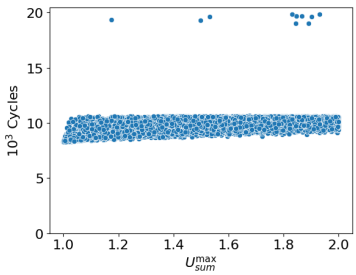
(t) Improved Init: List.



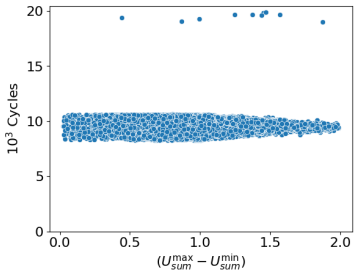
(u) Improved Init: List.



(v) Improved Compress: List.



(w) Improved Compress: List.



(x) Improved Compress: List.

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

$\lambda^{\max} \leftarrow 0$
forall $\tau_i \in \Gamma$ **do**
  $\lambda_i^{\max} \leftarrow \frac{U_i^{\max} - U_i^{\min}}{E_i}$
  $\lambda^{\max} \leftarrow \max\left(\lambda^{\max}, \lambda_i^{\max}\right)$
end
if $\Gamma(0)$ *is schedulable on* m *cores* **then return** *0*
if $\Gamma(\lambda^{\max})$ *is* ***not*** *schedulable on* m *cores* **then return** INFEASIBLE
$\lambda_{\text{HI}} \leftarrow \lambda^{\max}$
$\lambda_{\text{LO}} \leftarrow 0$
do
  $\lambda \leftarrow \left(\lambda_{\text{HI}} - \lambda_{\text{LO}}\right)/2$
  if $\Gamma(\lambda)$ *is schedulable on* m *cores* **then** $\lambda_{\text{HI}} \leftarrow \lambda$
  else $\lambda_{\text{LO}} \leftarrow \lambda$
while $\lambda_{\text{HI}} - \lambda_{\text{LO}} > \epsilon$
return λ_{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 $\lambda_{\max}$) 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.

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 $\{\Gamma_1, \ldots, \Gamma_m\}$ 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,

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

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. ◀

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 $|\lambda_{\text{LIN}} - \lambda_{\text{BIN}}| < \epsilon$. ◀

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 $\boldsymbol{\mathcal{O}(n \log n)}$ 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 $U_{\text{SUM}}^{\max}$ 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 $U_{\text{SUM}}^{\max}$ 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 $U_{\text{SUM}}^{\max}$ and *(ii)* each value $U_i^{\max}$ does not exceed the constraint imposed by the chosen value of α.
- Individual minimum utilizations $U_i^{\min}$ are assigned at random, selected uniformly from the range $(0, U_i^{\max}]$.
- Elastic coefficients E_i are assigned at random, selected uniformly from the range $(1, 5]$.

4.3.3 Linear versus Binary Search

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

Figure 6 shows – for each combination of m, n, α, and u – the median and maximum speedup achieved by binary search over linear search. As before, task sets not requiring compression ($\lambda = 0$) are excluded, as are those deemed infeasible under any amount of compression. 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 51×, while the maximum speedup observed was 86×.

Figure 7 shows the distribution of differences $\frac{(\lambda_{\text{BIN}} - \lambda_{\text{LIN}})}{\epsilon}$ between the value λ_{BIN} achieved by binary search and λ_{LIN} returned by linear search, normalized by ϵ. We again exclude trivial or infeasible task sets. Where outliers extend beyond the plotted boundaries, the y-axis labels denote the maximum value. We observe that, although the values λ_{BIN} and λ_{LIN} typically do not differ 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_{BIN} exceeds λ_{LIN} by more than $200\times\epsilon$. Generally, we see that outliers occur where λ_{BIN} 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 $0.262\times\epsilon$ for every considered combination. Therefore, given the significant speedups gained, binary search is an attractive approach.

4.3.4 Fast Compression

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

Results are shown in Figure 8, which alternates between two types of plots. The first type shows the percentage of task sets identified as feasible by BIN and UTIL. The second type compares the median and maximum speedup gained by UTIL over BIN to the mean λ values achieved by each approach. As in [22], 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.

