Uniprocessor Scheduling Strategies for Self-Suspending Task Systems

Georg von der Brüggen[1], Wen-Hung Huang[1], Jian-Jia Chen[1], and Cong Liu[2]
[1]Department of Informatics, TU Dortmund University, Germany
[2]Department of Computer Science, UT Dallas, USA

***Abstract*—We study uniprocessor scheduling for hard real-time self-suspending task systems where each task may contain a single self-suspension interval. We focus on improving state-of-the-art fixed-relative-deadline (FRD) scheduling approaches, where an FRD scheduler assigns a separate relative deadline to each computation segment of a task. Then, FRD schedules different computation segments by using the earliest-deadline first (EDF) scheduling policy based on the assigned deadlines for the computation segments. Our proposed algorithm, Shortest Execution Interval First Deadline Assignment (SEIFDA), greedily assigns the relative deadlines of the computation segments from the task with the smallest execution interval length (i.e., the period minus the self-suspension time). We show that any reasonable deadline assignment under this strategy has a speedup factor of 3. Moreover, we also present how to approximate the schedulability test and a generalized mixed integer linear programming (MILP) that can be formulated based on the tolerable loss in the schedulability test defined by the users. We show by both analysis and experiments that through designing smarter relative deadline assignment policies, the resulting FRD scheduling algorithms yield significantly better performance than existing schedulers for such task systems.**

1 Introduction

Self-suspension has become increasingly important for many real-time applications, due to 1) the interactions with external devices, such as GPUs [19], I/O devices [16], and accelerators [4], 2) multicore systems with shared resources [15], 3) suspension-aware multiprocessor synchronization protocols [5], [21], etc. Introducing suspension delays may negatively impact real-time schedulability, particularly given that such delays can be quite lengthy in many scenarios.

There are two models studied in the literature: *dynamic* and *segmented* self-suspension (sporadic) task models. The *segmented* self-suspension model characterizes the computation segments and suspension intervals as an array $(C_{i,1}, S_{i,1}, C_{i,2}, S_{i,2}, ..., S_{i,m_i-1}, C_{i,m_i})$, composed of m_i computation segments separated by $m_i - 1$ suspension intervals, in which $C_{i,j}$ is the worst-case execution time of a computation segment, and $S_{i,j}$ is the worst-case length of a self-suspension interval. On the other hand, the dynamic self-suspension model allows a job of task τ_i to suspend itself at any moment before it finishes as long as the worst-case self-suspension time S_i is not violated.

It was shown by Ridouard et al. [23] that the scheduler design problem for the segmented self-suspension task model is $\mathcal{NP}$-hard in the strong sense. Chen [7] has recently shown that deciding whether a segmented self-suspension task set can be schedulable by a fixed-priority scheduling policy is $co\mathcal{NP}$-hard in the strong sense. Although the computational complexity for the dynamic self-suspension task model is still unknown in most classes, it was shown by Chen [7] that a wide range of scheduling strategies are with unbounded speedup factors. For the computational complexity and the difficulty to handle self-suspension systems, please refer to [7], [23].

In this paper, we focus on segmented self-suspension task systems, in which a job of a task can suspend at most once, i.e., the execution/suspension behaviour of a job of task τ_i is characterized by $(C_{i,1}, S_{i,1}, C_{i,2})$. To resolve the computational complexity issues in many of these $\mathcal{NP}$-hard scheduling problems in real-time systems, approximation algorithms, and in particular, approximations based on *resource augmentation* (to quantify the worst-case speedup factors, detailed in Section 3.1) have attracted much attention. Designing scheduling algorithms and schedulability tests with bounded speedup factors (resource augmentation factors, equivalently) ensures a bounded gap between the derived solution and the optimal solution for such $\mathcal{NP}$-hard problems.

Overview of related work. The problem of scheduling and analyzing schedulability of real-time suspending tasks has received much attention. For more details, please refer to the recent review paper [11] for scheduling self-suspending tasks in real-time systems. Although the investigation of the impact of self-suspension behaviour in real-time systems has been started since 1990, the literature of this research topic has been seriously flawed as reported in [11].

Here, we summarize the existing results that are directly related to the studied task model, i.e., segmented self-suspending task systems. Under fixed-priority scheduling, Rajkumar [22] proposed a *period enforcer* algorithm to handle the impact of self-suspensions. Although the period enforcer algorithm can be applied for self-suspending tasks with multiple computation segments, Chen and Brandenburg [8] have recently shown that period enforcement can be a cause of deadline misses for self-suspending tasks sets that are otherwise schedulable. Moreover, its schedulability test is also concluded as unknown in [8]. For task systems with at most one self-suspension interval per task, Lakshmanan and Rajkumar proposed two slack enforcement mechanisms in [18], in which the objectives are similar to the period enforcer by shaping the higher-priority jobs so that the higher-priority interference can behave like ordinary periodic tasks. The correctness of the slack enforcement mechanisms was classified as an open issue, since the proofs in [18] were incomplete.

Lakshmanan and Rajkumar [18] also proposed an pseudo-polynomial-time worst-case response time test (*recently shown unsafe by Nelissen et al. [20]*) for a special case, in which there are ordinary sporadic tasks without any self-suspension and one segmented self-suspending task as the lowest-priority task. The sufficient schedulability test by Nelissen et al.

[20] requires exponential-time complexity even when the task system has *only one self-suspending task*. To handle multiple sporadic segmented self-suspending tasks, Nelissen et al. [20] proposed to convert higher-priority tasks into sporadic tasks with jitters, which is unsafe.[1] The methods in [13], [17] assign each computation segment a fixed-priority level and an offset, which was shown incorrect in [11]. For details with respect to these issues, please refer to [11].

Chen and Liu [10] and Huang and Chen [14] proposed to use release time enforcement, called fixed-relative-deadline (FRD), under dynamic-priority scheduling and fixed-priority scheduling, respectively. An FRD scheduler assigns a separate relative deadline to each computation segment of a task and assigns different computation segments different relative deadlines. Thus, relative deadline assignment policies become critical to the performance of FRD scheduling. It is shown in [10] that a rather simple assignment policy, namely equal-deadline assignment (EDA), that assigns relative deadlines equally to the computation segments of a self-suspending task and uses EDF in [10] and fixed-priority in [14] for scheduling the computation segments, yields better performance w.r.t. resource augmentation bound compared to traditional job-level or task-level fixed priority scheduling algorithms. Note that the study in [10] assumed only one self-suspension interval per task. EDA was later shown to have bounded speedup factors in [14] (under both EDF and fixed-priority scheduling) for multiple self-suspension intervals.

Contributions. In this paper we study the problem of scheduling a sporadic self-suspending hard real-time task system on a uniprocessor, where each self-suspending task may contain one suspension interval. We consider implicit-deadline task systems. Although EDA is shown to be superior to traditional real-time schedulers [10], [14] for such cases, its deadline assignment policy is rather straightforward and the potential of FRD scheduling seems not to be fully exploited under EDA.

Our proposed algorithm, Shortest Execution Interval First Deadline Assignment (SEIFDA), considers the deadline assignment from the task with the smallest execution interval length (i.e., the period minus the self-suspension time). When considering task τ_k, SEIFDA greedily chooses any feasible deadline only based on the interference from the other $k-1$ tasks with assigned deadlines, under an assumption that the shorter computation segment of task τ_k has a short relative deadline. This results in several strategies for the deadline selection, as presented in Section 5. We show that SEIFDA by adopting any deadline assignment has a speedup factor of 3 in Section 6. Moreover, we also present how to approximate the schedulability test in Section 7. Section 8 presents a generalized mixed integer linear programming (MILP) that can be formulated based on the tolerable loss in the schedulability test defined by the users. We show by both analysis and experiments that through designing smarter relative deadline assignment policies, the resulting FRD scheduling algorithms yield significantly better performance than existing schedulers for such task systems.

2 Task Model

We consider n independent sporadic one-segment self-suspending real-time tasks $\mathbf{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ in a uniprocessor system. Each task can release an infinite number of jobs (or task instances) under a given minimum inter-arrival time (temporal) constraints T_i, also called the tasks period. This means if a job of task τ_i arrives at time θ_a the next instance of the task must arrive not earlier than $\theta_a + T_i$. For one-segment self-suspending tasks the execution of each job of τ_i is composed of two *computation* segments separated by one *suspension* interval. After the first computation segment is finished the job suspends itself, i.e., for the length of the suspension interval it is removed from the ready queue and the job in the ready queue with the highest priority is executed. The second computation segment is eligible to execute only after the completion of the suspension interval. That is, after the suspension interval of a jobs ends the job will be reentered into the ready queue. A one-segment self-suspending task τ_i is characterized by 3 tuples:

$$\tau_i = ((C_{i,1}, S_i, C_{i,2}), T_i, D_i)$$

where T_i denotes the minimum inter-arrival time of τ_i; D_i denotes the relative deadline of task τ_i; $C_{i,1}$ and $C_{i,2}$ denote the worst case execution time (WCET) of the first and second computation segment respectively; and S_i denotes the upper bound on the suspension time of τ_i.

All the above numbers are positive. For the simplicity of presentations, for the rest of this paper, we implicitly call such tasks as self-suspending tasks as the context is clear. In this work, we restrict our attention to *implicit-deadline* task systems, i.e., $D_i = T_i$.

We do not assume, that each task in the task set must be a self-suspending task. If a task has no self-suspension behavior, there is only one computation segment of task τ_i, which is equivalent to the conventional sporadic task model. In our solution, such ordinary sporadic tasks should still be scheduled by using their original deadlines (and demand bound functions). However, for the simplicity of presentation, we do not consider these tasks in the paper.

For a self-suspending task we denote $C_i = C_{i,1} + C_{i,2}$ and assume that $C_i + S_i \leq D_i$ for any task $\tau_i \in \mathbf{T}$. Furthermore, we denote $C_i^{max} = \max\{C_{i,1},\ C_{i,2}\}$ and $C_i^{min} = \min\{C_{i,1},\ C_{i,2}\}$. The utilization of task τ_i is defined as $U_i = C_i/T_i$. Moreover, we also use $U_{i,1} = C_{i,1}/T_i$ and $U_{i,2} = C_{i,2}/T_i$ for notational brevity. We further assume that $\sum_{i=1}^{n} U_i \leq 1$. We use the following definitions of feasibility and schedulability in this paper:

- A schedule is *feasible* if there is no deadline miss and all the scheduling constraints are respected.
- A self-suspension task system $\mathbf{T}$ is called *schedulable* if there exists a feasible schedule for the task system for any release patterns under the temporal constraints.
- A self-suspension task system $\mathbf{T}$ is *schedulable* under a scheduling algorithm if the schedule produced by the algorithm for the task system is always feasible.

