Optimal Priority Assignment for Synchronous Harmonic Tasks With Dynamic Self-Suspension

Mario Günzel*, Marion Sudvarg‡, Max Deppert§, Ao Li‡, Ning Zhang‡, and Jian-Jia Chen*
*TU Dortmund University, ‡Washington University in St. Louis, §Kiel University
*{mario.guenzel,jian-jia.chen}@tu-dortmund.de, ‡{msudvarg,ao,zhang.ning}@wustl.edu, §made@cs.uni-kiel.de

***Abstract*—Self-suspension behavior happens when a job has to wait for some activity to complete and results in substantial schedulability degradation in real-time systems. Despite extensive studies for self-suspending real-time task systems, the state of the art has barely addressed the optimality of the scheduling algorithms, especially for tasks with dynamic self-suspension.**

In this paper, we explore optimal priority assignment for periodic real-time tasks with dynamic self-suspension under Task-level Fixed-Priority (T-FP) scheduling. To that end, we provide exact schedulability tests for frame-based and synchronous harmonic tasks. We show that the Suspension-Aware Deadline-Monotonic (SADM) priority assignment is an optimal fixed-priority scheduler for many scenarios. Further, for cases where SADM is not optimal, we adopt Audsley's Optimal Priority Assignment (OPA) approach to derive an optimal fixed-priority assignment. Evaluation results show that the exact tests outperform state-of-the-art schedulability tests from the literature, and that optimal priority assignments significantly improve schedulability over classical priority assignments.

I. Introduction

In real-time systems, some tasks may suspend themselves during execution. Such "self-suspension" typically happens when a job has to wait for some activity to complete, e.g., offloading its computation to a hardware accelerator [17], [27], [56] or waiting for a shared resource [16]. The 2019 review paper by Chen et al. [22] summarizes the existing self-suspension task models; provides general methodologies; explains the misconceptions in the literature, their consequences, and potential solutions to fix those flaws; and presents a summary of the computational complexity classes of different self-suspension task models and systems. Self-suspension can induce several non-trivial phenomena. Chen et al. conclude that "*[...] key insights underpinning the analysis of non-self-suspending tasks no longer hold*" [22], in which counterexamples for multiple analyses before 2014 were found.

Two self-suspension models, *dynamic* and *segmented*, are primarily studied in the literature [22], [23].[1] While in the segmented self-suspension model a sequence of computation segments separated by suspension intervals is specified, this work focuses on the *dynamic self-suspension* task model, where a task τ_i is assigned a maximum total self-suspension time parameter S_i. Specifically, under the dynamic self-suspension model, a task τ_i may suspend itself at any moment before it finishes and infinitely often as long as its total suspension time does not exceed S_i.

There are two correlated problems studied for self-suspension models: how to schedule the tasks (i.e., the *scheduler design problem*) and whether all jobs meet their deadlines under a scheduling algorithm (i.e., the *schedulability test problem*). For the scheduler design problem, the objective is to design an optimal scheduling algorithm that guarantees a feasible schedule whenever it exists. For the schedulability test problem, the objective is to derive exact (i.e., necessary and sufficient) schedulability tests.

In this paper, we consider periodic real-time tasks with dynamic self-suspension. This model can be found in many real-world applications. For example, many real-time autonomous automotive and aerial systems include real-time perception pipelines with tasks that self-suspend, e.g., while offloading computation to GPUs or other accelerators [2], [55]. A single job of a task might suspend multiple times in unpredictable patterns; we provide an example in a case study of Autoware's default LiDAR pipeline [40] in Section VII-B. Moreover, the recent industrial survey from Akesson et al. [1], which examines industrial practice in the field of real-time systems, shows that 82% of the investigated systems have periodic task activations.

For tasks with self-suspension, the literature has focused on sporadic task systems, where each task has a minimum inter-arrival time between two consecutive job releases. There are only four dedicated results for periodic task models.[2] Yalcinkaya et al. [72] analyze tasks with segmented self-suspension under non-preemptive scheduling. Günzel et al. [36, Section V] examine the schedulability under preemptive Earliest-Deadline-First (EDF) scheduling with dynamic self-suspension. Furthermore, Lin et al. [50] achieve anomaly-free online schedules for *periodic* tasks with segmented self-suspension. Liu et al. [52] consider synchronously released, periodic real-time tasks with harmonic periods under preemptive Rate-Monotonic (RM) scheduling and show that their analysis dominates the well-known (but pessimistic) suspension-oblivious schedulability test for uniprocessor systems. All these results show improvements by exploiting periodicity. However, to the best of our knowledge, there is not a single result on exact schedulability tests or optimal priority assignments of dynamic self-suspending tasks under preemptive Task-level Fixed-Priority (T-FP) scheduling.

[1] Some results [8], [13], [26], [41], [43], [46] have been disproved (c.f. the review by Chen et al. [22] and Günzel and Chen [32], [33]).

[2] Although Casini et al. [17] also focus only on periodic tasks in their paper, their analysis is based on the jitter-based analysis developed for sporadic tasks.

Contributions: This paper explores optimal preemptive T-FP scheduling algorithms to deal with dynamic self-suspension. To that end, we focus on periodic real-time task systems, to step away from the obstacles of sporadic arrivals of real-time jobs. Specifically, we make the following contributions:

- We show that the dynamics of self-suspension can be easily handled in polynomial time by Suspension-Aware Deadline-Monotonic (SADM) priority assignment when the periodic tasks have the same period and release their first jobs synchronously (namely, *frame-based* real-time tasks) in Section IV. This holds for both implicit-deadline and constrained-deadline task systems.
- For synchronous periodic tasks with harmonic periods and dynamic self-suspension, in Section V, we show that SADM is no longer optimal even for implicit-deadline tasks and develop a polynomial-time algorithm based on Audsley's approach [7].
- We note that our positive results for frame-based tasks and harmonic tasks are based on exact schedulability tests dedicated for these scenarios. Any extension should be done with care. In Section VI, we discuss the limitations for extensions to sporadic tasks, fixed numbers of self-suspension intervals, and segmented suspension models.
- The evaluation in Section VII shows that the proposed exact schedulability tests significantly outperform state-of-the-art schedulability tests. Furthermore, the optimal priority assignments lead to a substantial increase in schedulability. We demonstrate this using a case study with workloads from Autoware's LiDAR processing pipeline that dynamically self-suspend to invoke the GPU, and by considering randomly-generated synthetic task sets.

II. Related Work

For ordinary real-time tasks without self-suspension, the Rate-Monotonic (RM) priority assignment is known to be an optimal preemptive T-FP scheduler for implicit-deadline sporadic tasks [53], synchronous periodic tasks [53], and asynchronous periodic harmonic task systems [48, Theorem 3.2]. Furthermore, the Deadline-Monotonic (DM) priority assignment is an optimal preemptive T-FP scheduler for constrained-deadline sporadic tasks [48] and synchronous periodic tasks [48]. Ausley's OPA approach [6], [7] can be used to derive an optimal T-FP assignment when the exact schedulability test is OPA-compatible [25].

Given a priority assignment, the schedulability test problem for ordinary real-time tasks has been widely studied for different recurrent scenarios. Ekberg and Yi [29] provide a summary of the state of the art regarding the computational complexity of the schedulability test problem for uniprocessor systems. Specifically, due to the critical instant theorem by Liu and Layland [53] and the busy interval concept by Lehoczky [47], the schedulability analysis for a sporadic real-time task system is equivalent to its corresponding periodic task system with synchronous releases. It is known that the schedulability test problem for a given T-FP assignment can be solved in polynomial time when the tasks have harmonic periods and implicit [45] or constrained deadlines [15], [63].

Schedulability of dynamic self-suspending tasks has been examined in multiple papers [4], [14], [21], [34]–[37], [39], [51], [52] and in the book by Jane Liu [54, Page 162]. The segmented self-suspension task model specifies tasks by a sequence of computation segments separated by suspension intervals, studied in [17], [19], [20], [33], [38], [42], [50], [62], [64], [67], [72]. The generalization of self-suspension into different scenarios has been analyzed [5], [24], [70], [71].

Although self-suspending real-time task systems have been studied extensively, there are only sufficient schedulability analyses for sporadic real-time tasks with dynamic self-suspension [4], [14], [21], [34]–[37], [39], [51]. Regarding the optimality of scheduling algorithms, Chen [18] shows that there is no constant approximation bound (with respect to the resource augmentation factor) for sporadic real-time tasks with dynamic self-suspension under preemptive T-FP scheduling, compared to the optimal schedules, if the suspension time cannot be reduced by speeding up. Chen [18] also demonstrates the non-optimality of Earliest-Deadline-First (EDF), Least-Laxity-First (LLF), and Earliest-Deadline-Zero-Laxity (EDZL) scheduling algorithms and concludes that *how to design good schedulers with a constant speedup factor remains as an open problem* for sporadic real-time tasks.

Although it is widely understood that self-suspension results in substantial schedulability degradation in real-time systems, whether the scheduler design problem and the schedulability test problem under the T-FP paradigm are essentially more difficult has been only partially discussed in the literature. The computational complexity regarding schedulability tests [18], [58] and scheduling algorithms [65], [66] have been mostly limited to sporadic real-time releases, with an exception by Chen et al. [19] for scheduling frame-based real-time tasks with segmented self-suspension. As summarized by Chen et al. [22], the computational complexity for scheduling sporadic real-time tasks with dynamic self-suspension is unknown and never discussed in the literature.

Table I and Table II summarize the results in the literature as well as our findings in this paper (marked in blue) for self-suspending tasks regarding the scheduler design problem and the (exact) schedulability test problem.