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

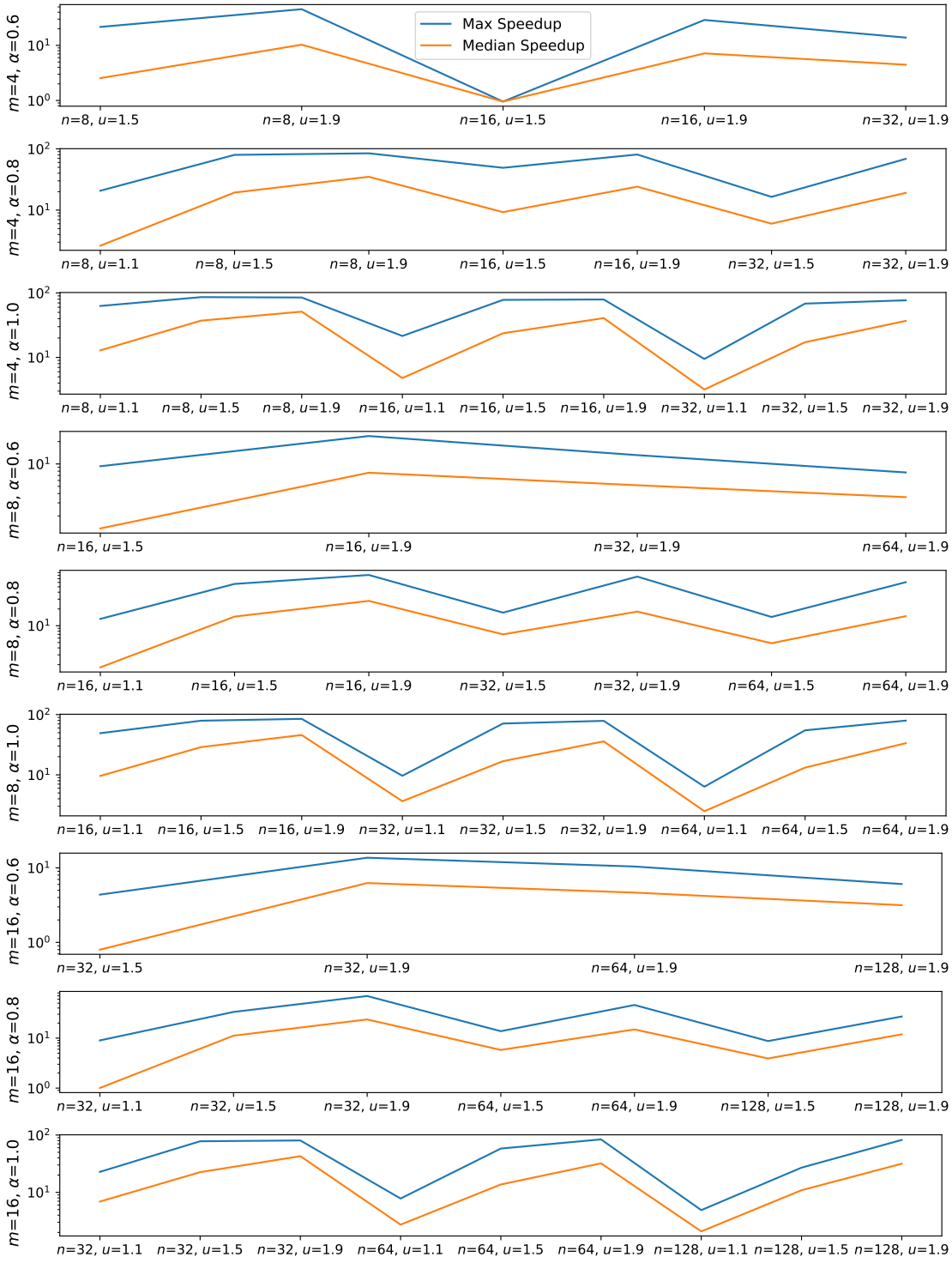


Figure 6 Speedups achieved by BIN over ITER.