3 Fixed-Relative-Deadline (FRD) Strategies

In this paper, we will adopt the Fixed-Relative-Deadline (FRD) strategies that have been already used in [10]. For each $\tau_i \in \mathbf{T}$ an FRD policy assigns relative deadlines $D_{i,1}$ and $D_{i,2}$ for the executions of the first subtask and the second subtask of τ_i, respectively. Specifically, when a job of task τ_i arrives at time t,

[1] To our knowledge, the erratum is still under preparation by the authors.

- the first subjob (i.e., the first computation segment) has the release time t and its absolute deadline is $t + D_{i,1}$,
- the suspension has to be finished before $t + D_{i,1} + S_i$,
- the second subjob (i.e., the second computation segment) *is enforced* to be released at time $t + D_{i,1} + S_i$ and the absolute deadline $t + D_{i,1} + S_i + D_{i,2}$.

Based on the assigned relative deadlines, each subjob has its own absolute deadline, assigned when a job arrives. The underlying scheduling policy uses the standard earliest-deadline-first (EDF) scheduling to schedule the subjobs with dynamic-priority scheduling.

An FRD has a feasible schedule if the worst-case response time of the first (second, respectively) computation segment of task τ_i is no more than $D_{i,1}$ ($D_{i,2}$, respectively). To ensure the feasibility of the resulting schedule, such a scheduling policy has to ensure that $D_{i,1} + D_{i,2} + S_i \leq T_i$. For the remaining parts of this paper we will always assume that $D_{i,1} + D_{i,2} + S_i = T_i$. Otherwise, if $D_{i,1} + D_{i,2} + S_i < T_i$, we can always increase $D_{i,2}$ by $T_i - S_i - D_{i,1} - D_{i,2}$ without jeopardizing (i.e., reducing) the schedulability of the task set.[2]

3.1 Resource Augmentation Factor

A common approach to quantify the quality of scheduling algorithms (or schedulability tests) is to bound the degree that the considered algorithm may under-perform a (maybe hypothetical) optimal one. To obtain such a bound, we adopt the concept of the resource augmentation factor or speedup factor. When the system is sped up by f, the worst-case execution times $C_{i,1}$ and $C_{i,2}$ become $\frac{C_{i,1}}{f}$ and $\frac{C_{i,2}}{f}$, respectively. However, in this paper, S_i remains the same. Note that there are also other practical scenarios which quantify the speedup factors by reducing the self-suspension time while speeding up. For detailed discussions with regard to this matter, please refer to [7]. Typically, the resource augmentation factor is defined, by referring to any arbitrarily feasible schedule under an optimal scheduling algorithm:

Definition 1. Scheduling algorithm with respect to arbitrary schedules*: We call such a factor the* arbitrary speedup factor. *Provided that the task set* $\mathbf{T}$ *can be feasibly scheduled, an algorithm* $\mathcal{A}$ *is called with an arbitrary speedup factor* α *when algorithm* $\mathcal{A}$ *guarantees to derive a feasible schedule by speeding up the system with a factor* α. □

3.2 Schedulability Test for FRD

Although FRD was introduced in [10], the schedulability tests provided in [10] were mainly for EDA. More general schedulability tests were not provided in [10]. Therefore, in this section, we will first explain how to perform schedulability tests under FRD. We use demand bound functions (DBF) to calculate the maximum cumulative execution time requirement of a task over a given interval $[t_0, t_0 + t)$ when the arrival time of the computation segments have to be within this interval. For the simplicity of presentation, we set t_0 to 0 for the illustrative example used in this section. The concrete appearance of the DBF for an FRD scheduling policy depends only on the value of $D_{i,1}$ as $D_{i,2} = T_i - S_i - D_{i,1}$ as discussed before.

One intuitive way to formulate the DBFs for an FRD scheduling policy is to represent it as a generalized multiframe (GMF) task [2] with two frames depending on the values of $D_{i,1}$, $D_{i,2}$, and S_i. In the GMF task model (with two frames), task τ_i is represented by a vector of 3-tuple $(\overrightarrow{C_i}, \overrightarrow{D_i}, \overrightarrow{T_i})$ where $\overrightarrow{C_i}$, $\overrightarrow{D_i}$, and $\overrightarrow{T_i}$ are 2-ary vectors of identical length representing the WCETs, relative deadlines, and interarrival times of the frames, respectively. The j-th frame of a task τ_i hast the WCET, relative deadline, and interarrival time of the $(j \mod 2)$-th frame. For a one-segment self-suspending task, there are two frames: $\tau_i = \{(C_{i,1}, D_i^1, T_i^1), (C_{i,2}, D_i^2, T_i^2)\}$.[3] As the second computation segment is released after the suspension interval we know that $D_i^1 = D_{i,1}$ and $T_i^1 = D_{i,1} + S_i$. Moreover, $T_i^2 = T_i - T_i^1 = T_i - D_{i,1} - S_i$ and $D_i^2 = D_{i,2} = T_i - D_{i,1} - S_i$. Now we can formulate the DBFs for the case where the segment released at time 0 is represented by the first frame and by the second frame in dbf_i^1 in Eq. (1) and dbf_i^2 in Eq. (2), respectively.

If the first computation segment is released at 0 the segment hast to be finished for $t = D_{i,1}$ while the second segment has to be finished at $t = T_i$. This pattern repeats periodically and is formalized in Eq. (1).

$$dbf_i^1(t, D_{i,1}) = \left\lfloor \frac{t + (T_i - D_{i,1})}{T_i} \right\rfloor C_{i,1} + \left\lfloor \frac{t}{T_i} \right\rfloor C_{i,2} \quad (1)$$

If the second computation segment is released at 0 it has to be finished at $t = D_{i,2}$, the behavior is identical with releasing the first segment at $-(D_{i,1} + S_i)$. This means the first segment has to be finished at $T_i - S_i$. This pattern repeats periodically and is formalized in Eq. (2).

$$dbf_i^2(t, D_{i,1}) = \left\lfloor \frac{t + (D_{i,1} + S_i)}{T_i} \right\rfloor C_{i,2} + \left\lfloor \frac{t + S_i}{T_i} \right\rfloor C_{i,1} \quad (2)$$

The exact DBF for τ_i under an FRD assignment is the maximum of the two possible arrival patterns:

$$\text{DBF}_i^{\text{frd}}(t, D_{i,1}) = \max(dbf_i^1(t, D_{i,1}), dbf_i^2(t, D_{i,1})) \quad (3)$$

Using the DBF in Eq. (3) we can now formulate the exact schedulability test:

Theorem 1 (Exact Schedulability Test for FRD Scheduling Policies)**.** *An FRD schedule is feasible if and only if*

$$\sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{frd}(t, D_{i,1}) \leq t, \qquad \forall t \geq 0.$$

Proof: This follows directly from Theorem 1 in [2], i.e., the schedulability condition for generalized multiframe task systems under EDF using demand bound functions. ■

In Figure 1 we see examples of DBFs for a task with $C_{i,1} = 2$, $C_{i,2} = 3$, $S_i = 4$, and $T_i = 20$ for three different settings of $D_{i,1}$, i.e., $D_{i,1} = 2$ (grey, dashed), 4 (red, solid), and 8 (blue, dotted). For example, with $D_{i,1} = 4$ we get $D_{i,2} = 12$. We have to take care of two cases, depending on weather the computation segment released at time 0 is $C_{i,1}$ or $C_{i,2}$ and take the maximum of both cases as we are looking for the maximum possible workload. If $C_{i,1}$ is released at 0 the DBF equals 0 in the interval $[0, 4)$, as no workload has to be finished up until this point, the maximum workload after $t = 4$ is at least 2, as $C_{i,1}$ has to be finished, and at $t = 20$ it

[2]This can be easily seen by the sufficient test shown in Theorem 1.

[3]We use superscripts to define the terms when we refer to the GMF task.

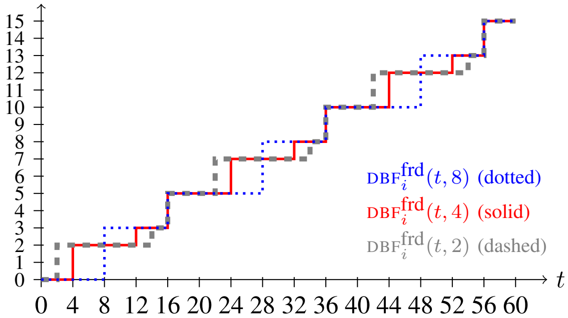

Fig. 1: An example of $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ for different values of $D_{i,1}$, where $C_{i,1} = 2$, $C_{i,2} = 3$, $S_i = 4$, and $T_i = 20$.

is 5 as both $C_{i,1}$ and $C_{i,2}$ have to be finished. When $C_{i,2}$ starts at $t = 0$ it has to be finished at 12 thus the total workload in $[0, 12)$ is 3. At $t = 12$ $C_{i,1}$ is released with absolute deadline 16, followed by the suspension interval, thus the workload is 3 in $[12, 16)$ and 5 in $[16, 20)$ if $C_{i,2}$ is released first. In total by taking the maximum of both cases we get the red line in Figure 1 in $[0, 20)$. As the task is released periodically with period 20, the DBF is also periodic with period 20. This also shows, that we only have 3 jump points in each period as the jump at T_i by $dbf_i^1(t, D_{i,1})$ is already covered by the jump of $dbf_i^2(t, D_{i,1})$ at $T_i - S_i$.

In addition to the exact schedulability test we present two necessary conditions for the schedulability of the task set. One for the schedulability under an FRD assignment and one for any arbitrary scheduling algorithm. This allows to compare our approach to the best possible result any scheduling algorithm could provide.