III. System Model

Tasks and Jobs: Let $\mathbb{T} = \{\tau_1, \ldots, \tau_n\}$ be a set of $n \in \mathbb{Z}_{\geq 1}$ tasks. Each task τ_i releases countably many jobs $\tau_{i,j}$ for $j \in \mathbb{Z}_{\geq 1}$. We denote by $r_{i,j}$ the release time of job $\tau_{i,j}$, and assume that job releases follow the job index, i.e., $r_{i,j} < r_{i,j+1}$. In this work, we consider *periodic* tasks, where two subsequent job releases are always exactly separated by the task period $T_i \in \mathbb{R}_{>0}$, and the first job release occurs at the task phase $\phi_i \in \mathbb{R}$ (sometimes also referred to as task offset). Specifically, we write

$$\tau_i \in Per(T_i, \phi_i) \tag{1}$$

if τ_i releases jobs at times $r_{i,j} = \phi_i + (j-1) \cdot T_i$ for $j \in \mathbb{Z}_{\geq 1}$. This differs from the *sporadic* task model, where job releases

	ordinary (no suspension)			dynamic self-suspension			segmented self-suspension
Release Model	**Implicit-DL**	**Constrained-DL**	**Arbitrary-DL**	**Implicit-DL**	**Constrained-DL**	**Arbitrary-DL**	
synchronous, frame-based	any	DM	DM	SADM (poly.-time, Section IV)		no analysis or optimization known	$\mathcal{NP}$-hard in the strong sense [19]
synchronous, periodic, harmonic	RM [45]	DM [48]	OPA [7] (expo.-time exists)	OPA (poly.-time, Section V)			
synchronous periodic	RM [48], [53]	DM [48]	OPA [7] (expo.-time exists)	no analysis or optimization known			
asynchronous periodic	strongly $\mathcal{NP}$-hard [11], [29], [48][1]						
sporadic	the same as *synchronous periodic*			non-existence of bounded speedup factors for T-FP if suspension time cannot be sped up [18], unknown computational complexity [22]			

[1] Leung and Whitehead [48] show that the problem is weakly $\mathcal{NP}$-hard by a reduction from the Simultaneous Congruences Problem (SCP) problem, and it is later proved to be strongly $\mathcal{NP}$-hard by Baruah et al. [11].

Table I: The scheduler design problem under T-FP for scheduling ordinary and self-suspending tasks.

	ordinary (no suspension)			dynamic self-suspension		
Release Model	**Implicit-DL**	**Constrained-DL**	**Arbitrary-DL**	**Implicit-DL**	**Constrained-DL**	**Arbitrary-DL**
synchronous, frame-based	poly.-time			poly.-time (Section IV)		no analysis or optimization known
synchronous, periodic, harmonic	poly.-time [45]	poly.-time [15], [63]	expo.-time exists	poly.-time (Section V)		
synchronous periodic	weakly $\mathcal{NP}$-complete [28], [29] (pseudo-poly.-time exists)	weakly $\mathcal{NP}$-complete [28], [29] (pseudo-poly.-time exists)	weakly $\mathcal{NP}$-hard [28], [29] (expo.-time exists)	no exact schedulability test known, at least as difficult as the corresponding task model without self-suspension		
asynchronous periodic	weakly $\mathcal{NP}$-hard and strongly co-$\mathcal{NP}$-hard [11], [29], [48]					
sporadic	the same as *synchronous periodic*					

Table II: The (exact) schedulability test for ordinary and self-suspending tasks under T-FP.

are not predetermined but only a minimum inter-arrival time is considered. Specifically, we denote by

$$\tau_i \in Spor(T_i) \tag{2}$$

the case that $r_{i,j+1} \geq r_{i,j} + T_i$ for all $j \in \mathbb{Z}_{\geq 1}$. Furthermore, for each task τ_i, a *worst-case execution time* (WCET) C_i is defined. That is, we write

$$\tau_i \in WCET(C_i) \tag{3}$$

to denote that any job $\tau_{i,j}$ of task τ_i needs to be executed for $c_{i,j} \in (0, C_i]$ time units to complete. Each task τ_i is assigned a relative deadline D_i, and it must be ensured that each job $\tau_{i,j}$ has completed by its absolute deadline $d_{i,j} := r_{i,j} + D_i$. The range of possible relative deadlines is denoted by

$$\tau_i \in DL(I) \tag{4}$$

with $I \subseteq \mathbb{R}_{>0}$. Specifically, for periodic tasks ($\tau_i \in Per(T_i, \phi_i)$) and sporadic tasks ($\tau_i \in Spor(T_i)$), we say that τ_i has an *implicit deadline* if $D_i = T_i$, i.e.,

$$\tau_i \in DL(\{T_i\}), \tag{5}$$

and a *constrained deadline* if $D_i \leq T_i$, i.e.,

$$\tau_i \in DL((0, T_i]). \tag{6}$$

Self-Suspension: Two self-suspension models are predominantly studied in the literature [22]. For *dynamic self-suspending tasks* τ_i, an upper bound S_i on the maximum value that a task can suspend is specified. That is, we denote by

$$\tau_i \in DynSus(S_i) \tag{7}$$

the case that every job $\tau_{i,j}$ of τ_i can suspend itself as long and as often as the maximum suspension time S_i is not exceeded. For *segmented self-suspending tasks* τ_i, the suspension pattern is specified. That is, we denote by

$$\tau_i \in SegSus(C_i^1, S_i^1, C_i^2, S_i^2, \ldots, S_i^{m_i}, C_i^{m_i+1}) \tag{8}$$

the case that every job $\tau_{i,j}$ of τ_i has m_{i+1} computation segments upper bounded by $C_i^1, \ldots, C_i^{m_i+1}$, and m_i suspension intervals upper bounded by $S_i^1, \ldots, S_i^{m_i}$. For the discussion in Section VI, we also consider tasks where, instead of the whole suspension pattern, only the number of self-suspension intervals is fixed. Such tasks are denoted as

$$\tau_i \in NumSus(m_i). \tag{9}$$

Scheduling Algorithms: A scheduling algorithm $\mathcal{A}$ determines which jobs are executed. We use $s_{i,j}$ and $f_{i,j}$ to denote the start and finish times of job $\tau_{i,j}$ in a corresponding job schedule. The response time of $\tau_{i,j}$ is denoted as $R_{i,j}$. This work focuses on preemptive Task-level Fixed-Priority (T-FP) scheduling algorithms, where each task has a fixed priority level, and where higher-priority jobs can preempt the execution of lower-priority jobs at any time. Classical T-FP prioritization strategies are Deadline Monotonic (DM), where the tasks are prioritized according to relative deadlines D_i, and Rate

Monotonic (RM), where the tasks are prioritized according to period or minimum inter-arrival time T_i, i.e., task with lowest D_i or T_i has highest priority. We assume that tasks have unique priorities. To that end, ties (e.g., same relative deadline under DM scheduling) are broken arbitrarily but deterministically. We denote by $\text{HP}(i)$ be the set of tasks with higher priorities than τ_i. Our analysis focuses on a single-core platform (or on the behavior of a single core of a partitioned T-FP scheduler), and assume that preemption costs are negligible.

Task Systems under Consideration: In Section IV, we consider synchronously released frame-based tasks with dynamic self-suspension, i.e.,

$$\tau_i \in Per(T, 0) \cap WCET(C_i) \cap DynSus(S_i) \tag{10}$$

holds for all tasks $\tau_i \in \mathbb{T}$. In other words,

$$\phi_i = 0 \qquad \text{for all } \tau_i \in \mathbb{T} \tag{11}$$
$$T_i = T_j = T \qquad \text{for all } \tau_i, \tau_j \in \mathbb{T}. \tag{12}$$

Section V considers synchronous harmonic tasks with dynamic self-suspension, i.e.,

$$\tau_i \in Per(T_i \in T^H, 0) \cap WCET(C_i) \cap DynSus(S_i) \tag{13}$$

holds for all tasks $\tau_i \in \mathbb{T}$. Here, T^H denotes a set of *harmonic* periods. That is,

$$\phi_i = 0 \tag{14}$$

for all $\tau_i \in \mathbb{T}$, and

$$\frac{T_i}{T_j} \in \mathbb{Z}_{\geq 1} \qquad \text{or} \qquad \frac{T_j}{T_i} \in \mathbb{Z}_{\geq 1} \tag{15}$$

holds for all $\tau_i, \tau_j \in \mathbb{T}$. Section VI presents further task models to examine extensions and limitations of our results.

IV. Frame-Based Tasks

This section discusses optimal priority assignment under preemptive T-FP scheduling for synchronously-released frame-based tasks with dynamic self-suspension, i.e., for all $\tau_i \in \mathbb{T}$

$$\tau_i \in Per(T, 0) \cap WCET(C_i) \cap DynSus(S_i) \tag{16}$$

holds. We allow all tasks to have constrained deadlines, i.e.,

$$\tau_i \in DL((0, T]) \tag{17}$$

for all $\tau_i \in \mathbb{T}$. First, we present an exact schedulability test under any T-FP priority assignment. Afterwards, we use that test to derive the optimal priority assignment.

Usually, schedulability tests need to integrate the additional carry-in due to suspension of higher-priority tasks in the form of jitter, blocking, or additional carry-in jobs. However, for constrained-deadline frame-based tasks, the suspension behavior of higher-priority tasks does not play a role because it avoids the carry-in completely. Hence, the worst case is achieved if all higher-priority tasks do not suspend, and the task under analysis suspends its full maximum suspension time. This observation is formalized in the following lemma.

Lemma 1 (Exact test, frame-based)**.** *Let $\mathbb{T}$ be a set of frame-based tasks with dynamic self-suspension and constrained deadlines. For all $\tau_k \in \mathbb{T}$, let*

$$R_k \coloneqq C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} C_i. \tag{18}$$

The task set $\mathbb{T}$ is schedulable under preemptive T-FP scheduling if and only if $R_k \leq D_k$ for all $\tau_k \in \mathbb{T}$. In that case, R_k is the (exact) worst-case response time of τ_k.

Proof. By induction over $\tau_k \in \mathbb{T}$ in descending priority ordering, we show that: '*If all tasks in* $\text{HP}(k)$ *are schedulable, then τ_k is schedulable with worst-case response time R_k (from Equation* (18)*) if $R_k \leq D_k$, and unschedulable if $R_k > D_k$.*'

Base case: Let τ_k be the highest priority task, i.e., $\text{HP}(k) = \emptyset$. If $R_k = C_k + S_k > D_k$, then a job that executes for C_k time units and suspends for S_k time units cannot finish until the deadline, i.e., τ_k is not schedulable. However, if $R_k = C_k + S_k \leq D_k$, then R_k is an upper bound on the worst-case response time, because there are no higher-priority tasks that could interfere and there is no carry-in from previous jobs to consider. The bound is exact because it can be achieved for a job which executes C_k time units and suspends S_k time units.