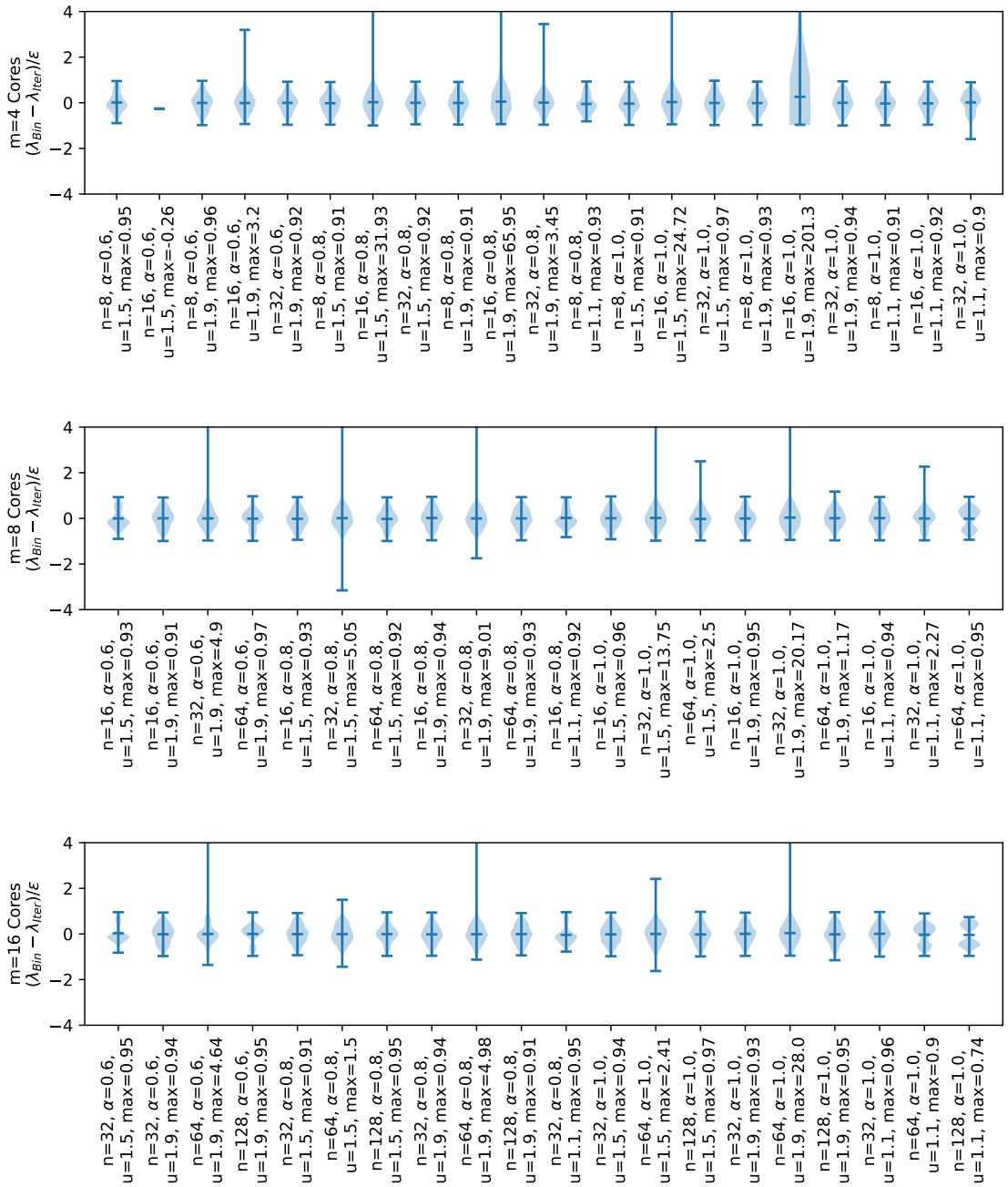


Figure 7 Difference between λ_{BIN} and λ_{LIN}, normalized by ϵ.