Lemma 1 (Necessary Condition for FRD Scheduling Policies)**.** *If there exists an FRD schedule to feasibly schedule* $\mathbf{T}$*, then*

$$\sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\textit{frd-nece}}(t) \leq t, \qquad \forall t \geq 0,$$

where

$$\text{DBF}_i^{\textit{frd-nece}}(t) = \left(\left\lfloor \frac{t - (T_i - S_i)}{T_i} \right\rfloor + 1 \right) (C_{i,1} + C_{i,2}) \tag{4}$$

Proof: This was proved in Lemma 1 in [10] with a slightly different formulation of the equation. ■

We now provide a necessary condition for any arbitrary scheduling algorithm for implicit-deadline one-segment self-suspension task sets, assuming that $C_{i,1}$, $C_{i,2}$, and S_i are given. We use Eq. (5) for a lower bound of the workload in the current period which together with the workload created in already finished periods leads to Eq. (6) to calculate a lower bound over a given time interval of length t:

$$\text{G}_i(t) = \begin{cases} 0 & \text{if } 0 \leq t < T_i - S_i \\ C_i^{max} & \text{if } T_i - S_i \leq t < T_i \end{cases} \tag{5}$$

$$\text{DBF}_i^{\text{nece}}(t) = \left\lfloor \frac{t}{T_i} \right\rfloor (C_{i,1} + C_{i,2}) + \text{G}_i \left(t - \left\lfloor \frac{t}{T_i} \right\rfloor \right) \tag{6}$$

Lemma 2. *If task set* $\mathbf{T}$ *can be feasibly scheduled, then*

$$\sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\textit{nece}}(t) \leq t, \qquad \forall t \geq 0.$$

Proof: It is easy to observe that independent from the concrete scheduling policy $C_{i,1} + C_{i,2}$ have to be scheduled after a complete interval of length T_i. What remains is to show, that Eq. (5) is a lower bound on the possible workload distributions over one period.[4] We have to look at the two cases $C_{i,1} \geq C_{i,2}$ and $C_{i,1} < C_{i,2}$. If for both cases a release pattern exists where the jump of the DBF for any arbitrary scheduling policy has to happen before $T_i - S_i$ the proof is done. If $C_{i,1} \geq C_{i,2}$ and we release $C_{i,1}$ at time $t_0 = 0$ the first subjob has to be finished before $T_i - S_i$ as S_i and the execution of $C_{i,2}$ still have to happen before T_i. If $C_{i,1} < C_{i,2}$ we release $C_{i,1}$ at $-S_i - C_{i,1}$ and thus $C_{i,2}$ has to be finished before $T_i - S_i$ independent from the scheduling policy. As both the release patterns and the DBF are periodic with period T_i this concludes the proof. ■

3.3 Existing FRD Approaches

The general concept of FRD approaches was introduced in [10], in which two existing approaches were discussed:

- Proportional (Proportional relative deadline assignment): $D_{i,1} = \frac{C_{i,1}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$; $D_{i,2} = \frac{C_{i,2}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$.
- EDA (Equal relative Deadline Assignment): $D_{i,1} = D_{i,2} = (T_i - S_i)/2$.

At the first glance, Algorithm Proportional may seem very reasonable and Algorithm EDA may seem very pessimistic. Unfortunately, there exists a concrete input task set, as shown in [10], for which the arbitrary speedup factor of Algorithm Proportional is not even a constant. The reason why Algorithm Proportional does not have a constant speedup factor is due to the aggressive relative deadline assignment which greedily sets the $D_{i,1}$ as $\frac{C_{i,1}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$ without considering the interference from the other tasks.

Although Algorithm EDA only greedily assigns the relative deadline, it was already shown in [10] that $\text{DBF}_i^{\text{frd}}(t, (T_i - S_i)/2) \leq \text{DBF}_i^{\text{frd-nece}}(t)$ for any $t \geq 0$. Therefore, the spirit behind Algorithm EDA was to keep this constant factor by setting $D_{i,1}$ to $(T_i - S_i)/2$. Chen and Liu [10] showed that EDA has an arbitrary speedup factor of 3. However, there are still a few drawbacks in Algorithm EDA, even though it has constant speedup factors:

- First, it cannot handle any task set, in which there exists a task τ_i with $C_i^{max} > (T_i - S_i)/2$.
- Second, as shown in Figure 1, the demand bound function by setting $D_{i,1}$ to $(T_i - S_i)/2$ is not always the best option. Assigning $D_{i,1}$ to $(T_i - S_i)/2$ is pretty aggressive.

4 Transformation

Before presenting our solution, we first examine some characteristics of the demand bound function $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$. This section will provide an important transformation of task τ_i to simplify the presentation of the following sections. Since all the step functions in Eqs. (1) and (2) have a period T_i, it is clear that $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ is in general periodic with at most four individual increasing points in a period of T_i.

[4] The remainder of the proof is the same as the proof of Lemma 2 in [10]. Since our condition here is stronger, we include the proof for completeness.

Suppose that we are interested in $\ell T_i \leq t < (\ell+1)T_i$ where ℓ is a non-negative integer. For $dbf_i^1(t)$, we have

- $dbf_i^1(t) = \ell(C_{i,1} + C_{i,2})$ when $\ell T_i \leq t < \ell T_i + D_{i,1}$;
- $dbf_i^1(t) = \ell(C_{i,1} + C_{i,2}) + C_{i,1}$ when $\ell T_i + D_{i,1} \leq t < (\ell+1)T_i$.

For $dbf_i^2(t)$, we have

- $dbf_i^2(t) = \ell(C_{i,1} + C_{i,2})$ when $\ell T_i \leq t < \ell T_i + (T_i - S_i - D_{i,1}) = \ell T_i + D_{i,2}$;
- $dbf_i^2(t) = \ell(C_{i,1} + C_{i,2}) + C_{i,2}$ when $\ell T_i + D_{i,2} \leq t < \ell T_i + T_i - S_i$;
- $dbf_i^2(t) = (\ell+1)(C_{i,1} + C_{i,2})$ when $\ell T_i + T_i - S_i \leq \ell T_i < (\ell+1)T_i$.

Therefore, we know $dbf_i^2(t) \geq dbf_i^1(t)$ if $(t \mod T_i) > T_i - S_i$. Moreover, we also have the following properties:

Lemma 3. *If $C_{i,1} \leq C_{i,2}$ and $D_{i,1} \geq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) \geq \text{DBF}_i^{frd}(t, T_i - S_i - D_{i,1}).$$

Lemma 4. *If $C_{i,1} \geq C_{i,2}$ and $D_{i,1} \leq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) \geq \text{DBF}_i^{frd}(t, T_i - S_i - D_{i,1}).$$

Proof: The proof of these two lemmas follow directly from the definitions. ∎

Therefore, the above lemmas suggest to assign a shorter relative deadline to the shorter computation segment with C_i^{min} for each task τ_i. However, it is notationally inconvenient to distinguish these two difference cases, depending on whether $C_{i,1}$ is smaller or not. Fortunately, the notational complication can be easily handled by swapping $C_{i,1}$ and $C_{i,2}$ if $C_{i,1} > C_{i,2}$, due to the following lemma.

Lemma 5. *Suppose that $C_{i,1} > C_{i,2}$ for a task τ_i. We can create a corresponding task τ_i^* with the same parameters as τ_i but $C_{i,1}$ and $C_{i,2}$ are swapped in task τ_i^*. If $D_{i,1} \geq (T_i - S_i)/2$, then*

$$\forall t \geq 0, \qquad \text{DBF}_i^{frd}(t, D_{i,1}) = \text{DBF}_{i^*}^{frd}(t, T_i - S_i - D_{i,1}),$$

where $\text{DBF}_{i^*}^{frd}(t, T_i - S_i - D_{i,1})$ is the demand bound function of task τ_i^* by setting the relative deadline of the first computation segment in task τ_i^* (i.e., execution time $C_{i,2}$) to $T_i - S_i - D_{i,1}$.

Proof: This can be proved by inspecting the corresponding demand bound functions, as they are identical. ∎

By Lemma 5, *for the rest of this paper, we will implicitly consider that $C_{i,1} \leq C_{i,2}$*. If $C_{i,2} < C_{i,1}$, we should simply reorder them before proceeding to the relative deadline assignment of task τ_i and swap them, together with the assigned deadlines, back after the assignment. By Lemma 5 and the discussions earlier, this does not give any additional restriction, but make our presentation flow much easier.

5 Our Greedy Approach

Our proposed algorithm, Shortest Execution Interval First Deadline Assignment (SEIFDA), works as follows: First, we re-index (sort) the given n tasks such that $T_i - S_i \leq T_j - S_j$ for $i < j$. Then, we start from task τ_1 to task τ_n iteratively to assign their relative deadlines under FRD scheduling. Suppose that the relative deadlines $D_{i,1}$ and $D_{i,2}$ of all tasks $\tau_i \in \{\tau_1, \tau_2, \ldots, \tau_{k-1}\}$ have been already assigned.

Algorithm 1 Shortest Execution Interval First Deadline Assignment (SEIFDA)

Input: set $\mathbf{T}$ of n one-segment self-suspension sporadic real-time tasks with implicit deadlines;
re-index (sort) tasks such that $T_i - S_i \leq T_j - S_j$ for $i < j$;
for $k = 1$ to n **do**
  if $\exists x \in \left(C_{k,1}, \frac{T_k - S_k}{2}\right]$ such that the condition in Eq. (7) holds **then**
    let x^* be one of such values
$$\text{DBF}_k^{\text{frd}}(t, x^*) + \sum_{i=1}^{k-1} \text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq t, \ \ \forall t \geq 0;$$
    set $D_{k,1} \leftarrow x^*$, and $D_{k,2} \leftarrow T_k - S_k - x^*$;
  else
    return "no feasible FRD schedule is found";
  end if
end for
return the relative deadline assignment for each task τ_i in $\mathbf{T}$;

Note that, by the transformation in Section 4, we only have to consider $C_{k,1} \leq C_{k,2}$ for the deadline assignment. If $C_{k,1} > C_{k,2}$ we swap $C_{k,1}$ and $C_{k,2}$ before the deadline assignment, swap them back after the assignment, and swap the respective deadlines as well. As shown in Lemma 3 if a feasible FRD assignment exists we can always assign the deadline of $C_{k,1}$ to a $D_{k,1}$ with $D_{k,1} \leq (T_k - S_k)/2$. To be more precise, if there exists a certain x in the range of $(C_{k,1}, (T_k - S_k)/2]$ such that

$$\text{DBF}_k^{\text{frd}}(t, x) + \sum_{i=1}^{k-1} \text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq t, \ \ \forall t \geq 0, \qquad (7)$$

then, we will greedily assign $D_{k,1}$ to one of such an x value. The pseudocode of Algorithm SEIFDA is presented in Algorithm 1.