Induction step: Consider a job $\tau_{k,j}$ of τ_k without carry-in from previous jobs of τ_k. The worst-case response time of that job is lower bounded by R_k, because that is achieved if $\tau_{k,j}$ executes for C_k and suspends for S_k time units, and further all higher-priority jobs $\tau_i \in \text{HP}(k)$ released at the same time execute for C_i time units without suspension. Therefore, if $R_k > D_k$, then τ_k is unschedulable. However, if $R_k \leq D_k \leq T$, then R_k is also an upper bound on the response time of $\tau_{k,j}$ because (i) C_i is the worst-case interference from the higher-priority job $\tau_{i,j}$, (ii) no subsequent jobs of higher-priority tasks can interfere with $\tau_{k,j}$, and (iii) previous jobs of higher-priority tasks finish before $r_{k,j}$ due to the induction hypothesis. Furthermore, if $R_k \leq D_k \leq T$ then the subsequent job $\tau_{k,j+1}$ cannot have carry-in from previous jobs of τ_k either, i.e., considering jobs without carry-in from previous jobs is sufficient. This proves the induction step. □

Next, we exploit the exact test to build an optimal scheduling algorithm. To that end, we observe that the schedulability condition $R_k \leq D_k$ from Lemma 1 can be reformulated as

$$C_k + \sum_{\tau_i \in \text{HP}(k)} C_i \leq D_k - S_k. \tag{19}$$

For each task $\tau_i \in \mathbb{T}$, we construct a **non-**suspending task

$$\tau'_i \in Per(T, 0) \cap WCET(C'_i) \cap DynSus(0) \cap DL(\{D'_i\}) \tag{20}$$

with execution time $C'_i = C_i$ and $D'_i = D_i - S_i$. Since the task set $\mathbb{T}' \coloneqq \{\tau'_i \mid i = 1, \ldots, n\}$ is not self-suspending, its T-FP schedulability condition is

$$\forall \tau'_k \in \mathbb{T}' \colon \quad C'_k + \sum_{\tau_i \in \text{HP}(k)} C'_i \leq D'_k, \tag{21}$$

which is equivalent to the scheduling condition of Equation (19). Therefore, a T-FP scheduling algorithm can feasibly schedule $\mathbb{T}$ if and only if it can feasibly schedule $\mathbb{T}'$.

It is well-known that the Deadline-Monotonic (DM) algorithm is an optimal T-FP scheduling algorithm for a set of periodic non-suspending preemptive real-time tasks [53] with the same phase. Therefore, assigning task priorities according to $D'_i = D_i - S_i$ (i.e., task with lowest D'_i has highest priority) is optimal. This leads us to the following theorem, which says that *suspension-aware deadline-monotonic (SADM)* priority assignment is an optimal T-FP scheduling algorithm.

Theorem 2 (Optimality of SADM, frame-based)**.** *Assigning task τ_k whose $D_k - S_k$ is smaller a higher priority (ties are broken arbitrarily) is an optimal T-FP scheduling algorithm for frame-based tasks with dynamic self-suspensions and constrained deadlines.*

Proof. To prove that SADM is optimal, we need to show that if $\mathbb{T}$ is schedulable under any other T-FP scheduling algorithm then it is also schedulable under SADM.

Assume we find a prioritization of $\mathbb{T}$, which is not SADM, such that $\mathbb{T}$ is schedulable. Without loss of generality, we assume that the tasks are indexed in priority order, i.e., $\text{HP}(k+1) = \tau_k \cup \text{HP}(k)$ for all $k \in \{1, \ldots, n-1\}$. Since the prioritization is different from SADM, there exist tasks τ_i, τ_j, with $i < j$ and $(D_i - S_i) > (D_j - S_j)$. Consequently, there must be adjacent tasks τ_k, τ_{k+1} with $k \in \{i, \ldots, j-1\}$ and

$$(D_k - S_k) > (D_{k+1} - S_{k+1}). \tag{22}$$

In the following, we show that swapping the priority of these adjacent tasks preserves schedulability of $\mathbb{T}$.

We know that τ_k and τ_{k+1} are schedulable under the given prioritization. Therefore, by using the exact test from Lemma 18, we obtain:

$$C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} C_i \leq D_k \tag{23}$$

$$C_{k+1} + S_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} C_i \leq D_{k+1} \tag{24}$$

To conclude that after swapping the priorities of τ_k and τ_{k+1} the task set $\mathbb{T}$ remains schedulable, we must show that the following two equations hold:

$$C_k + S_k + \sum_{\tau_i \in \text{HP}(k) \cup \{\tau_{k+1}\}} C_i \leq D_k \tag{25}$$

$$C_{k+1} + S_{k+1} + \sum_{\tau_i \in \text{HP}(k+1) \setminus \{\tau_k\}} C_i \leq D_{k+1} \tag{26}$$

We derive Equation (25) by reformulating Equation (24). Specifically, Equation (24) is equivalent to $C_{k+1} + C_k + \sum_{\tau_i \in \text{HP}(k)} C_i \leq D_{k+1} - S_{k+1}$. Using $D_{k+1} - S_{k+1} < D_k - S_k$, we obtain $C_{k+1} + C_k + \sum_{\tau_i \in \text{HP}(k)} C_i \leq D_k - S_k$, which is equivalent to Equation (25). Furthermore, Equation (26) is fulfilled, because $C_{k+1} + S_{k+1} + \sum_{\tau_i \in \text{HP}(k+1) \setminus \{\tau_k\}} C_i \leq C_{k+1} + S_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} C_i \overset{(24)}{\leq} D_{k+1}$.

We have shown that by swapping the priorities of tasks τ_k and τ_{k+1} with $(D_k - S_k) > (D_{k+1} - S_{k+1})$, the task set $\mathbb{T}$ remains schedulable. By swapping the priority of adjacent tasks finitely many times, we achieve the prioritization of SADM. We conclude that when $\mathbb{T}$ is schedulable under any T-FP scheduling algorithm, it is also schedulable under SADM, i.e., SADM is an optimal T-FP scheduling algorithm. □

Time Complexity: Sorting the n tasks in $\mathbb{T}$ according to $D_k - S_k$ requires $O(n \log n)$ time complexity. Testing whether Equation (19) holds for the given n tasks from the highest-priority task to the lowest-priority task can be done in $O(n)$ time complexity, amortized to $O(1)$ time per task. Therefore, validating whether the schedule of SADM is meets their deadlines for synchronous, frame-based tasks with dynamic self-suspension can be handled in $O(n \log n)$ time.

V. Harmonic Tasks

This section discusses optimal priority assignment under preemptive T-FP for synchronous harmonic tasks with dynamic self-suspension, i.e., for all $\tau_i \in \mathbb{T}$

$$\tau_i \in Per(T_i \in T^H, 0) \cap WCET(C_i) \cap DynSus(S_i), \tag{27}$$

where T^H is a set of harmonic periods. As in Section IV, if not stated otherwise, we allow tasks to have constrained deadlines, i.e.,

$$\tau_i \in DL((0, T_i]) \tag{28}$$

for all $\tau_i \in \mathbb{T}$. For this exploration, we first explore exact schedulability tests, and derive optimal priority assignments afterwards. While we show that SADM is only optimal in specific cases, we prove that the *Optimal Priority Assignment (OPA)* algorithm, proposed by Audsley [6], [7], can be used to find optimal priority assignments for the remaining cases.

The exact schedulability test builds upon a similar observation as Lemma 1, namely that carry-in from higher-priority tasks can be avoided by assigning synchronous harmonic periods. The only difference is that more jobs of higher-priority tasks can be released during the execution of the job under analysis $\tau_{k,j}$. Specifically, during the interval $[r_{k,j}, r_{k,j} + t)$, task τ_i can release up to $\left\lceil \frac{t}{T_i} \right\rceil$ jobs. This leads us to the following exact test.

Lemma 3 (Exact test, harmonic)**.** *Let $\mathbb{T}$ be a set of synchronous harmonic tasks with dynamic self-suspension and constrained deadlines. The following is an exact schedulability test under preemptive T-FP scheduling: For all $\tau_k \in \mathbb{T}$ there exists $t \in (0, D_k]$ such that*

$$C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t. \tag{29}$$

Proof. During any interval $[r_{k,j}, r_{k,j} + t)$, a higher-priority task $\tau_i \in \text{HP}(\tau_k)$ releases at most $\left\lceil \frac{t}{T_i} \right\rceil$ many jobs due to the synchronous harmonic periods. Therefore, the available time for a job $\tau_{k,j}$ of task τ_k in the interval $[r_{k,j}, r_{k,j} + t)$, provided that all higher-priority tasks $\text{HP}(\tau_k)$ are schedulable, is:

$$t - \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{30}$$

If $C_k + S_k \leq t - \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i$, then $\tau_{k,j}$ finishes during the interval $[r_{k,j}, r_{k,j} + t]$. Therefore, if there exists a $t \in (0, D_k]$ such that Equation (29) holds, then all jobs of τ_k finish before their deadline, i.e., τ_k is schedulable.

On the other hand, if τ_k is schedulable, then consider the scenario that (i) all higher priority jobs $\tau_{i,j}$ with $\tau_i \in \text{HP}(\tau_k)$ execute for C_i time units and suspend for 0 time units, (ii) τ_k executes for C_k time units and suspends for S_k time units, and (iii) jobs of τ_k suspend only if no higher-priority job is executed. Let $f_{k,1}$ be the finishing time of the first job under that scenario. Since there are exactly $\left\lceil \frac{t}{T_i} \right\rceil$ jobs of each higher-priority task $\tau_i \in \text{HP}(\tau_k)$ released during $[0, f_{k,1})$, we have $C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{f_{k,1}}{T_i} \right\rceil C_i = f_{k,1}$. Therefore, Equation (29) holds with $t = f_{k,1} \in (0, D_k]$. We conclude that the test formulated in Lemma 3 is exact. □

For the special case with implicit deadline, i.e.,

$$\tau_i \in DL(\{T_i\}), \tag{31}$$

there is no need to check all $t \in (0, D_k] = (0, T_k]$, because the best case for Equation (29) is achieved with $t = T_k$. This leads us to the following lemma.

Lemma 4 (Exact test, harmonic, implicit deadline)**.** *Let $\mathbb{T}$ be a set of synchronous harmonic tasks with dynamic self-suspension and* implicit *deadlines. The following is an exact schedulability test under T-FP scheduling: For all $\tau_k \in \mathbb{T}$,*

$$C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i \leq T_k. \tag{32}$$

Proof. Since $t - \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq T_k - \sum_{\tau_i \in \text{HP}(k)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i$ for harmonic periods and $t \leq T_k$, Equation (29) holds for some $t \in (0, D_k] = (0, T_k]$ if and only if Equation (29) holds for $t = T_k$, i.e., it is sufficient to check only $t = T_k$ in Lemma 3. □

While our schedulability test is generally applicable to all T-FP scheduling algorithms, we note that it reduces to the schedulability test proposed by Liu et al. [52] if applied to RM scheduling. Although that schedulability test by Liu et al. was originally only shown to be sufficient, in the following we prove that it is even *exact*.