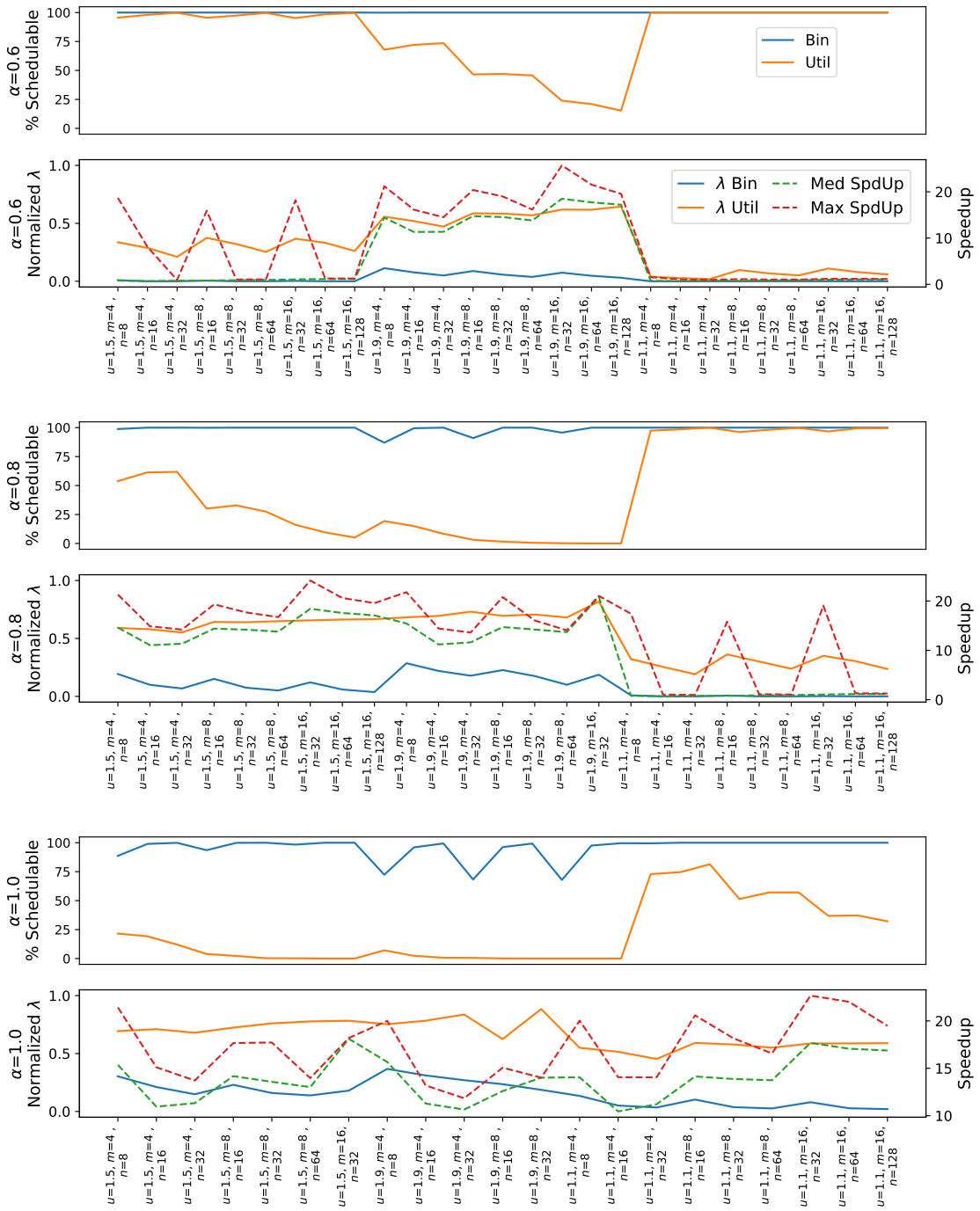


Figure 8 Speed and Schedulability Tradeoffs Between BIN and UTIL.

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

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

▶ **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.,

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

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

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

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 $\sum_{\tau_i \in \Gamma, i \neq j} U_i(\lambda) + mU_j(\lambda) = m$ 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 Γ in non-decreasing order of ϕ_i (see Equation 7)
4 $\lambda \leftarrow \lambda^{\max}$
5 **forall** $\tau_i \in \Gamma$ **do**
6 $\quad \tau_j^* \leftarrow (U_j^{\max} : mU_i^{\max}, U_j^{\min} : mU_i^{\min}, E_j : mE_i)$
7 $\quad \Gamma^* \leftarrow \Gamma$, Remove τ_i and insert τ_j^* into Γ^*
8 $\quad \triangleright$ `Invoke Algorithm 2`
9 $\quad \lambda^* \leftarrow$ ELASTIC_COMPRESSION(Γ^*,m)
10 $\quad$ **if** $\frac{U_j^*}{m}$ *is the maximum compressed utilization* **and** $\lambda^* < \lambda$ **then** $\lambda \leftarrow \lambda^*$
11 **end**
12 **return** λ

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 $\lambda^{\max}$ 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 $\boldsymbol{\mathcal{O}(n^2)}$.