5.1 Selection of Relative Deadlines for Task τ_k

Algorithm 1 provides a framework to assign the relative deadlines for FRD scheduling. However, it also leaves an open design option. If there are multiple values of x such that the condition in Eq. (7) holds, which one should be chosen? Due to the greedy strategy, after the relative deadlines are assigned, they will not be changed later. Suppose that x^* is the chosen value of x when considering task τ_k. We are not able to provide the best strategy to choose x^*, but there are several strategies that can be applied:

- Minimum x (denoted by minD): The selection of x^* is to use the minimum x such that Eq. (7) holds.
- Maximum x (denoted by maxD): The selection of x^* is to use the maximum x such that Eq. (7) holds.
- Proportionally-Bounded-Min x (denoted by PBminD): The selection of x^* is to use the minimum $x \geq \frac{C_{i,1}}{C_{i,1}+C_{i,2}}(T_k - S_k)$ such that Eq. (7) holds.

By the above discussions, depending on how we assign $D_{k,1}$ and $D_{k,2}$ in Algorithm SEIFDA, the resulting solutions are different. We denote the combinations that have been presented above by SEIFDA-minD, SEIFDA-maxD, and SEIFDA-PBminD. By the following theorem, EDA is a special case of SEIFDA-maxD and dominated by SEIFDA-maxD.

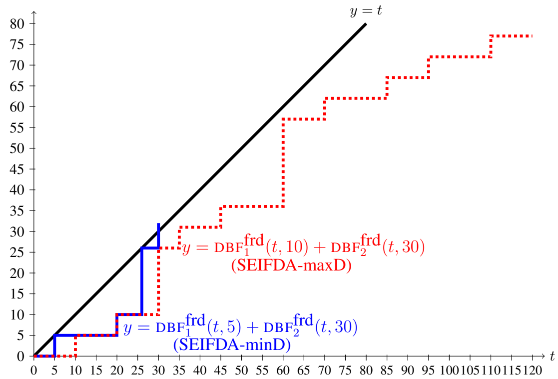

Fig. 2: Schedulability test for Algorithms SEIFDA-maxD (red) and SEIFDA-minD (blue) for the task set in Table I, $\varepsilon = 1$.

					SEIFDA-minD		SEIFDA-maxD	
Task	$C_{i,1}$	$C_{i,2}$	S_i	T_i	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$
τ_1	5	5	5	25	5	15	10	10
τ_2	15+ε	15+ε	940	1000	↯	↯	30	30

TABLE I: An example for comparing SEIFDA-maxD and SEIFDA-minD, where $0 < \varepsilon \leq 1$. ↯ denotes that SEIFDA-minD does not find a feasible value for $D_{2,1}$ and thus $D_{2,2}$ is not assigned either.

Theorem 2. *If a task set* $\mathbf{T}$ *is schedulable by Algorithm EDA, the task set* $\mathbf{T}$ *is also schedulable by Algorithm SEIFDA-maxD.*

Proof: EDA assigns $D_{i,1} = D_{i,2} = (T_i - S_i)/2 \; \forall \tau_i \in \mathbf{T}$. If $\mathbf{T}$ is schedulable by Algorithm EDA, then

$$\forall t \geq 0, \sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\text{frd}}(t, (T_i - S_i)/2) \leq t.$$

When assigning the relative deadlines for task τ_k, Algorithm SEIFDA-maxD assigns the maximum $x \in \left(C_{k,1}, \frac{T_k - S_k}{2}\right]$ that satisfies Eq. (7). Therefore, it always assigns $D_{k,1} = (T_k - S_k)/2 \; \forall \tau_k$. Hence, the resulting deadline assignment by Algorithm SEIFDA-maxD is the same as by EDA. ■

5.2 SEIFDA-maxD and SEIFDA-minD

In the previous subsection we showed that our Algorithm SEIFDA-maxD dominates EDA. It would be also interesting to have such a relation between SEIFDA-minD and EDA or between SEIFDA-maxD and SEIFDA-minD. Here we show that such a relation does not exists by creating one task set, that is schedulable by SEIFDA-maxD but not by SEIFDA-minD (Table I, Figure 2) and another one that is schedulable by SEIFDA-minD but not by SEIFDA-maxD (Table II, Figure 3).

For the task set in Table I SEIFDA-minD assigns $D_{1,1} = 5 \Rightarrow D_{1,2} = 15$, resulting in steps at 5 and 20 for $\text{DBF}_1^{\text{frd}}(t, 5)$, periodically repeated with period 25. This leads to $D_{2,1} \in [25 + \varepsilon; 30]$ as possible values. No matter which value is assigned (in Figure 2 we assume 26) this results in a deadline miss for the second job of $C_{1,1}$ at $t = 30$ as the total workload is $2 \cdot C_{1,1} + C_{1,2} + C_{2,1} = 30 + \varepsilon > 30$. However, the EDA is feasible as $D_{1,1} = 10 \Rightarrow D_{1,2} = 10$ and the second release of $C_{i,1}$ is feasible with absolute deadline 35.

For the task set in Table II SEIFDA-minD assigns $D_{1,1} = \varepsilon \Rightarrow D_{1,2} = 20 + \varepsilon$. For $\text{DBF}_1^{\text{frd}}(t, \varepsilon)$ the steps are at

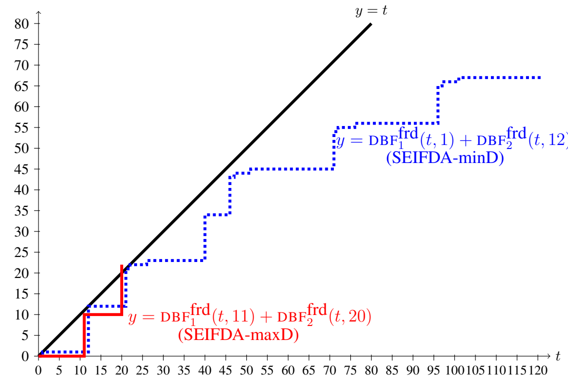

Fig. 3: Schedulability test for Algorithms SEIFDA-maxD (red) and SEIFDA-minD (blue) for the task set in Table II, $\varepsilon = 1$.

ε, 20+ε, and 20+2ε. With $D_{2,1} = 10 + 2\varepsilon$ this leads to a schedulable task set as shown by the DBF in Figure 3. If SEIFDA-maxD is used $D_{1,1} = D_{1,2} = 10 + \varepsilon$. This leads to a deadline miss for $C_{2,1}$ no matter which deadline is assigned (in Figure 2 we assume $C_{2,1} = 20$) as the total workload in the interval $[0, 20]$ is $20 + 2\varepsilon$ if $C_{1,2}$ and at $C_{2,1}$ are both released at 0.

This directly leads to the following theorem:

Theorem 3. *SEIFDA-minD does not dominate SEIFDA-maxD and SEIFDA-maxD does not dominate SEIFDA-minD.*

6 Speedup Factor of SEIFDA

Based on the assumption that $C_{i,1} \leq C_{i,2}$, the following lemma gives the inequalities between $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ and the necessary conditions when $t \geq (T_i - S_i)/2$.

Lemma 6. *Suppose that* $0 < D_{i,1} \leq (T_i - S_i)/2$*. For any* $t \geq (T_i - S_i)/2$*, we have*

$$\text{DBF}_i^{frd}(t, D_{i,1}) \leq 2\text{DBF}_i^{nece}(t) \quad \text{if } T_i - S_i \leq t < T_i + D_{i,1} \tag{8}$$

$$\text{DBF}_i^{frd}(t, D_{i,1}) \leq \text{DBF}_i^{nece}(2t) \quad \text{otherwise} \tag{9}$$

Proof: We consider all the cases when $t \geq (T_i - S_i)/2$:

- If $(T_i - S_i)/2 \leq t < T_i - S_i$, we have $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq C_{i,2} = \text{DBF}_i^{\text{nece}}(T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $T_i - S_i \leq t < T_i + D_{i,1}$, we have $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) = C_{i,1} + C_{i,2} \leq 2\text{DBF}_i^{\text{nece}}(t)$.
- If $T_i + D_{i,1} \leq t \leq (3T_i - S_i)/2$, we have $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq C_{i,1} + 2C_{i,2} = \text{DBF}_i^{\text{nece}}(2T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $(3T_i - S_i)/2 < t < 2T_i + D_{i,1}$, we have $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq 2(C_{i,1} + C_{i,2}) \leq 2C_{i,1} + 3C_{i,2} = \text{DBF}_i^{\text{nece}}(3T_i - S_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.
- If $2T_i + D_{i,1} \leq t$, we have $\text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq (\left\lfloor \frac{t}{T_i} \right\rfloor + 1)(C_{i,1} + C_{i,2}) \leq \text{DBF}_i^{\text{nece}}(t + 2T_i) \leq \text{DBF}_i^{\text{nece}}(2t)$.

■

Theorem 4. *The arbitrary speedup factor of SEIFDA by adopting the schedulability test in Theorem 1 is* 3.

Proof: Suppose that the task set $\mathbf{T}$ cannot be feasibly scheduled by SEIFDA. We will show that this task set is

					SEIFDA-minD		SEIFDA-maxD	
Task	$C_{i,1}$	$C_{i,2}$	S_i	T_i	$D_{i,1}$	$D_{i,2}$	$D_{i,1}$	$D_{i,2}$
τ_1	ε	10	5-2ε	25	ε	20+ε	10+ε	10+ε
τ_2	10+ε	10+ε	960	1000	10+2ε	30-2ε	$\lightning$	$\lightning$

TABLE II: An example for comparing SEIFDA-maxD and SEIFDA-minD, where $0 < \varepsilon \leq 1$. $\lightning$ denotes that SEIFDA-maxD does not find a feasible value for $D_{2,1}$ and thus $D_{2,2}$ is not assigned either.

not schedulable by any algorithm at speed $\frac{1}{3}$. Recall that the tasks are indexed such that $T_i - S_i \leq T_j - S_j$ if $i \leq j$. Let $\mathbf{T}' = \{\tau_1, \tau_2, \dots, \tau_k\}$ be the subset of $\mathbf{T}$ such that task set $\mathbf{T}'$ cannot be feasibly scheduled by SEIFDA, and $\mathbf{T}' \setminus \{\tau_k\}$ can be feasibly schedule by SEIFDA (under the schedulability test in Theorem 1).