Corollary 5 (Exactness of [52])**.** *Let $\mathbb{T}$ be a set of synchronous harmonic tasks with dynamic self-suspension and* implicit *deadlines. The following is an exact schedulability test* under RM scheduling*: For all $\tau_k \in \mathbb{T}$, we have*

$$\frac{C_k + S_k}{T_k} + \sum_{\tau_i \in \text{HP}(k)} \frac{C_i}{T_i} \leq 1. \tag{33}$$

Proof. Under RM scheduling, we have $\left\lceil \frac{T_k}{T_i} \right\rceil = \frac{T_k}{T_i}$ for all $\tau_i \in \text{HP}(\tau_k)$. Therefore, Equation (32) from Lemma 4 simplifies to $C_k + S_k + \sum_{\tau_i \in \text{HP}(k)} \frac{T_k}{T_i} C_i \leq T_k$. Dividing both sides by T_k yields Equation (33). □

In Section IV, we show that Suspension-Aware Deadline-Monotonic (SADM), i.e., assigning priorities to tasks τ_k according to $D_k - S_k$, is an optimal T-FP priority assignment for synchronous frame-based tasks. However, SADM is not optimal for synchronous harmonic tasks and dynamic self-suspension, even for tasks with implicit deadlines.

Lemma 6 (Non-Optimality of SADM, harmonic)**.** *In general, even with implicit deadlines, SADM does not provide optimal T-FP priority assignment for synchronous harmonic tasks $\mathbb{T}$ with dynamic self-suspension.*

Proof. We prove that SADM is not optimal in the general case by a counterexample. Suppose we have a system $\mathbb{T}$ with just two tasks, characterized as follows.

- $\tau_1 \in Per(3, 0) \cap DL(\{3\}) \cap WCET(1) \cap DynSus(1)$
- $\tau_2 \in Per(9, 0) \cap DL(\{9\}) \cap WCET(1) \cap DynSus(6)$

Since $T_1 - S_1 = 2$ and $T_2 - S_2 = 3$, then under SADM, τ_1 is prioritized over τ_2. By Equation (32), task τ_2 is *not* schedulable, i.e.,

$$C_2 + S_2 + \left\lceil \frac{T_2}{T_3} \right\rceil C_1 = 1 + 6 + 3 \cdot 1 = 10 > T_2 = 9. \tag{34}$$

However, if τ_2 is prioritized over τ_1, then

$$C_2 + S_2 \leq T_2 \tag{35}$$

$$C_1 + S_1 + \left\lceil \frac{T_1}{T_2} \right\rceil C_2 = 3 \leq T_1 = 3, \tag{36}$$

i.e., tasks τ_1 and τ_2 are both schedulable. □

However, SADM is still optimal if certain conditions are fulfilled as identified in the following lemma. The proof that these conditions are sufficient for optimality is in the appendix.

Lemma 7 (Conditions for Optimality of SADM, harmonic)**.** *If one of the following conditions holds, then SADM is optimal:*

(a) Tasks in $\mathbb{T}$ have implicit deadlines, and for all $\tau_i, \tau_j \in \mathbb{T}$ with $(T_i - S_i) \leq (T_j - S_j)$, we have $T_i \geq T_j$.

(b) Tasks in $\mathbb{T}$ have implicit deadlines, and for all $\tau_i, \tau_j \in \mathbb{T}$ with $(T_i - S_i) \leq (T_j - S_j)$, we have $T_j \cdot (C_j + S_i) \geq T_i \cdot (C_j + S_j)$.

Proof. The proof of the two conditions for SADM optimality can be found in Appendix A. □

For cases where SADM is not optimal, in the following we show that the *Optimal Priority Assignment (OPA)* algorithm, originally proposed by Audsley in [6], [7], can be applied to find a schedulable T-FP priority assignment (if one exists). As the name implies, OPA always finds an optimal priority assignment if one exists. However, to apply OPA, the corresponding schedulability analysis must be *OPA compatible*.

Definition 8 (OPA compatibility; Reformulated from [25])**.** A schedulability test is OPA compatible if the following conditions hold:

(C1) The schedulability of task τ_i may depend on independent properties of higher-priority tasks, but not on any properties that depend on their relative ordering.

(C2) Similarly, for lower-priority tasks.

Algorithm 1: OPA Algorithm

```
Input: Set of tasks 𝕋, schedulability test 𝒮
Output: A schedulable T-FP priority assignment (if one exists)
forall priority levels j, lowest first do
    forall unassigned tasks τ_k ∈ 𝕋 do
        if τ_k is schedulable at priority j according to 𝒮
          with all unassigned tasks assumed to have higher
          priorities then
            Assign τ_k to priority j
            break (continue outer loop)
    return UNSCHEDULABLE
return priority assignments
```

(C3) Say tasks τ_i and τ_j are assigned adjacent priorities, $p_i > p_j$. If τ_j is schedulable with this assignment, it cannot become unschedulable if the priorities are swapped.

The OPA algorithm calls the schedulability test at most $O(n^2)$ times (where n is the number of tasks in $\mathbb{T}$) according to the procedure in Algorithm 1, which we reproduce from [25, Algorithm 1] in our notation.

Theorem 9 (Optimality of OPA, harmonic, constrained deadline)**.** *The schedulability test from Lemma 3 is OPA compatible. Therefore, OPA can derive an optimal T-FP priority assignment.*

Proof. This follows from the conditions for OPA compatability from Definition 8. (C1) is satisfied by the commutative property of addition: Equation (29) is independent of the relative order of higher-priority tasks. (C2) is satisfied since parameters of lower-priority tasks don't appear in Equation (29), i.e., they do not contribute to schedulability of τ_k. And (C3) is satisfied: if a task τ_k is schedulable, and it is swapped with the next higher-priority task, then it remains schedulable because an element is removed from the sum in Equation (29). □

Time Complexity: For each τ_k, Lemma 3 is equivalent to asking for the worst-case response time of a task system $\mathbb{T}_k$ consisting of the **non**-suspending versions (i.e., considering only C_i without S_i of $\tau_i \in \text{HP}(k)$) of the tasks $\text{HP}(k)$ and a modified suspension-oblivious task replacement τ'_k for τ_k with processing times $C'_k := C_k + S_k \leq D_k \leq T_k$. Similarly, Lemma 4 is equivalent to asking for the schedulability of task τ'_k in the corresponding task system $\mathbb{T}_k$.

In $\mathbb{T}_k$, the lowest-priority task is τ'_k and its worst-case response time can be computed in time $O(|\text{HP}(k)| \log(|\text{HP}(k)|)) \leq O(n \log n)$ according to Nguyen et al. [63]. Therefore, testing Equation (29) for a single task τ_k can be done in $O(n \log n)$ time. Furthermore, testing Equation (32) for a single task τ_k can be done in $O(n)$ time.

For a set $\mathbb{T}$ of n tasks, assigning SADM priorities requires sorting tasks τ_k according to $(D_k - S_k)$, which takes time $O(n \log n)$. Since testing the schedulability of a task τ_k takes $O(n)$ time for implicit-deadline task systems (respectively, $O(n \log n)$ time for constrained-deadline task systems), the overall SADM priority assignment and schedulability test take in total $O(n^2)$ time (respectively, $O(n^2 \log n)$ time). OPA runs in time $O(n^2 \times x)$ where x is the time complexity to check schedulability for an individual task. Therefore, the overall OPA approach takes in total $O(n^3)$ time for implicit-deadline task systems (respectively, $O(n^3 \log n)$ time for constrained-deadline task systems). Since SADM is asymptotically faster, the time complexity to first assign SADM priorities while also checking both the optimality of the priorities and the resulting schedulability, then (if the resulting priority assignment is not schedulable but is also not optimal) using OPA is no worse than running OPA only. And, if SADM *is* found to be optimal or to assign schedulable priorities, then OPA is unnecessary.

VI. Extensions and Limitations

Seemingly, the problem of finding optimal priority assignment for T-FP scheduling can be resolved whenever we find an exact schedulability test that is OPA compatible. In this section, we demonstrate the limitations either due to the non-existence of such an exact test in Section VI-A for sporadic real-time tasks with dynamic self-suspension or due to the difficulty to construct an exact test in Sections VI-B and VI-C when the suspension pattern has certain fine-grained structures. Specifically, we show that such an OPA-compatible exact schedulability test cannot exist for *sporadic* tasks. Furthermore, we show that with *fixed numbers of suspension intervals* or *segmented self-suspending tasks*, our analysis remains only exact for frame-based constrained-deadline tasks. Therefore, in future work, the extension of our approach to tasks with fixed numbers of suspension intervals or segmented self-suspending tasks requires the design of new schedulability tests which are OPA compatible. For sporadic tasks, the development of optimal scheduling algorithms requires completely new approaches or a refinement of the OPA compatibility conditions.

A. Sporadic Tasks

To extend our approach to sporadic tasks, i.e.,

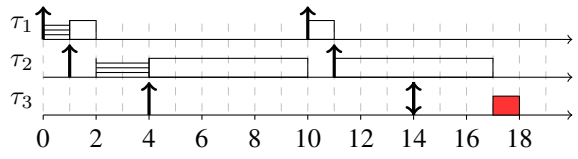
$$\tau_i \in \mathit{Spor}(T_i) \tag{37}$$

for all $\tau_i \in \mathbb{T}$, we need to find a schedulability test which is *exact* and *OPA compatible*. However, as discussed by Liu et al. [52], it is potentially difficult to derive exact schedulability tests for sporadic self-suspending tasks. The underlying reason is the back-to-back influence of self-suspension behavior and release pattern to find the worst-case.

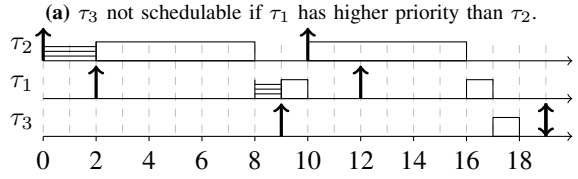
While the difficulties in the analysis were only indicated in [52], an actual concretization for frame-based or harmonic sporadic tasks is missing. While frame-based or harmonic periods potentially limit the search space for worst-case scenarios significantly, the complexity of an exact analysis remains unclear. In the following we show that even if an exact analysis for sporadic self-suspending tasks can be found, that it cannot be OPA compatible. Consequently, the OPA approach is infeasible in the presence of sporadic self-suspending tasks.