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:

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

▶ **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.,

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

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:

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

Equivalently,

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

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 implementatio 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 Γ(0) is schedulable on m cores then return 0
2  if Γ(λ^max) is not schedulable on m cores then return INFEASIBLE
3  Sort Γ in non-decreasing order of φ_i (see Equation 7)
4  λ ← λ^max
5  forall τ_i ∈ Γ do
6      τ_j^* ← (U_j^max : (m/2)U_i^max, U_j^min : (m/2)U_i^min, E_j : (m/2)E_i)
7      Γ* ← Γ, Remove τ_i and insert τ_j^* into Γ*
8      ▷ Invoke Algorithm 2
9      λ* ← ELASTIC_COMPRESSION(Γ*, m/2)
10     if 2U_j^*/m is the maximum compressed utilization and λ* < λ then λ ← λ*
11 end
12 return λ
```

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×, while the maximum speedup observed was 60×.

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×, while the maximum speedup observed was 51.8×.

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

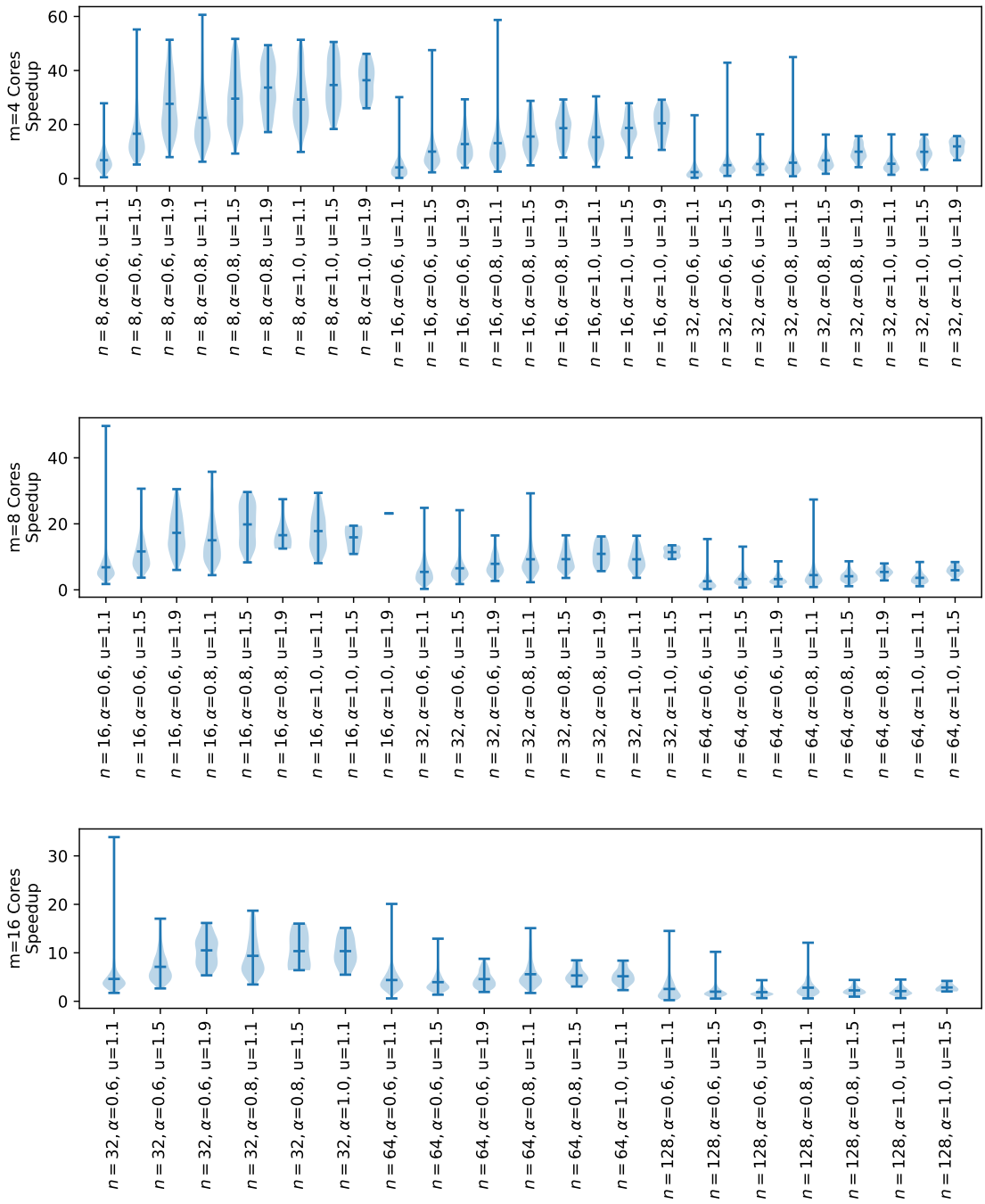


Figure 9 Speedups achieved by Algorithm 4 over iterative search.

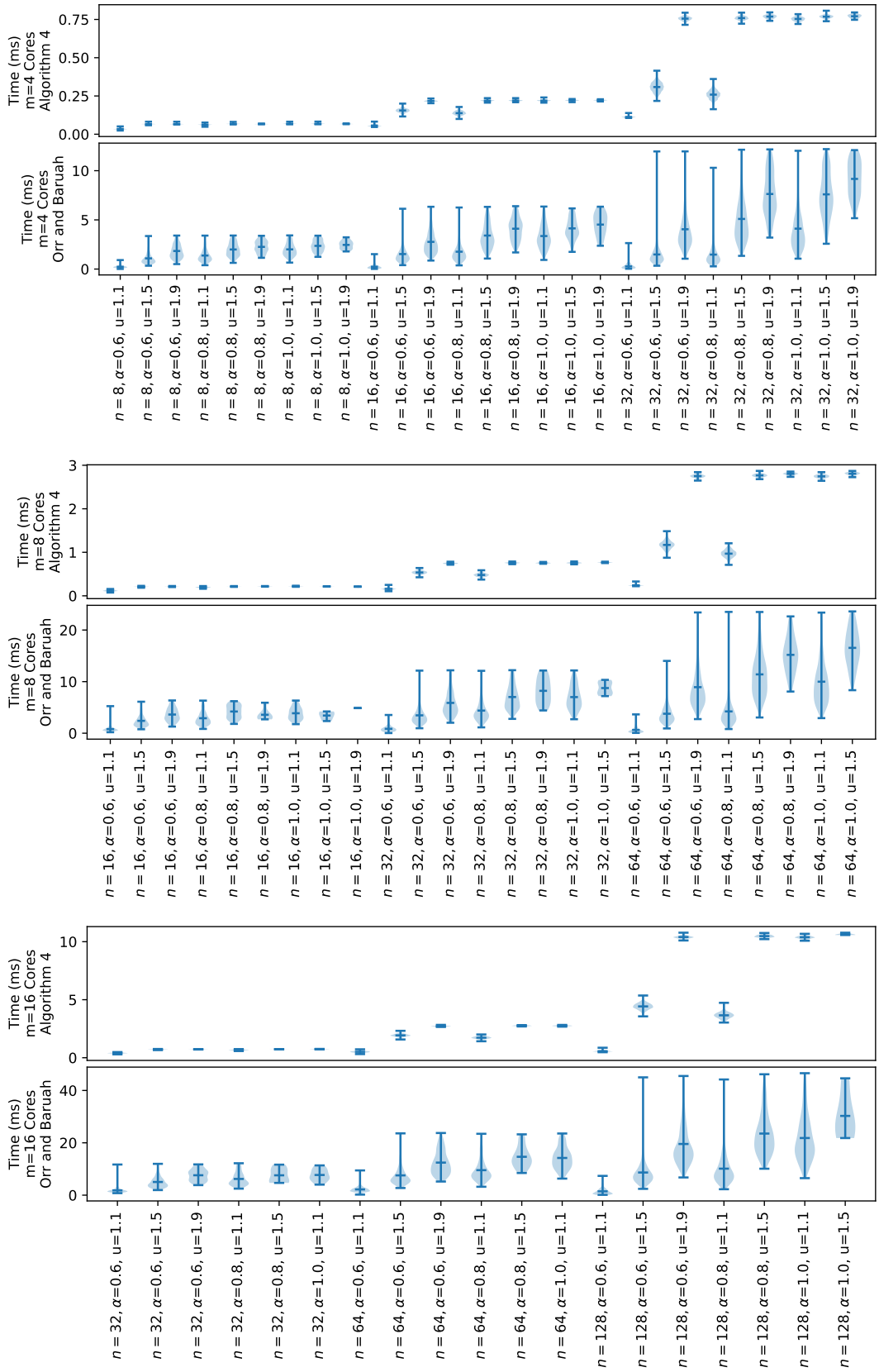


Figure 10 Execution times for global EDF compression algorithms.

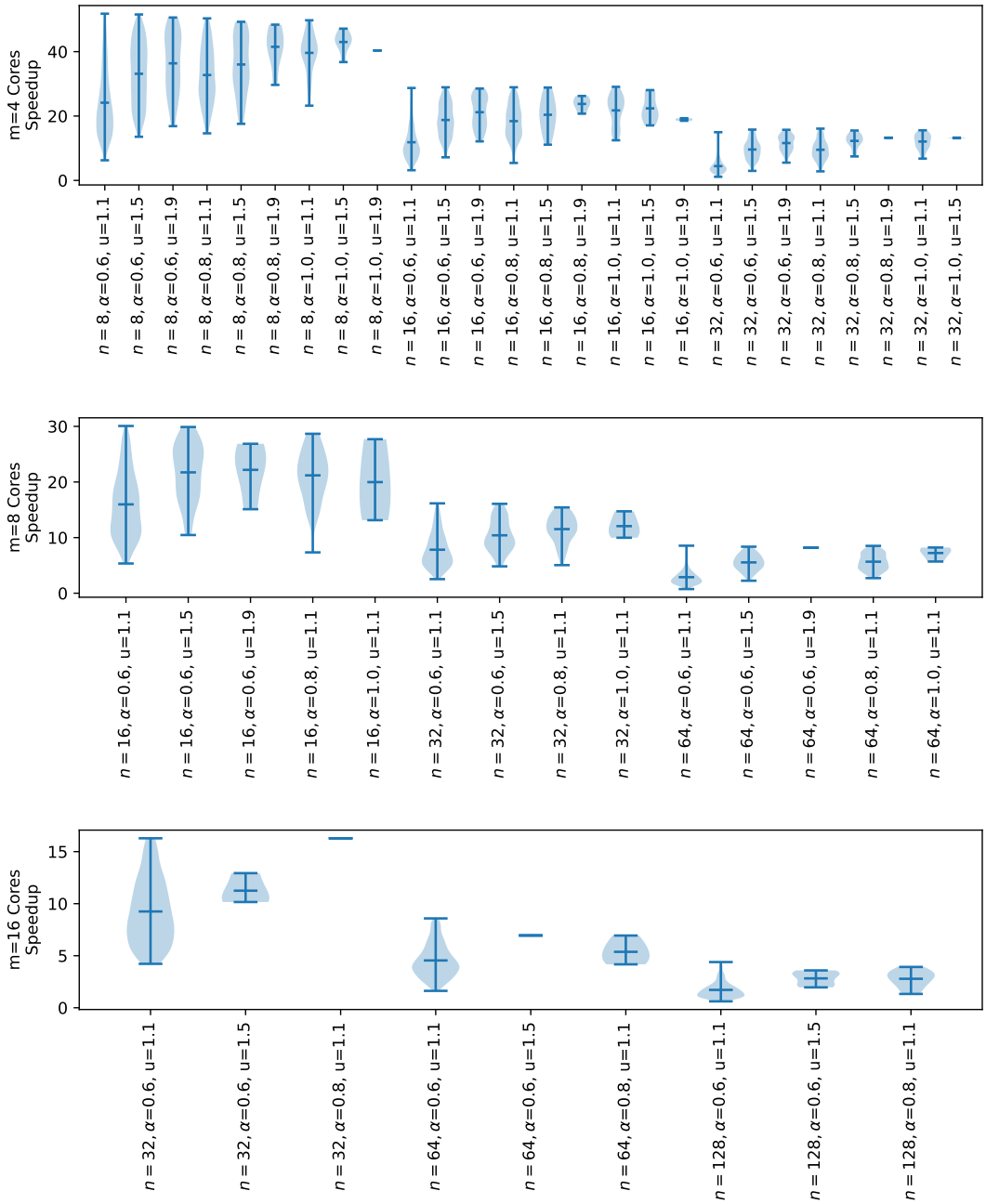


Figure 11 Speedups achieved by Algorithm 5 over iterative search.

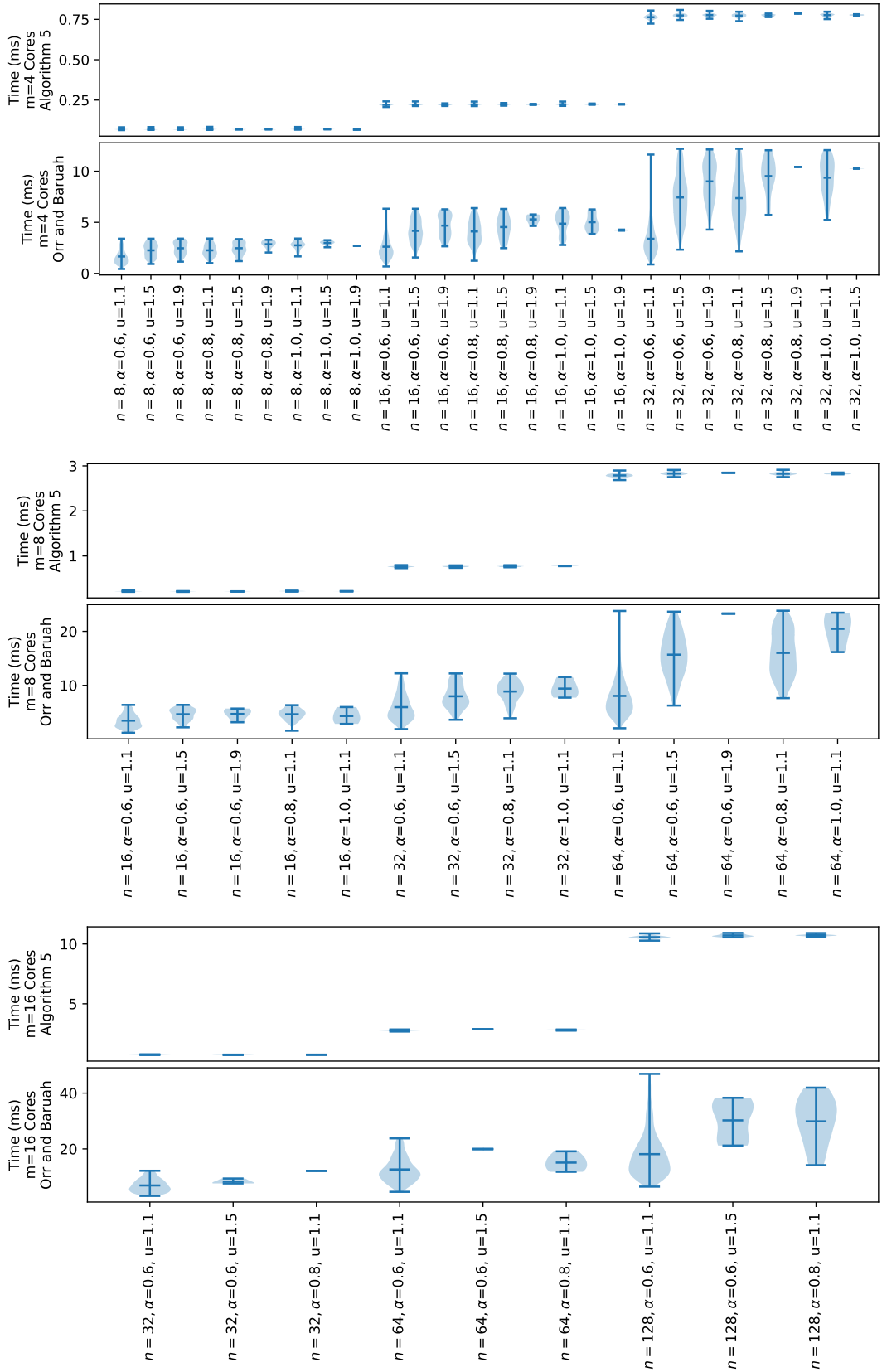


Figure 12 Execution times for global RM compression algorithms.

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, June 1996. doi:10.1007/BF01940883.

2 Sanjoy K. Baruah. Partitioned EDF scheduling: a closer look. *Real Time Syst.*, 49(6):715–729, November 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. 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, November 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, March 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, September 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, October 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, October 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.