If k is 1, it is due to $C_{i,1} + C_{i,2} > T_i - S_i$. For such a case, the arbitrary speedup factor is 1 since the task set is by definition not schedulable by any algorithm at the original system speed. We only focus on the other cases when $k \geq 2$. By the assumption that $\mathbf{T}' \setminus \{\tau_k\}$ can be feasibly scheduled by SEIFDA under the schedulability test in Theorem 1, we have

$$\forall t \geq 0, \qquad \sum_{i=1}^{k-1} \text{DBF}_i^{\text{frd}}(t, D_{i,1}) \leq t, \tag{10}$$

where $D_{i,1}$ is the relative deadline for $C_{i,1}$ under SEIFDA.

The infeasibility of SEIFDA for $\mathbf{T}'$ under the schedulability test in Theorem 1 when we intend to assign the relative deadlines for task τ_k implies that

$$\exists t \geq 0, \qquad \text{DBF}_k^{\text{frd}}\left(t, \frac{T_k - S_k}{2}\right) + \sum_{i=1}^{k-1} \text{DBF}_i^{\text{frd}}(t, D_{i,1}) > t. \tag{11}$$

That is, at least setting $D_{k,1}$ to $(T_k - S_k)/2$ cannot successfully pass the schedulability test in Theorem 1. For notational brevity, we set $D_{k,1}$ to $(T_k - S_k)/2$ for the rest of the proof. This indicates that SEIFDA fails to derive a feasible FRD schedule when assigning $D_{k,1}$ to $(T_k - S_k)/2$.

Suppose that t^* is a certain t such that the above condition in Eq. (11) holds. By the definition that $\text{DBF}_k^{\text{frd}}(t, D_{k,1}) = 0$ when $t < D_{k,1}$ and the assumption that the FRD schedule of $\mathbf{T}' \setminus \{\tau_k\}$ is feasible defined in Eq. (10), we know that t^* must be no less than $D_{k,1}$, defined as $(T_k - S_k)/2$. Since $T_i - S_i \leq T_k - S_k$ for $i = 1, 2, \dots, k$, we also know that $t^* \geq (T_i - S_i)/2$, i.e., the conditions in Lemma 6 are applicable. We further classify the task set $\mathbf{T}'$ into two subsets:

- $\mathbf{T}'_1 =^{\text{def}} \{\tau_i \in \mathbf{T}' \mid T_i - S_i \leq t^* < T_i + D_{i,1}\}$, and
- $\mathbf{T}'_2 =^{\text{def}} \mathbf{T}' \setminus \mathbf{T}'_1$.

That is, for task τ_i in $\mathbf{T}'_1$, we can use the condition in Eq. (8) by Lemma 6; for task τ_i in $\mathbf{T}'_2$, we can use the condition in Eq. (9) by Lemma 6. By the above discussions, we have

$$\begin{aligned} t^* &< \sum_{\tau_i \in \mathbf{T}'_1} \text{DBF}_i^{\text{frd}}(t^*, D_{i,1}) + \sum_{\tau_i \in \mathbf{T}'_2} \text{DBF}_i^{\text{frd}}(t^*, D_{i,1}) \\ &\leq \sum_{\tau_i \in \mathbf{T}'_1} 2\text{DBF}_i^{\text{nece}}(t^*) + \sum_{\tau_i \in \mathbf{T}'_2} \text{DBF}_i^{\text{nece}}(2t^*) \end{aligned} \tag{12}$$

By dividing t^* in both sides, we have

$$1 < 2 \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} + 2 \sum_{\tau_i \in \mathbf{T}'_2} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*}. \tag{13}$$

Since $\mathbf{T}'_1 \cup \mathbf{T}'_2$ is $\mathbf{T}'$ and $\mathbf{T}'_1 \cap \mathbf{T}'_2$ is $\emptyset$, we have

$$y =^{\text{def}} \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} \leq \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*}. \tag{14}$$

$$\begin{aligned} z =^{\text{def}} & \sum_{\tau_i \in \mathbf{T}'_2} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} = \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} \\ \leq & \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - \sum_{\tau_i \in \mathbf{T}'_1} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{2t^*} \\ = & \sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} - y/2 \end{aligned} \tag{15}$$

Therefore, $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} \geq z + y/2$ and $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} \geq y$. By the fact $1 < 2y + 2z$ in Eq. (13), we reach the conclusion that either $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(t^*)}{t^*} > 1/3$ or $\sum_{\tau_i \in \mathbf{T}'} \frac{\text{DBF}_i^{\text{nece}}(2t^*)}{2t^*} > 1/3$.[5] Therefore, the arbitrary speedup factor is 3. ∎

7 Approximated Test and Time Complexity

Although the schedulability test in Theorem 1 is a necessary and sufficient test, it takes exponential time complexity. To make the test faster, we do not have to test for all $t \geq 0$. Instead, we only have to test at the t values where the demand bound function $\text{DBF}_i^{\text{frd}}(t)$ changes. That is, the test in Theorem 1 is equivalent to

$$\forall \tau_i \in \mathbf{T}, \quad \forall t \in \mathbf{\Psi}_i, \quad \sum_{\tau_i \in \mathbf{T}} \text{DBF}_i^{\text{frd}}(t) \leq t \tag{16}$$

$$\mathbf{\Psi}_i = \{D_{i,1} + \ell T_i, T_i - S_i - D_{i,1} + \ell T_i, T_i - S_i + \ell T_i | \ell \in \mathbb{N}^0\} \tag{17}$$

where $\mathbb{N}^0$ is the set of non-negative integers. One may further constrain ℓ to be at most $LCM(\mathbf{T})/T_i$, where $LCM(\mathbf{T})$ is the least common multiple of the periods of the tasks in $\mathbf{T}$. However, the time complexity remains exponential.

To reduce the time-complexity, we can use the approximate demand bound functions, as used in [6], [9]. Our general approach is to use the exact demand bound function for g periods of a task, where g is a user-defined (positive) integer, and use a linear approximation to upper bound the DBF after the given number of periods. Similar to the construction of the exact DBFs we will use one approximated DBF for the case where $C_{i,1}$ is released at $t = 0$ in Eq. (18a), one for the case where $C_{i,2}$ is released at $t = 0$ in Eq. (18b), and take the maximum of both values in Eq. (19).

$$\widehat{dbf}_i^1(t, D_{i,1}) = \begin{cases} dbf_i^1(t, D_{i,1}) & \text{if } t < gT_i \\ U_i t - D_{i,1} U_{i,1} + C_{i,1} & \text{otherwise.} \end{cases} \tag{18a}$$

$$\widehat{dbf}_i^2(t, D_{i,1}) = \begin{cases} dbf_i^2(t, D_{i,1}) & \text{if } t < gT_i - S_i \\ U_i(t + S_i) + C_{i,2} \frac{D_{i,1}}{T_i} & \text{otherwise.} \end{cases} \tag{18b}$$

$$\widehat{\text{DBF}}_i^{\text{frd}}(t, D_{i,1}) = \max(\widehat{dbf}_i^1(t, D_{i,1}), \widehat{dbf}_i^2(t, D_{i,1})) \tag{19}$$

[5]That is, either $y > 1/3$ or $z + y/2 > 1/3$ holds. The can be calculated by using the intersection $z + y/2 = y$, i.e., $z = y/2$, and $1 < 2y + 2z = 3y$.

As the proofs in this section are rather technical and straight forward we just provide the ideas of the proofs here. The complete proofs can be found in the extended report [24].

Theorem 5. *The function* $\widehat{\text{DBF}^{frd}}_i(t, D_{i,1})$ *defined in Eq.* (19) *is a safe upper bound of* $\text{DBF}^{frd}(t, D_{i,1})$ *for any* $t \geq 0$ *and a specified* $D_{i,1} \leq (T_i - S_i)/2$. *Therefore, if* $\sum_{\tau_i \in \mathbf{T}} U_i \leq 1$ *and*

$$\forall t \geq 0, \qquad \sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{frd}}(t, D_{i,1}) \leq t,$$

then the resulting FRD schedule is feasible. Moreover, this schedulability test can be done in $O(g|\mathbf{T}|^2)$ *time complexity.*

Proof: The first part of the proof to show that Eq. (18a) is an over approximation of Eq. (1) and that Eq. (18b) is an over approximation of Eq. (2) can be done by inspecting the corresponding values at the non-linear points of Eq. (1) and Eq. (2), respectively, for $t > gT_i - S_i$. This directly leads to the conclusion that Eq. (19) is an over approximation of Eq. (3). The details are in the appendix of the technical report [24].

For analyzing the time complexity we know we only have to test the schedulability at the points in time where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ changes discontinuously. Each task τ_i has exactly 3 jump points in each of the g periods when $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ (Eq. (19)) is used which leads to $3g$ discrete jump points at $\ell T_i + D_{i,1}$, $\ell T_i + T_i - S_i - D_{i,1}$, and $t = \ell T_i + T_i - S_i$ with $\ell = 0, 1, 2, \ldots, g-1$ for each $\tau_i \in \mathbf{T}$.[6] Let $\mathbf{P}$ be the set of all these $3g|\mathbf{T}|$ jump points of all $\tau_i \in \mathbf{T}$ and let t^* be the maximum of the points in $\mathbf{P}$. It is easy to see that $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ is a linear function for $t > t^*$. Due to the condition that $\sum_{i=1}^n U_i \leq 1$ this means $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1}) \leq t$ for all $t > t^*$. In addition to testing $\sum_{i=1}^n U_i \leq 1$ we have to check all the time points where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ is not linear, i.e., all points in $\mathbf{P}$ which are $3g|\mathbf{T}|$ points in total. As each test hast to calculate the workload up to the tested point for each of the $|\mathbf{T}|$ tasks this leads to $O(g|\mathbf{T}|^2)$ time complexity. ■

In Theorem 5 we proved that a linear approximation of the demand bound functions in Eq. (3) can be calculated in $O(g|\mathbf{T}|^2)$ where $g \in \mathbb{N}^0$ is given and $|\mathbf{T}|$ is the number of tasks in the set. For a good approximated algorithm we need to give some information about the quality of the approximation with relation to the given g, i.e., an upper bound on ratio between over approximation and exact value.