Assume an exact schedulability test $\mathcal{T}$. Then we consider the implicit-deadline task set

- $\tau_1 \in \mathit{WCET}(1) \cap \mathit{DynSus}(1) \cap \mathit{Spor}(10) \cap \mathit{DL}(\{10\})$



(a) τ_3 not schedulable if τ_1 has higher priority than τ_2.



(b) τ_3 schedulable if τ_2 has higher priority than τ_1.

Figure 1: Example showcasing that exact schedulability tests for sporadic tasks violate the conditions for OPA compatibility.

- $\tau_2 \in WCET(6) \cap DynSus(2) \cap Spor(10) \cap DL(\{10\})$
- $\tau_3 \in WCET(1) \cap DynSus(0) \cap Spor(10) \cap DL(\{10\})$

The schedulability of task τ_3 depends on the ordering of tasks τ_1 and τ_2. Specifically, as depicted in Figure 1, τ_3 can miss deadlines if τ_1 has a higher priority than τ_2. However, if τ_2 has a higher priority than τ_1, then within every interval of 10 time units, τ_3 can execute for at least 1 time unit, i.e., τ_3 is schedulable. Since any *exact* schedulability test $\mathcal{T}$ has to identify the correct schedulability for τ_3, it has to take the ordering of τ_1 and τ_2 into account. Hence, $\mathcal{T}$ has to violate (C1) from Definition 8, and is therefore not OPA compatible.

We note that in the example every task has the same period and subsequent job releases occur after exactly the minimum inter-arrival time. Therefore, the example is also valid to show that exact schedulability tests for periodic or frame-based tasks with unspecified phase cannot be OPA compatible.

We conclude that the OPA-based procedure of this work is unsuitable to find optimal scheduling algorithms for sporadic tasks. Instead, new approaches have to be explored, or more refined conditions for OPA compatibility must be developed.

B. Fixed Number of Suspension Intervals

This subsection explores whether our approach applies to dynamic self-suspending tasks with fixed numbers of suspension intervals, i.e., for all $\tau_i \in \mathbb{T}$,

$$\tau_i \in DynSus(S_i) \cap NumSus(m_i). \tag{38}$$

To that end, we start by considering **frame-based** tasks with constrained deadlines. Since $DynSus(S_i) \cap NumSus(m_i) \subseteq DynSus(S_i)$, we know that Lemma 1 still provides an upper bound on the response time of any task τ_k. However, due to the additional restriction ($\tau_i \in NumSus(m_i)$), it seems at first glance that Lemma 1 is only sufficient but not necessary. However, the proof of the necessary condition of Lemma 1 only requires task τ_k to suspend at most once. Therefore, the bound R_k from Lemma 1 is exact. Consequently, Theorem 2 applies, and the SADM is an optimal T-FP scheduling algorithm. Therefore even when τ_k only suspends at most once, the optimality of SADM (achievable in $O(n \log n)$ time complexity) is ensured for synchronous, constrained-deadline frame-based dynamic-suspension tasks.

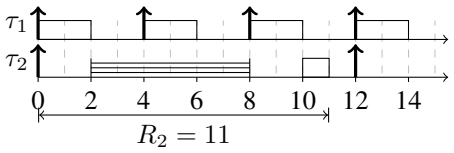


Figure 2: Example showcasing invalidity of Lemma 4.

This result is very interesting as the same setting under segmented self-suspension (with at most one *fixed* suspension interval) is strongly $\mathcal{NP}$-hard. It may seem to be counter-intuitive as the dynamic self-suspension problem is considered to be more difficult than the segmented self-suspension problem. However, the dynamic self-suspension indeed results in a simple worst-case pattern to be handled and analyzed, used in the proof of Lemma 1. We note that this is not the first counter-intuitive scenario in real-time systems. For example, as shown in Table I, the scheduler design problem for a sporadic real-time task system is indeed easier than the same problem for an asynchronous, periodic task system.

For **harmonic** tasks, the exact tests are formulated in Lemma 3 for constrained-deadline tasks and in Lemma 4 for implicit-deadline tasks. The following example shows that the tests are only sufficient but not necessary if the tasks have a fixed number of suspension intervals:

- $\tau_1 \in WCET(2) \cap DynSus(0) \cap NumSus(0) \cap Per(4, 0)$
- $\tau_2 \in WCET(1) \cap DynSus(6) \cap NumSus(1) \cap Per(12, 0)$

For $k = 2$, Lemmas 3 and 4 state that a deadline miss can be observed. However, the worst case, as depicted in Figure 2, is $R_2 = 11$. The reason is that Lemmas 3 and 4 fail to detect the amount of suspension of τ_2 that occurs while τ_1 is being executed in the interval $[4, 6]$. To extend our approach to harmonic tasks with fixed numbers of suspension intervals, the development of an exact OPA-compatible schedulability test remains an open problem.

C. Segmented Self-Suspending Tasks

For segmented self-suspending tasks, i.e.,

$$\tau_i \in SegSus(C_i^1, S_i^1, C_i^2, S_i^2, \ldots, S_i^{m_i}, C_i^{m_i+1}), \tag{39}$$

the limitations are the same as the limitations for fixeds number of suspension intervals in Section VI-B.

That is, for **frame-based** tasks with constrained deadlines, the test of Lemma 1 is still exact, because $SegSus(C_i^1, S_i^1, C_i^2, S_i^2, \ldots, S_i^{m_i}, C_i^{m_i+1}) \subseteq DynSus(\sum_j C_i^j, \sum_j S_i^j)$, and the worst case from Lemma 1 is achieved if all suspension intervals of higher-priority tasks are 0. Therefore, our results from Section IV directly apply, and SADM is an optimal T-FP scheduling algorithm.

Finally, for **harmonic** tasks, the task set $\mathbb{T} = \{\tau_1, \tau_2\}$ with

- $\tau_1 \in SegSus(2) \cap Per(4, 0)$
- $\tau_2 \in SegSus(0, 6, 1) \cap Per(12, 0)$

shows that Lemmas 3 and 4 are only sufficient but not necessary, with worst case depicted in Figure 2, because

Lemmas 3 and 4 are not aware of suspension of τ_2 occurring at the same time as execution of τ_1 during $[4, 6]$. The design of an exact OPA-compatible test remains an open problem.

VII. Evaluation

This paper presents both *optimal priority assignment* schemes and *exact schedulability tests* for frame-based and synchronous, harmonic task sets with dynamic self-suspension and constrained deadlines. To evaluate our approach, we first present a case study of the tasks that make up Autoware's default LiDAR pipeline [40], showing that the required task period can be reduced significantly. Besides this illustration of real-world applicability, we also evaluate our approach using synthetically generated task systems, examining the gain in acceptance compared to the prior state-of-the-art techniques for dynamic self-suspension.

A. Case Study: Analysis of Autoware

Autoware [9], [40], built on ROS2 [57], is an open-source framework for autonomous driving, supporting a broad range of vehicles and applications. The Autoware task system is highly complex, and real-time schedulability analysis challenge is further exacerbated by the fact that many of its tasks offload image processing and ML models to a GPU. In this case study, we first present the suspension patterns of five tasks from Autoware's default LiDAR processing pipeline, then highlight the benefit of optimal priority assignment.

Tasks Evaluated: We evaluate the two primary task chains in Autoware v1.0's "LiDAR pipeline (default)" component [10], comprising the following five tasks (i)–(v):

(i) *lidar_centerpoint* (LC) detects dynamic objects in 5 phases. First, it retrieves and checks the validity of LiDAR point cloud data. Second, it performs GPU preprocessing, including data enqueuing and transformation, buffer initialization, voxelization, and feature generation. Third, it launches GPU operations for detection model inference. Fourth, it generates bounding boxes on the GPU. Fifth, it translates the results into readable messages for downstream publication.

(ii) *obstacle_pointcloud_based_validator* (OPV), which performs CPU-based object validation through density checks of surrounding points within a specified radius.

(iii) *compare_map_filter* (CMF), which uses map data to filter out the ground surface.

(iv) *euclidean_cluster* (EC), which performs CPU-based point clustering to enable object classification.

(v) *shape_estimation* (SE), which produces a label-based fit of refined object shapes to the point clusters in 3 steps. First, incoming sensor data is transformed into the necessary structure. Second, it performs GPU-based shape estimation. Third, results are post-processed and published. Both the LC and SE tasks self-suspend due to GPU offloading; notably, the LC task suspends 3 times at dynamic intervals.

Timing Measurements: We instrumented Autoware v1.0 on a system with an AMD Ryzen 9 3900X 3.8 GHz CPU and an NVIDIA GeForce RTX 3070 Ti GPU running Linux 6.8.0-48-generic. For each CPU-based task, we measured the execution time of each job; for each task with GPU offloading, we measured the total CPU time, total suspension time, and the execution time before and after suspension. Measurements were performed over 698 task iterations.

Task	LC	OPV	CMF	EC	SE
C_i	21	7.8	115	137	10.4
S_i	325	0	0	0	0.41

Table III: Autoware Timing Measurements (ms).

Distributions for the LC task's phases are shown in Figure 3. Notice the non-uniformity in the suspension times of each phase, as well as the CPU execution before and after the GPU offloading phases. Moreover, due to these dynamic execution patterns, using the worst-observed execution time of each phase to characterize the task is pessimistic: the total worst-observed times across CPU phases is 36.37 ms, while the total across GPU phases is 380 ms. Using the dynamic self-suspension model allows us to be more optimistic, since we need to only characterize a *single* worst-case value for each task's execution and suspension times, resulting in the values specified in Table III.

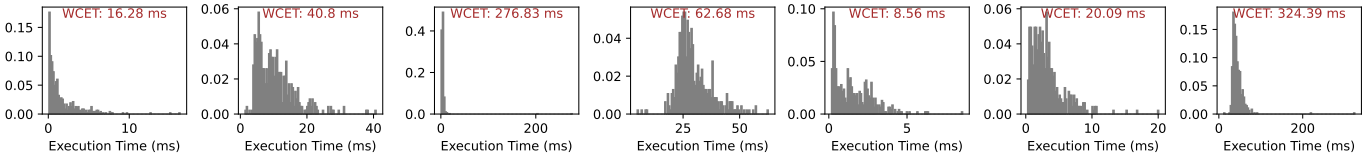
Impact of Priority-Assignment: Using the measurements above, we evaluate the reduction of periods offered by optimal priority assignments. That is, given a priority assignment, an analysis is applied, and the minimal period to achieve schedulability is determined. Since we are considering frame-based tasks, RM priority assignment is equivalent to giving an arbitrary (i.e., random) priority assignment, therefore all 120 possible priority assignments are considered. Figure 4 shows the required periods computed for all 120 possible priority assignments using the following analyses:

SUSPOBL This is the well-known, efficient yet pessimistic, suspension-oblivious analysis [52], where suspension is treated as execution. In other words, a set of self-suspending tasks with execution times C_i and suspension times S_i are tested as non-suspending preemptive tasks with execution times $C'_i = C_i + S_i$.

UNI This is the Unifying Response Time Analysis proposed in [21], which achieves tight analytical results by unifying carry-in-based and jitter-based approaches. This is considered the prior state of the art.

EXACT This is the exact analysis proposed in this paper that safely ignores the suspension behavior of higher-priority tasks. Specifically, this subsection applies Lemma 1.

We observe that the result of SUSPOBL is not impacted by the priority-assignment because it analyzes the tasks as non-self-suspending, and always determines a required period of 617 ms. Furthermore, UNI has the same performance as EXACT, which is due to the fact that only 5 tasks are considered and not all tasks have self-suspension behavior, leaving not much need for UNI to over-approximate the impact of self-suspension. Considering all possible periodizations, then in median a period of 483 ms is required to achieve



(a) Phase 1 (CPU) **(b)** Phase 2 (GPU) **(c)** Phase 3 (GPU) **(d)** Phase 4 (GPU) **(e)** Phase 5 (CPU) **(f)** CPU Total **(g)** GPU Total

Figure 3: Lidar Centerpoint (LC) Task Phase Execution Time Distributions.

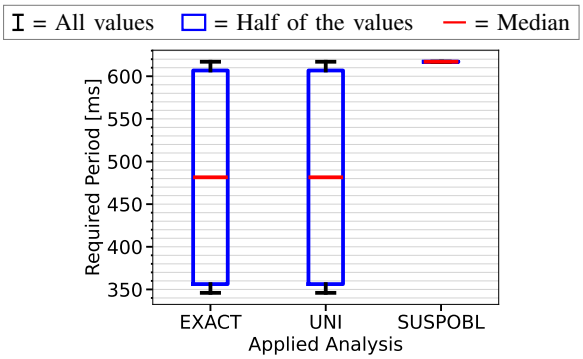


Figure 4: Impact of priority assignment.

schedulability, compared to only 346 ms with the optimal SADM priority assignment, which is a reduction of 28.36%.

B. Synthetic Task Systems

The metric of comparison for the synthetic task set evaluation is the *acceptance ratio* with respect to the task set utilization, i.e., the percentage of task sets deemed schedulable. Note that, in contrast to our proposed exact tests, all considered approaches from the prior work are only *sufficient* in the sense that an accepted task set is guaranteed to be schedulable, but a rejected task set is not guaranteed unschedulable.

Generating Task Sets: We separately consider frame-based and harmonic task systems; for each of these, we consider implicit and constrained deadlines. We generate systems of 5, 10, and 20 tasks. For each type of task system, we consider total utilizations in the range $[0.02, 1.0]$ in steps of 0.02. For each value, we generate 1000 task sets. The total utilization is distributed in an unbiased random fashion among the tasks in each task set using the UUniFast algorithm [12]. For frame-based tasks, periods T_i are randomly selected from a log-uniform distribution (per [30]) in the range $[100, 10\,000]$. For harmonic tasks, periods are selected uniformly from the set $\{100 \cdot 2^n \mid 0 \leq n \leq 7\}$. Execution times C_i are then derived from each task's utilization and period. For each task, a *suspension ratio* s_i is selected uniformly from the range $[0.01, 0.99]$. The total dynamic self-suspension time is then assigned as $S_i = s_i \cdot (T_i - C_i)$ per the methodology in [36]. For constrained-deadline task systems, each task has a deadline D_i drawn uniformly at random from the range $[C_i + S_i, T_i]$, since a task with a deadline less than the sum of the execution and suspension times cannot be scheduled.

Considered Analysis: For the 1000 task sets generated for each combination of task model, task set size, and suspension range, we measure the acceptance ratio according to the following T-FP assignments and schedulability analyses.

For all task sets, we evaluate **schedulability** using the exact analysis presented in this paper (EXACT), the Unifying Response Time Analysis framework (UNI) [21], and suspension-oblivious analysis (SUSPOBL) [52]. Please note that while for Section VII-A only Lemma 1 was utilized for EXACT, this subsection applies Lemma 1, Lemma 3 or Lemma 4, depending on the configuration of the task set.

For all task sets, we evaluate **priority assignment** by comparing our Suspension-Aware Deadline-Monotonic (SADM), where tasks are prioritized according to $D_i - S_i$ (i.e., lowest $D_i - S_i$ has highest priority) to the traditional Deadline-Monotonic (DM). We additionally consider Execution-Monotonic (EM), where priorities are assigned according to C_i (i.e., lowest C_i has lowest priority); and Suspension-Aware Execution-Monotonic (SAEM) where tasks are prioritized according to $C_i + S_i$—again, lowest value meaning lowest priority. For harmonic tasks, we also use the Optimal Priority Assignment (OPA) algorithm from [6], [7], listed in this paper as Algorithm 1.

Results for Frame-Based Tasks: The first two columns of Figure 5 show the acceptance ratios for frame-based tasks with implicit and constrained deadlines, respectively. These results illustrate the *power of SADM*: SADM priority assignments provide significantly better schedulability than the other assignments tested, both under exact analysis, and the earlier unifying framework in [21]. Moreover, the *exact analysis provides a strong improvement over the prior state of the art*: for each priority assignment, the exact analysis dominates the unifying framework by a wide margin, and the simple and well-known suspension-oblivious analysis fails to schedule task sets with processor utilization of only a few percent.

We note that Corollary 5 says that the test given in [52] is exact for frame-based tasks with dynamic self-suspension and implicit deadlines. However, it requires Rate-Monotonic (RM) priority assignment, and is therefore equivalent to the acceptance of "EXACT_DM". The results of this test are shown in the plots in the first column of Figure 5, underperforming SADM by a wide margin.

Results for Harmonic Tasks: The third and fourth columns of Figure 5 show the acceptance ratios for synchronous, harmonic tasks with implicit and constrained deadlines, respectively. As predicted by the theory, OPA with exact analysis dominates the acceptance ratio, and the gap widens as the number of tasks increases. Nonetheless, *SADM remains a powerful tool*: although it is not always optimal, it remains better than the other priority assignments considered, and provides better asymptotic complexity than OPA. In fact, we observe that even the unifying framework with SADM tends to outperform exact analysis with DM, SAEM, and EM.

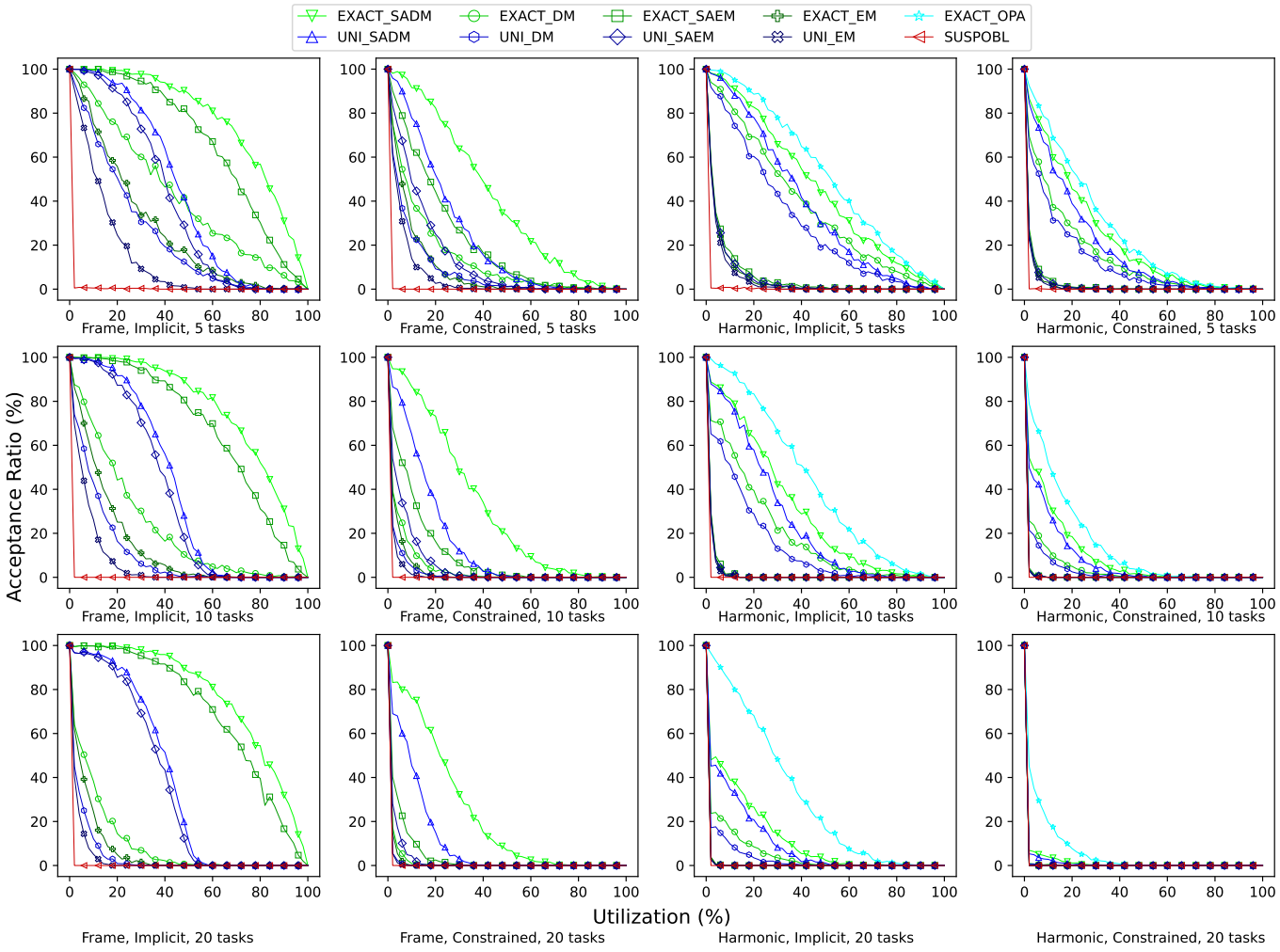


Figure 5: Acceptance ratios for randomly-generated synthetic task sets.