Theorem 6. *For a given integer* $g \geq 1$

$$\forall t \geq 0, \qquad \widehat{\text{DBF}_i^{frd}}(t, D_{i,1}) \leq \left(1 + \frac{1}{g}\right) \text{DBF}_i^{frd}(t, D_{i,1})$$

Proof: We know that both the exact DBFs Eq. (1) and Eq. (2) are step functions with two steps per period, resulting in two intervals with the same value. We have to compare the value they have over this interval to the maximum value the approximated DBF takes over this value. For example, we have to compare the values of $dbf_i^1(gT_i, D_{i,1})$ with $\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1})$ and $dbf_i^1(gT_i + D_{i,1}, D_{i,1})$ with $\widehat{dbf}_i^1((g+1)T_i, D_{i,1})$ to conclude for Eq. (1) compared to Eq. (18a). The details can be found in [24]. ■

This shows that we can use Eq. (19) to formulate Algorithm 1 as an approximation scheme for finding FRD solutions. The needed quality guarantee of $1 + \frac{1}{g}$ (in the schedulability test) follows directly from Theorem 6.

8 Mixed Integer Linear Programming

In this section, we present a programming under logical conditions to assign the relative deadlines of the computation segments. This can be rephrased as a mixed integer linear programming (MILP). In this section, we will use the schedulability test in Theorem 5 by assuming that $g \geq 1$ is given as an integer. Moreover, let $\mathbf{L}$ be $\{0, 1, 2, \ldots, g-1\}$ for notational brevity. We can formulate the studied problem as the following programming under logical constraints:

$$\text{find a feasible solution} \tag{20a}$$

s.t.

$$0 \leq D_{i,1} \leq \frac{T_i - S_i}{2}, \qquad \forall \tau_i \in \mathbf{T} \tag{20b}$$

$$b_{i,j}^{3\ell+1} = \widehat{\text{DBF}_i^{\text{frd}}}(\ell T_i + D_{i,1}, D_{j,1}), \tag{20c}$$

$$b_{i,j}^{3\ell+2} = \widehat{\text{DBF}_i^{\text{frd}}}((\ell+1)T_i - S_i - D_{i,1}, D_{j,1}), \tag{20d}$$

$$b_{i,j}^{3\ell+3} = \widehat{\text{DBF}_i^{\text{frd}}}((\ell+1)T_i - S_i, D_{j,1}), \tag{20e}$$

(Eqs. (20c), (20d), (20e) $\forall \tau_i \in \mathbf{T}, , \tau_j \in \mathbf{T}, \ell \in \{\mathbf{L}\}$)

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+1} \leq \ell T_i + D_{i,1}, \tag{20f}$$

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+2} \leq (\ell+1)T_i - S_i - D_{i,1} \tag{20g}$$

$$\sum_{\tau_j \in \mathbf{T}} b_{i,j}^{3\ell+3} \leq (\ell+1)T_i - S_i \tag{20h}$$

(Eqs. (20f) (20g) (20h) $\forall \tau_i \in \mathbf{T}, \ell \in \{\mathbf{L}\}$)

In the above programming, $D_{i,1}$, $b_{i,j}^h$ are variables that can be assigned to real numbers. The variable $b_{i,j}^{3\ell+1}$ is the approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of task τ_j when $t = \ell T_i + D_{i,1}$. Similarly, the variable $b_{i,j}^{3\ell+2}$ is the approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of task τ_j when $t = \ell T_i + D_{i,2} = (\ell+1)T_i - S_i - D_{i,1}$. The variable $b_{i,j}^{3\ell+3}$ is the approximate demand bound function $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1})$ of task τ_j when $t = \ell T_i + T_i - S_i = (\ell+1)T_i - S_i$. Therefore, the condition in Eqs. (20f), (20g), and (20h) is identical to $\sum_{\tau_j \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{j,1}) \leq t$ when t is $\ell T_i + D_{i,1}$, $(\ell+1)T_i - S_i - D_{i,1}$, and $(\ell+1)T_i - S_i$ for every task τ_i in $\mathbf{T}$ and $\ell = 0, 1, 2, \ldots, g-1$.

Therefore, by Theorem 5, the above programming can be used to search a feasible relative deadline assignment $D_{i,1}$ for $\tau_i \in \mathbf{T}$. In the above programming in Eq. (20), except Eqs. (20c), (20d), and (20e) (due to the logical conditions inherited from Eq. (18a) and Eq. (18b)), the other constraints are linear functions with respect to the variables. Fortunately, by adopting the well-known Big-M Method, each of the above logical conditions in Eqs. (20c), (20d), and (20e) can be

[6] The jump of Eq. (18a) at $(l+1)T_i$ is covered by the jump at $t = \ell T_i + T_i - S_i$ already as at both points the total workload of the DBF is $(l+1)C_i$.

expressed by several linear constraints and several binary variables. Therefore, the above programming can be implemented as an MILP.

9 Experimental Results

We conducted experiments using synthesized task sets to evaluate the proposed approaches compared with other approaches. The metric to compare the results is measuring the *acceptance ratio* of these approaches with respect to the task set utilization. We generated 100 task sets with a cardinality of 10 tasks for each of the analyzed utilization levels that ranged from 0% to 100% with steps of 5%. The acceptance ratio of a level is the percentage of accepted task sets.

For each task set we first generated a set of sporadic tasks with cardinality 10 where the UUniFast method [3] was adopted to generate a set of utilization values with the given goal. We used the approach suggested by Davis et al. [12] to generate the task periods according to an exponential distribution that is of two orders of magnitude, i.e., $[10ms - 1000ms]$. The execution time was set accordingly, i.e., $C_i = T_i U_i$ and the relative deadline was set to the tasks periods, i.e., $D_i = T_i$. We converted them to self-suspending tasks where the suspension lengths of the tasks were generated according to a uniform distribution, in either of three ranges depending on the self-suspension length (sslen):

- short suspension (sslen=Short): $[0.01(T_i - C_i), 0.1(T_i - C_i)]$
- moderate susp. (sslen=Moderate): $[0.1(T_i - C_i), 0.3(T_i - C_i)]$
- long suspension (sslen=Long): $[0.3(T_i - C_i), 0.6(T_i - C_i)]$

We then generated $C_{i,1}$ as a percentage of C_i, according to a uniform distribution, and set $C_{i,2}$ accordingly.

First, we analyzed the acceptance rate of SEIFDA with the approaches minD, maxD, and PBminD for $g \in \{1, 2, 3, 5\}$ and compared it to the MILP approach in Eq. (20) with $g = 1$ under all three types of suspension lengths. In Figure 4 these results are displayed for SEIFDA-minD. In all three subfigures we can see that SEIFDA-minD-1 already does not loose much compared to the MILP with $g = 1$ while SEIFDA-minD-2, SEIFDA-minD-3, and SEIFDA-minD-5 deliver far better results. Also the gap between SEIFDA-minD-2 and SEIFDA-minD-5 is pretty small due to our approximation scheme.[7]

In the next step we compared SEIFDA-minD, SEIFDA-maxD, and SEIFDA-PBminD with each other, using $g = 2$ and $g = 5$, to evaluate the performance of the different approaches as shown in Figure 5. It can be seen that SEIFDA-minD and SEIFDA-PBminD are close to each other. SEIFDA-PBminD has better performance than SEIFDA-minD in most cases. Only for some values with long suspension length SEIFDA-minD performs slightly better. SEIFDA-minD and SEIFDA-PBminD both perform clearly better than SEIFDA-maxD. Even SEIFDA-minD-2 and SEIFDA-PBminD-2 are better than SEIFDA-maxD-5 most of the time.

After that we compared SEIFDA-maxD-5 and SEIFDA-PBminD-5 with the following scheduling approaches:

- *SCEDF*: the suspension-oblivious approach by converting suspension time into computation time.
- EDA: The state-of-the-art approach using Equal-Deadline Assignment (EDA) under linear demand bound approximations, in Theorem 8 in [10].
- MILP: The proposed approach in Section 8 in this paper. Gurobi [1], a state-of-the-art MILP solver, is used to solve Eq. (20) with our manual Big-M Method.
- NC: The necessary condition in Lemma 2. We compared to the necessary condition to know how much we may lose to a theoretical optimal algorithm in the worst case.

We chose SEIFDA-PBminD-5 and SEIFDA-maxD-5 to have an idea about the performance range of SEIFDA-Algorithm. The results are shown in Figure 6. EDA is clearly outperformed by the MILP (g=1), SEIFDA-PBminD-5, and SEIFDA-maxD-5. While NC does not decrease much when the suspension length is increasing the gap of SEIFDA-PBminD-5 to NC gets larger with increasing suspension length.

10 Conclusion and Future Work

In this paper, we investigate uniprocessor scheduling for hard real-time self-suspending task systems where each task may contain a single self-suspension interval. We improve the state-of-the-art by designing new FRD scheduling algorithms that yield significantly better performance than existing approaches, as shown by both analysis and experiments.

As we only consider preemptive scheduling for tasks with one suspension interval on a uniprocessor system we plan to explore multiprocessor scheduling, non-preemptive scheduling and tasks with multiple suspension intervals.

References

[1] Gurobi optimization inc. http://www.gurobi.com. 2016.
[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17:5–22, 1999.
[3] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.
[4] 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.
[5] B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *RTAS*, 2013.
[6] S. Chakraborty, S. Künzli, and L. Thiele. Approximate schedulability analysis. In *IEEE Real-Time Systems Symposium*, pages 159–168, 2002.
[7] J.-J. Chen. Computational complexity and speedup factors analyses for self-suspending tasks. In *Real-Time Systems Symposium, RTSS*, 2016.
[8] J.-J. Chen and B. B. Brandenburg. A note on the period enforcer algorithm for self-suspending tasks. *Computing Research Repository (CoRR)*, 2016. http://arxiv.org/abs/1606.04386.
[9] J.-J. Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *IEEE Real-Time Systems Symposium*, pages 272 – 281, 2011.
[10] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium (RTSS)*, pages 149–160, 2014. **Erratum: http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2014-chen-FRD-erratum.pdf**,.
[11] J.-J. Chen, G. Nelissen, W.-H. Huang, M. Yang, B. Brandenburg, K. Bletsas, C. Liu, P. Richard, F. Ridouard, Neil, Audsley, R. Rajkumar, and D. de Niz. Many suspensions, many problems: A review of self-suspending tasks in real-time systems. Technical Report 854, Faculty of Informatik, TU Dortmund, 2016. http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2016-chen-techreport-854.pdf.
[12] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *Computers, IEEE Transactions on*, 57(9):1261–1276, 2008.