As before, the "EXACT_DM" acceptance data shown in the third column of Figure 5 reflects RM priority assignments (since these are implicit-deadline tasks), and per Corollary 5 is equivalent to the test in [52]. Though it is an efficient and exact test, SADM and OPA priorities, coupled with the exact tests we present, remain significantly more optimistic while still running in polynomial time.

VIII. Conclusion

Despite the existence of extensive examination of self-suspending tasks, the optimality of scheduling algorithms for dynamic suspension is only barely addressed by the prior work. This paper explores optimal priority assignment for synchronous harmonic and frame-based tasks with dynamic self-suspension and constrained deadlines, built upon novel exact schedulability tests.

Specifically, this work goes beyond traditional analysis approaches (i.e., modeling suspension as execution, blocking or jitter) by determining system configurations, where it is safe to ignore the suspension behavior of higher-priority tasks, to built exact schedulability tests. The new tests are used to show that Suspension-Aware Deadline-Monotonic (SADM) is an optimal priority assignment for frame-based tasks. Furthermore, though SADM does not remain optimal in many cases for synchronous harmonic tasks, Audsley's Optimal Priority Assignment (OPA) algorithm can be applied. The evaluation shows that the proposed exact schedulability tests significantly outperform those of the prior state of the art. Moreover, optimal priority assignments lead to a substantial increase in schedulability.

We note that, although the analysis in this paper primarily applies to synchronous frame-based and harmonic task systems, it is nonetheless broadly applicable in real-world applications. Frame-based task scheduling has seen wide adoption in embedded cyber-physical and IoT devices, especially those with energy or thermal constraints [3], [31], [44], [49], [73]. It is also a useful model for scheduling execution along processing chains, e.g., in video streams (where an execution frame corresponds to processing of a single video frame) [69] and in the Autoware LiDAR pipeline that served as a case study in this paper. Moreover, synchronous harmonic task systems (which are a generalization of synchronous frame-based task systems, i.e., $Per(T, 0) \subset Per(T^H, 0)$) are often constructed for convenience and ease of analysis. Indeed, in

many applications, task periods may be somewhat flexible, and methods exist for assigning harmonic periods at design time [59]–[61]. Moreover, many control systems demand harmonic rates, and in applications that capture and synchronize frames from multiple sensing devices, harmonic task periods guarantee consistent temporal alignment [68].

While this work focuses on constrained-deadline tasks, the design of exact schedulability tests and optimal priority assignments for arbitrary-deadline systems remains an open problem. Such examination needs further exploration to resolve carry-in issues inherent to arbitrary-deadline scheduling.

ACKNOWLEDGMENTS

This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170). Funding was also provided in part by NASA award 80NSSC21K1741 and a Washington University OVCR seed grant.

APPENDIX A
PROOFS OF CONDITIONS FOR SADM OPTIMALITY

In this section we prove the two conditions for the optimality of SADM, formulated in Lemma 7.

A. Proof of condition (a)

Assume a task system $\mathbb{T}$ such that for all $\tau_k \in \mathbb{T}$, τ_k is described by Equation (13). Furthermore, assume that for every $\tau_i, \tau_j \in \mathbb{T}$, if $(T_i - S_i) \leq (T_j - S_j)$, then $T_i \geq T_j$.

Without loss of generality, let tasks be indexed in priority order, i.e., $\text{HP}(k) = \tau_{k+1} \cup \text{HP}(k+1)$. Further, assume the system is schedulable, i.e., for all $\tau_k \in \mathbb{T}$, Equation (32) holds. Now, assume that there exist tasks $\tau_i,\ \tau_j,\ i < j$, where $(D_i - S_i) < (D_j - S_j)$. Then it follows that there are adjacent tasks τ_k, τ_{k+1} where $(D_k - S_k) < (D_{k+1} - S_{k+1})$. Since $\mathbb{T}$ is schedulable, Equation (32) says that for task τ_k,

$$C_k + \left\lceil \frac{T_k}{T_{k+1}} \right\rceil C_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i \leq T_k - S_k \quad (40)$$

By the above-stated condition, $T_k \geq T_{k+1}$. If $T_k = T_{k+1}$, then the proof in Theorem 2 of SADM optimality for frame-based tasks says that both τ_k and τ_{k+1} remain schedulable even if their priorities are swapped.

Now, assume $T_k > T_{k+1}$. Then since periods are harmonic, there exists some $a \in \mathbb{N}$, $a > 1$ such that $T_k = a \cdot T_{k+1}$. Then: $C_k + C_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i < C_k + a \cdot C_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{a \cdot T_{k+1}}{T_i} \right\rceil C_i$. Since $\left\lceil \frac{T_k}{T_{k+1}} \right\rceil = a$ and $\left\lceil \frac{T_{k+1}}{T_k} \right\rceil = 1$, this can be restated as:

$$C_{k+1} + \left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i < C_k + \left\lceil \frac{T_k}{T_{k+1}} \right\rceil C_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i$$

By applying Equation (40), we obtain $C_{k+1} + \left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i < T_k - S_k$, and since $T_k - S_k < T_{k+1} - S_{k+1}$, we have:

$$C_{k+1} + \left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i < T_{k+1} - S_{k+1}$$

In particular, τ_{k+1} remains schedulable if τ_k is assigned a higher priority. Furthermore, from Equation (40) we obtain $C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i \leq T_k - S_k$, i.e., τ_k also remains schedulable. It is clear from Equation (19) that the schedulability of other tasks is unaffected when the priorities of τ_k and τ_{k+1} are swapped. By swapping priorities of all such adjacent tasks so that $(T_k - S_k) \geq (T_{k+1} - S_{k+1})$, an SADM T-FP priority assignment is achieved. □

B. Proof of condition (b)

Assume a task system $\mathbb{T}$ such that for all $\tau_k \in \mathbb{T}$, τ_k is described by Equation (13). Furthermore, assume that for every $\tau_i, \tau_j \in \mathbb{T}$, if $(T_i - S_i) \leq (T_j - S_j)$, then $T_j(C_j + S_i) \geq T_i(C_j + S_j)$.

Without loss of generality, let tasks be indexed in priority order, i.e., $\text{HP}(k) = \tau_{k+1} \cup \text{HP}(k+1)$. Further, assume the system is schedulable, i.e., for all $\tau_k \in \mathbb{T}$, Equation (32) holds. Now, assume that there exist tasks $\tau_i,\ \tau_j,\ i < j$, where $(D_i - S_i) < (D_j - S_j)$. Then it follows that there are adjacent tasks τ_k, τ_{k+1} where $(D_k - S_k) < (D_{k+1} - S_{k+1})$. Since $\mathbb{T}$ is schedulable, Equation (40) holds true.

If $T_k \geq T_{k+1}$, then by the proof in Section A-A, the system remains schedulable if the priorities of τ_k and τ_{k+1} swap.

Therefore, it is sufficient to consider the case $T_k < T_{k+1}$. Specifically, if $T_k < T_{k+1}$, then there exists some $a \in \mathbb{N}$, $a > 1$ such that $a \cdot T_k = T_{k+1}$. From Equation (40), we know:

$$aC_k + a\left\lceil \frac{T_k}{T_{k+1}} \right\rceil C_{k+1} + a \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_k}{T_i} \right\rceil C_i \leq aT_k - aS_k$$

We derive: $aC_k + a\left\lceil \frac{T_k}{T_{k+1}} \right\rceil C_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{aT_k}{T_i} \right\rceil C_i \leq aT_k - aS_k$ Since $\left\lceil \frac{T_{k+1}}{T_k} \right\rceil = a$, $aT_k = T_{k+1}$, and $\left\lceil \frac{T_k}{T_{k+1}} \right\rceil = 1$, we have: $\left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + aC_{k+1} + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i \leq T_{k+1} - aS_k$, or equivalently:

$$C_{k+1} + \left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i \leq T_{k+1} - aS_k - (a-1)C_{k+1}$$

By the condition stated above, we obtain that $T_{k+1}(C_{k+1} + S_k) \geq T_k(C_{k+1} + S_{k+1})$. This is equivalent to $S_{k+1} \leq \frac{T_{k+1}S_k}{T_k} + \left(\frac{T_{k+1}}{T_k} - 1\right) C_{k+1}$. Furthermore, since $a = \frac{T_{k+1}}{T_k}$, we have $S_{k+1} \leq aS_k + (a-1)C_{k+1}$. Therefore,

$$C_{k+1} + \left\lceil \frac{T_{k+1}}{T_k} \right\rceil C_k + \sum_{\tau_i \in \text{HP}(k+1)} \left\lceil \frac{T_{k+1}}{T_i} \right\rceil C_i \leq T_{k+1} - S_{k+1}$$

In particular, τ_{k+1} remains schedulable if τ_k is assigned a higher priority. By swapping adjacent tasks, similar to the proof in Section A-A, we show that SADM is optimal. □

References

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. A comprehensive survey of industry practice in real-time systems. *Real Time Syst.*, 58(3):358–398, 2022.

[2] S. Aldegheri, N. Bombieri, D. D. Bloisi, and A. Farinelli. Data flow orb-slam for real-time performance on embedded gpu boards. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5370–5375. IEEE, 2019.

[3] A. Allavena and D. Mosse. Scheduling of frame-based embedded systems with rechargeable batteries. In *Workshop on Power Management for Real-time and Embedded systems (in conjunction with RTAS 2001)*, 2001.

[4] F. Aromolo, A. Biondi, and G. Nelissen. Response-time analysis for self-suspending tasks under EDF scheduling. In *34th Euromicro Conference on Real-Time Systems, ECRTS*, pages 13:1–13:18, 2022.

[5] F. Aromolo, A. Biondi, G. Nelissen, and G. C. Buttazzo. Event-driven delay-induced tasks: Model, analysis, and applications. In *RTAS*, pages 53–65. IEEE, 2021.

[6] N. Audsley. On priority assignment in fixed priority scheduling. Technical report, May 2001.

[7] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS-164, Department of Computer Science, University of York, 1991.

[8] N. C. Audsley and K. Bletsas. Fixed priority timing analysis of real-time systems with limited parallelism. In *16th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 231–238, 2004.