[7]While MILP with $g = 1$ is better than SEIFDA-minD-1, the number of variables and constraints grows quadratically with respect to g in our MILP implementation in Gurobi by using the Big-M Method while SEIFDA is linear with respect to g.

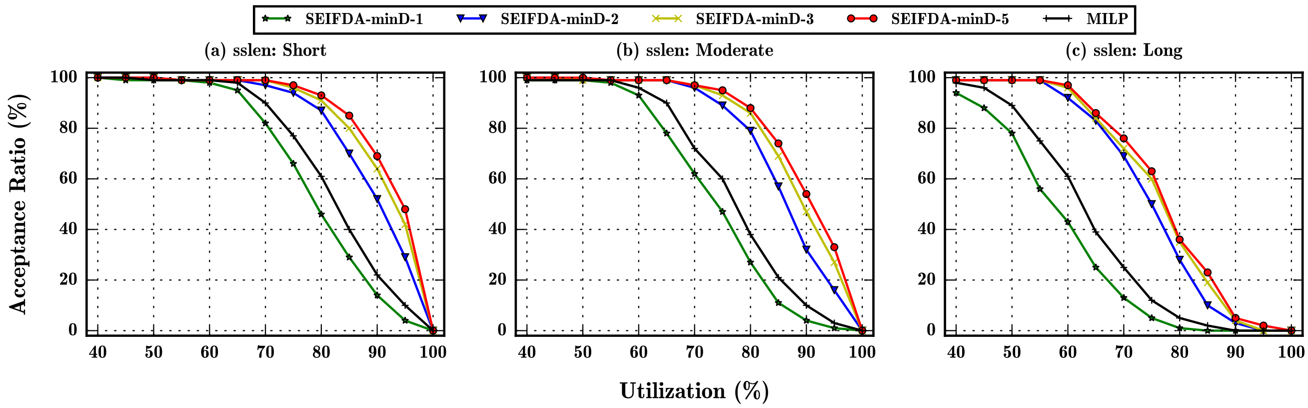

Fig. 4: Impact of the g value for SEIFDA-minD under different suspension lengths (sslen) compared to MILP in Eq. (20) with $g = 1$.

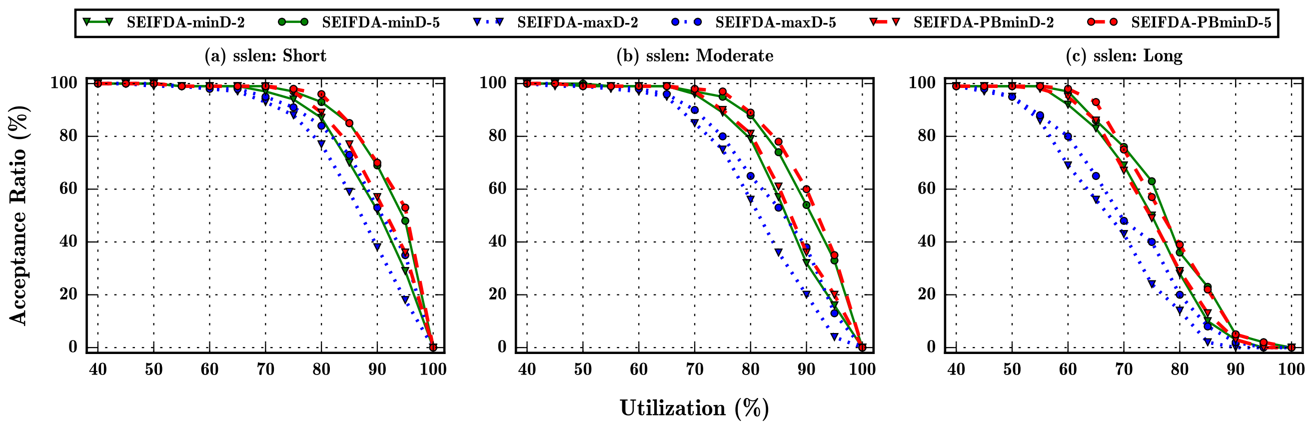

Fig. 5: Comparison of the three presented approaches for SEIFDA: minD, maxD, and PBminD for g-values 2 and 5 under different sslen.

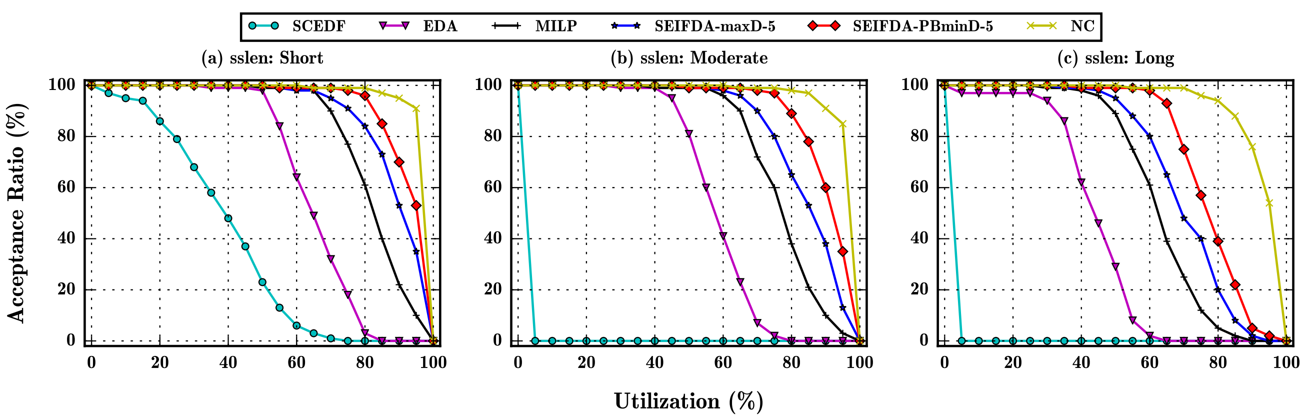

Fig. 6: Comparison of SEIFDA-maxD-5 and SEIFDA-PBminD-5 with suspension-oblivious EDF (SCEDF), EDA, the MILP in Eq. (20) with $g = 1$, and the necessary condition (NC) for schedulability with arbitrary algorithms (Lemma 2) under different suspension lengths (sslen).

[13] S. Ding, H. Tomiyama, and H. Takada. Effective scheduling algorithms for I/O blocking with a multi-frame task model. *IEICE Transactions*, 92-D(7):1412–1420, 2009.

[14] 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)*, 2016.

[15] W.-H. Huang, J.-J. Chen, and J. Reineke. MIRROR: symmetric timing analysis for real-time tasks on multicore platforms with shared resources. In *Design Automation Conference*, pages 158:1–158:6, 2016.

[16] W. Kang, S. Son, J. Stankovic, and M. Amirijoo. I/O-Aware Deadline Miss Ratio Management in Real-Time Embedded Databases. In *Proc. of the 28th IEEE Real-Time Systems Symp.*, pages 277–287, 2007.

[17] 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.

[18] 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.

[19] Y. Liu, C. Liu, X. Zhang, W. Gao, L. He, and Y. Gu. A computation offloading framework for soft real-time embedded systems. In *Euromicro Conference on Real-Time Systems, (ECRTS)*, pages 129–138, 2015.

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

[21] R. Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *ICDCS*, 1990.

[22] R. Rajkumar. Dealing with Suspending Periodic Tasks. Technical report, IBM T. J. Watson Research Center, 1991.

[23] 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.

[24] G. von der Brüggen, W.-H. Huang, J.-J. Chen, and C. Liu. Uniprocessor scheduling strategies for self-suspending task systems. Technical report, 2016. http://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/seifda.pdf.

Appendix

Here we provide some more detailed proofs of Theorem 5 and Theorem 6 together with some additional observations.

Theorem (5). *The function* $\widehat{\text{DBF}^{frd}}_i(t, D_{i,1})$ *defined in Eq.* (19) *is a safe upper bound of* $\text{DBF}^{frd}(t, D_{i,1})$ *for any* $t \geq 0$ *and a specified* $D_{i,1} \leq (T_i - S_i)/2$. *Therefore, if*

$$\forall t \geq 0, \qquad \sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{frd}}(t, D_{i,1}) \leq t,$$

then the resulting FRD schedule is feasible. Moreover, this schedulability test can be done in $O(g|\mathbf{T}|^2)$ *time complexity.*

Proof: We will show that Eq. (18a) is an over approximation of Eq. (1) and that Eq. (18b) is an over approximation of Eq. (2). This directly leads to the conclusion that Eq. (19) is an over approximation of Eq. (3). We only have to check for the parts that are not identical, i.e., $t \geq gT_i$ in Eq. (18a) and $t \geq gT_i - S_i$ in Eq. (18b).

In Eq. (18a) we take care of the case where $C_{i,1}$ is released at the beginning of the period as displayed in Fig 7 for $g = 1$. This means $\widehat{dbf}_i^1(t, D_{i,1})$ is identical with $dbf_i^1(t, D_{i,1})$ for the first g releases of τ_i, i.e., the functions jumps by $C_{i,1}$ at $\ell T_i + D_{i,1}$ and by $C_{i,2}$ at $(\ell+1)T_i$ for $\ell = 0, 1, 2, \ldots, g-1$. Lets look at the g-th release of τ_i. The total workload created by τ_i in an interval of length T_i is C_i and thus we can get a linear and save upper bound by using a straight line with gradient U_i as the task is strictly periodic. Without any adjustment $U_i t$ is a save upper bound for the jump at $(g+1)T_i + D_{i,2}$ as it happens at the end of the period (red). However, if $\frac{C_{i,1}}{D_{i,1}} > U_i$ this is not sufficient to cover the jump at $gT_i + D_{i,1}$. A simple but not tight way is to add $C_{i,1}$ as the resulting linear function covers the case that the jump happens at gT_i instead of $gT_i + D_{i,1}$ (blue). As the utilization created by $C_{i,1}$ in $[gT_i; gT_i + D_{i,1}]$ is $\frac{C_{i,1}}{D_{i,1}}$ we can make the linear approximation tighter by subtracting $D_{i,1}U_{i,1}$ (green). As the tasks are released with a fixed inter arrival time T_i we know that $\widehat{dbf}_i^1(t, D_{i,1})$ is an over approximation of $dbf_i^1(t, D_{i,1})$.