[9] Autoware Foundation. The autoware project, 2023. Accessed: 2024-11-14.

[10] Autoware Foundation. Node diagram - autoware architecture documentation, 2024. Accessed: 2024-11-14.

[11] S. K. Baruah, R. R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

[12] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-time systems*, 30(1):129–154, 2005. Publisher: Springer.

[13] K. Bletsas and N. C. Audsley. Extended analysis with reduced pessimism for systems with limited parallelism. In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 525–531, 2005.

[14] K. Bletsas, N. C. Audsley, W.-H. Huang, J.-J. Chen, and G. Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. *Leibniz Trans. Embed. Syst.*, 5(1):02:1–02:20, 2018.

[15] V. Bonifaci, A. Marchetti-Spaccamela, N. Megow, and A. Wiese. Polynomial-time exact schedulability tests for harmonic real-time tasks. In *IEEE 34th Real-Time Systems Symposium, RTSS*, pages 236–245, 2013.

[16] B. B. Brandenburg. Multiprocessor real-time locking protocols. In Y. Tian and D. C. Levy, editors, *Handbook of Real-Time Computing*, pages 347–446. Springer, 2022.

[17] D. Casini, P. Pazzaglia, A. Biondi, and M. D. Natale. Optimized partitioning and priority assignment of real-time applications on heterogeneous platforms with hardware acceleration. *J. Syst. Archit.*, 124:102416, 2022.

[18] J.-J. Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *Real-Time Systems Symposium (RTSS)*, pages 327–338, 2016.

[19] J.-J. Chen, T. Hahn, R. Hoeksma, N. Megow, and G. von der Brüggen. Scheduling self-suspending tasks: New and old results. In S. Quinton, editor, *31st Euromicro Conference on Real-Time Systems, ECRTS*, volume 133, pages 16:1–16:23, 2019.

[20] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014.

[21] J.-J. Chen, G. Nelissen, and W.-H. Huang. A unifying response time analysis framework for dynamic self-suspending tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[22] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, D. de Niz, and G. von der Brüggen. Many suspensions, many problems: a review of self-suspending tasks in real-time systems. *Real Time Syst.*, 55(1):144–207, 2019.

[23] J.-J. Chen, G. von der Brüggen, W. Huang, and C. Liu. State of the art for scheduling and analyzing self-suspending sporadic real-time tasks. In *23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2017, Hsinchu, Taiwan, August 16-18, 2017*, pages 1–10. IEEE Computer Society, 2017.

[24] K. Chen, M. Günzel, B. Jablkowski, M. Buschhoff, and J.-J. Chen. Unikernel-based real-time virtualization under deferrable servers: Analysis and realization (artifact). *Dagstuhl Artifacts Ser.*, 8(1):02:1–02:2, 2022.

[25] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016.

[26] U. C. Devi. An improved schedulability test for uniprocessor periodic task systems. In *15th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 23–32, 2003.

[27] Z. Dong, C. Liu, S. Bateni, K.-H. Chen, J.-J. Chen, G. von der Brüggen, and J. Shi. Shared-resource-centric limited preemptive scheduling: A comprehensive study of suspension-based partitioning approaches. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 164–176, 2018.

[28] P. Ekberg and W. Yi. Fixed-priority schedulability of sporadic tasks on uniprocessors is np-hard. In *2017 IEEE Real-Time Systems Symposium, RTSS 2017, Paris, France, December 5-8, 2017*, pages 139–146, 2017.

[29] P. Ekberg and W. Yi. *Complexity of Uniprocessor Scheduling Analysis*, pages 1–18. Springer Singapore, Singapore, 2022.

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

[31] M. E. T. Gerards and J. Kuper. Optimal dpm and dvfs for frame-based real-time systems. *ACM Trans. Archit. Code Optim.*, 9(4), Jan. 2013.

[32] M. Günzel and J.-J. Chen. Correspondence article: Counterexample for suspension-aware schedulability analysis of EDF scheduling. *Real Time Syst.*, 56(4):490–493, 2020.

[33] M. Günzel and J.-J. Chen. A note on slack enforcement mechanisms for self-suspending tasks. *Real Time Systems Journal*, 57(4):387–396, 2021.

[34] M. Günzel, N. Ueter, and J.-J. Chen. Suspension-aware fixed-priority schedulability test with arbitrary deadlines and arrival curves. In *42nd IEEE Real-Time Systems Symposium, RTSS*, pages 418–430, 2021.

[35] M. Günzel, G. von der Brüggen, and J. Chen. Tighter worst-case response time bounds for jitter-based self-suspension analysis. In R. Pellizzoni, editor, *36th Euromicro Conference on Real-Time Systems, ECRTS*, volume 298 of *LIPIcs*, pages 4:1–4:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[36] M. Günzel, G. von der Brüggen, and J.-J. Chen. Suspension-aware earliest-deadline-first scheduling analysis. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 39(11):4205–4216, 2020.

[37] M. Günzel, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. EDF-like scheduling for self-suspending real-time tasks. In *IEEE Real-Time Systems Symposium, (RTSS)*, pages 172–184, 2022.

[38] W.-H. Huang and J.-J. Chen. Self-suspension real-time tasks under fixed-relative-deadline fixed-priority scheduling. In *Design, Automation, and Test in Europe (DATE)*, pages 1078–1083, 2016.

[39] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. PASS: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, pages 154:1–154:6, 2015.

[40] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296, 2018.

[41] I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *RTCSA*, pages 54–59, 1995.

[42] J. Kim, B. Andersson, D. de Niz, J.-J. Chen, W.-H. Huang, and G. Nelissen. Segment-fixed priority scheduling for self-suspending real-time tasks. Technical Report CMU/SEI-2016-TR-002, CMU/SEI, 2016.

[43] J. Kim, B. Andersson, D. de Niz, and R. Rajkumar. Segment-fixed priority scheduling for self-suspending real-time tasks. In *Real-Time Systems Symposium, (RTSS)*, pages 246–257, 2013.

[44] C. Krishna. Task sequencing in frame-based cps. *IEEE Embedded Systems Letters*, 13(4):154–157, 2021.

[45] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *IEEE Real-Time Systems Symposium*, pages 160–171, 1991.

[46] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2010.

[47] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *proceedings Real-Time Systems Symposium (RTSS)*, pages 201–209, Dec 1990.

[48] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Eval.*, 2(4):237–250, 1982.

[49] D. Li and J. Wu. Minimizing energy consumption for frame-based tasks on heterogeneous multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 26(3):810–823, 2015.

[50] C.-C. Lin, M. Günzel, J. Shi, T. T. Seidl, K.-H. Chen, and J.-J. Chen. Scheduling periodic segmented self-suspending tasks without timing anomalies. In *29th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 161–173, 2023.

[51] C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.

[52] C. Liu, J.-J. Chen, L. He, and Y. Gu. Analysis techniques for supporting harmonic real-time tasks with suspensions. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 201–210, 2014.

[53] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[54] J. W. S. Liu. *Real-Time Systems*. Prentice Hall PTR, 1st edition, 2000.

[55] S. Liu, R. Wagle, J. H. Anderson, M. Yang, C. Zhang, and Y. Li. Autonomy Today: Many Delay-Prone Black Boxes. In R. Pellizzoni, editor, *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, volume 298 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:27, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[56] W. Liu, J.-J. Chen, A. Toma, T.-W. Kuo, and Q. Deng. Computation offloading by using timing unreliable components in real-time systems. In *Design Automation Conference (DAC)*, volume 39:1 – 39:6, 2014.

[57] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[58] M. Mohaqeqi, P. Ekberg, and W. Yi. On fixed-priority schedulability analysis of sporadic tasks with self-suspension. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS*, pages 109–118, 2016.

[59] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén. Optimal harmonic period assignment: complexity results and approximation algorithms. *Real-Time Systems*, 54(4):830–860, Oct 2018.

[60] M. Nasri and G. Fohler. An efficient method for assigning harmonic periods to hard real-time tasks with period ranges. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 149–159, 2015.

[61] M. Nasri, G. Fohler, and M. Kargahi. A framework to construct customized harmonic periods for real-time systems. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 211–220, 2014.

[62] G. Nelissen, J. Fonseca, G. Raravi, and V. Nélis. Timing Analysis of Fixed Priority Self-Suspending Sporadic Tasks. In *Euromicro Conference on Real-Time Systems (ECRTS)*, pages 80–89, 2015.

[63] T. H. C. Nguyen, W. Grass, and K. Jansen. Exact polynomial time algorithm for the response time analysis of harmonic tasks. In *International Workshop on Combinatorial Algorithms (IWOCA)*, volume 13270, pages 451–465, 2022.

[64] B. Peng and N. Fisher. Parameter adaptation for generalized multiframe tasks and applications to self-suspending tasks. In *International Conference on Real-Time Computing Systems and Applications*, 2016.

[65] P. Richard. On the complexity of scheduling real-time tasks with self-suspensions on one processor. In *Proceedings. 15th Euromicro Conference onReal-Time Systems, (ECRTS)*, pages 187–194, July 2003.

[66] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *RTSS*, pages 47–56, 2004.

[67] L. Schönberger, W. Huang, G. von der Brüggen, K. Chen, and J. Chen. Schedulability analysis and priority assignment for segmented self-suspending tasks. In *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, pages 157–167, 2018.

[68] M. Sudvarg, A. Li, D. Wang, S. Baruah, J. Buhler, C. Gill, N. Zhang, and P. Ekberg. Elastic scheduling for harmonic task systems. In *2024 IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 334–347. IEEE, 2024.

[69] A. Toma and J.-J. Chen. Computation offloading for frame-based real-time tasks with resource reservation servers. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 103–112, 2013.

[70] N. Ueter, M. Günzel, G. von der Brüggen, and J.-J. Chen. Hard real-time stationary gang-scheduling. In *33rd Euromicro Conference on Real-Time Systems, ECRTS*, pages 10:1–10:19, 2021.

[71] G. von der Brüggen, W.-H. Huang, and J.-J. Chen. Hybrid self-suspension models in real-time embedded systems. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2017.

[72] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg. An exact schedulability test for non-preemptive self-suspending real-time tasks. In *Design, Automation & Test in Europe Conference & Exhibition, DATE*, pages 1228–1233, 2019.

[73] Y. Yang and W. Diao. An energy-efficient frame-based task scheduling algorithm for heterogeneous multi-core soc in iot devices. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 1404–1409, 2020.