In Eq. (18b) we upper bound the workload for the case that $C_{i,2}$ is released at time $t = 0$, i.e., the functions jumps by $C_{i,2}$ at $\ell T_i + D_{i,2}$ and by $C_{i,1}$ at $\ell T_i + T_i - S_i$ for $\ell = 0, 1, 2, \ldots, g-1$. $U_i(t + S_i)$ is the related linear approximation by a straight line starting at $-S_i$. This is sufficient to cover the jumps at $\ell T_i + T_i - S_i$ as the workload in $[-S_i; T_i - S_i]$ is C_i. We have to make sure that the jump at $\ell T_i + D_{i,2}$ is covered as well. An easy and save upper bound would be to use $U_i(t + S_i + D_{i,1})$, i.e., letting the straight line start $-S_i - D_{i,1}$. We tighten this approach by only adding the amount of utilization that $C_{i,2}$ contributes in an interval of length $D_{i,1}$, i.e. $C_{i,2}\frac{D_{i,1}}{T_i}$, leading to a save upper bound on Eq. (2) for $t \geq gT_i - S_i$.

As both Eq. (18a) and Eq. (18b) are over approximations of Eq. (1) and Eq. (2), respectively, Eq. (19) is an over approximation of Eq. (3).

We know we only have to test the schedulability at the points in time where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ changes. Each task τ_i has exactly 3 jump points in each of the g periods when $\widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ (Eq. (19)) is used which leads to $3g$ discrete jump points at $\ell T_i + D_{i,1}$, $\ell T_i + T_i - S_i - D_{i,1}$, and $t = \ell T_i + T_i - S_i$ with $\ell = 0, 1, 2, \ldots, g-1$ for each $\tau_i \in \mathbf{T}$.[8] Let P be the set of all these $3g|\mathbf{T}|$ jump points of all $\tau_i \in \mathbf{T}$ and let t^* be the maximum of the points in P. It is easy to see that $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ is a linear function for $t > t^*$. This means if $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1}) \leq t$ for some $t > t^*$ it holds $\forall t > t^*$, i.e., the schedulability after t^* according to the linearly approximated test can be tested by testing one $t > t^*$. In addition we have to check all the time points where $\sum_{\tau_i \in \mathbf{T}} \widehat{\text{DBF}_i^{\text{frd}}}(t, D_{i,1})$ is not linear, i,e, all points in P which are $3g\mathbf{T}$ points in total. As each test hast to calculate the workload up to the tested point for each of the $|\mathbf{T}|$ tasks this leads to $O(g|\mathbf{T}|^2)$ time complexity. ∎

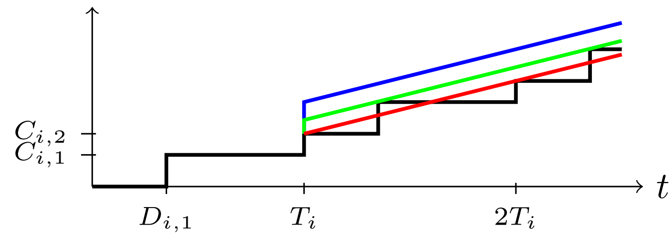

Fig. 7: The linearized DBF for Eq. (18a) with $g = 1$. We use exact steps identical to Eq. (1) up to gT_i and linearization after gT_i. We show the linearization of the original, exact curve (black) without adjustment (red, does not work), linearization after jumping by $C_{i,1}$ at gT_i (blue, over approximation), and after adjusting by $D_{i,1}U_{i,1}$ at gT_i (green, tight).

Theorem (6). *For a given integer* $g \geq 1$

$$\forall t \geq 0, \qquad \widehat{\text{DBF}_i^{frd}}(t, D_{i,1}) \leq \left(1 + \frac{1}{g}\right) \text{DBF}_i^{frd}(t, D_{i,1})$$

Proof: We will check this condition for both cases (Eq. (18a) compared to Eq. (1) and Eq. (18b) compared to Eq. (2)) individually. It is easy to see that it is sufficient to check directly before the jumps of Eq. (1) (Eq. (2), respectively) happen, as between jump points the exact demand bound function is not changing while the approximated BDF is constantly increasing. It is sufficient to check the first period after the beginning of the linearization as with each completed period the values of all equations are increase by C_i which will only lead to a lower ratio.

When we check the ratio of Eq. (18a) compared to Eq. (1) we can obviously use $\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1})$ and $\widehat{dbf}_i^1((g+1)T_i, D_{i,1})$ as upper bounds for each $t \in [gT_i; gT_i + D_{i,1})$ and $t \in [gT_i + D_{i,1}; (g+1)T_i)$, respectively, while $dbf_i^1(gT_i, D_{i,1})$ and $dbf_i^1(gT_i + D_{i,1}, D_{i,1})$ are lower bounds for the values of dbf_i^1 in those intervals. $dbf_i^1(gT_i, D_{i,1}) = g(C_{i,1} + C_{i,2})$ while $\widehat{dbf}_i^1(gT_i + D_{i,1}, D_{i,1}) = U_i(gT_i + D_{i,1}) - D_{i,1}U_{i,1} + C_{i,1} = g(C_{i,1} + C_{i,2}) + D_{i,1}U_i - D_{i,1}U_{i,1} + C_{i,1} = g(C_{i,1} + C_{i,2}) + D_{i,1}U_{i,2} + C_{i,1} < (g+1)(C_{i,1} + C_{i,2})$ as $D_{i,1}U_{i,2} < C_{i,2}$. In the other case we have $dbf_i^1(gT_i + D_{i,1}, D_{i,1}) = g(C_{i,1} + C_{i,2}) + C_{i,1}$ while $\widehat{dbf}_i^1((g+1)T_i, D_{i,1}) = (g+1)(C_{i,1} + C_{i,2}) - D_{i,1}U_{i,1} + C_{i,1} <$

[8] The jump of Eq. (18a) at $(l+1)T_i$ is covered by the jump at $t = \ell T_i + T_i - S_i$ already as at both points the total workload of the DBF is $(l+1)C_i$.

$(g+1)(C_{i,1}+C_{i,2})+C_{i,1}$. By dividing by $(g+1)$ in both cases we reach the conclusion for Eq. (18a).

For Eq. (18b) compared to Eq. (2) we can obviously use $\widehat{dbf}_i^2(gT_i - S_i + D_{i,2}, D_{i,1})$ and $\widehat{dbf}_i^2(gT_i + T_i - S_i, D_{i,1})$ as upper bounds for the values of $\widehat{dbf}_i^2$ in the analyzed interval compared to $dbf_i^2(gT_i - S_i, D_{i,1})$ and $dbf_i^2(gT_i - S_i + D_{i,2}, D_{i,1})$ as lower bounds for the values of dbf_i^2 in those intervals. $dbf_i^2(gT_i - S_i, D_{i,1}) = g(C_{i,1} + C_{i,2})$ while $\widehat{dbf}_i^2(gT_i - S_i + D_{i,2}, D_{i,1}) = U_i(gT_i + D_{i,2}) + C_{i,2}\frac{D_{i,1}}{T_i} = g(C_{i,1} + C_{i,2}) + U_iD_{i,2} + C_{i,2}\frac{D_{i,1}}{T_i} = g(C_{i,1} + C_{i,2}) + D_{i,2}U_i + D_{i,1}U_2 = g(C_{i,1} + C_{i,2}) + C_{i,2} + D_{i,1}U_1 < (g+1)(C_{i,1}+C_{i,2})$ as $D_{i,1}U_1 < C_{i,1}$. In the second case we have $dbf_i^2(gT_i - S_i + D_{i,2}, D_{i,1}) = g(C_{i,1}+C_{i,2})+C_{i,2}$ while $\widehat{dbf}_i^2(gT_i+T_i-S_i, D_{i,1}) = U_i(gT_i+T_i-S_i+S_i)+C_{i,2}\frac{D_{i,1}}{T_i} < (g+1)(C_{i,1}+C_{i,2}) + C_{i,2}$ as $C_{i,2}\frac{D_{i,1}}{T_i} < C_{i,2}$. Dividing by $(g+1)$ in both cases reaches the conclusion for Eq. (18b). ■

As a special case when g is 1, we approximate $\text{DBF}_i^{\text{frd}}(t, D_{i,1})$ by using the following function with only three dis-continuous points (at $D_{i,1}$, $D_{i,2} = T_i - S_i - D_{i,1}$, and $T_i - S_i$) before $T_i - S_i$:

$$\text{DBF}_i^{\text{lin}}(t, D_{i,1}) = \begin{cases} 0 & \text{if } t < D_{i,1} \\ C_{i,1} & \text{if } D_{i,1} \le t < D_{i,2} \\ C_{i,1} + C_{i,2} & \text{if } D_{i,2} \le t < T_i - S_i \\ U_i(t+S_i) + C_{i,2}\frac{D_{i,1}}{T_i} & \text{if } T_i - S_i \le t, \end{cases} \tag{21}$$

where $D_{i,2}$ is defined as $T_i - S_i - D_{i,1}$.

Lemma 7. *The function $\text{DBF}_i^{lin}(t, D_{i,1})$ defined in Eq. (21) is a safe upper bound of $\text{DBF}^{frd}(t, D_{i,1})$ for any $t \ge 0$ and a specified $D_{i,1} \le (T_i - S_i)/2$.*

Proof: This comes from Lemma 5 when g is 1. ■

Lemma 8. *Suppose that $D_{i,1}$ for every task τ_i in $\mathbf{T}$ is given, in which $D_{i,1} \le (T_i - S_i)/2$ and $C_{i,1} \le C_{i,2}$. Let $\mathbf{D}$ be the set $\{D_{i,1} \mid \tau_i \in \mathbf{T}\} \cup \{T_i - S_i - D_{i,1} \mid \tau_i \in \mathbf{T}\} \cup \{T_i - S_i \mid \tau_i \in \mathbf{T}\}$, i.e., $\mathbf{D}$ consists of all the relative deadlines (for both computation segments) and $T_i - S_i$ of the tasks τ_i's in $\mathbf{T}$. The resulting FRD schedule is feasible if $\sum_{i=1}^n U_i \le 1$, and*

$$\sum_{i=1}^{n} \text{DBF}_i^{lin}(t, D_{i,1}) \le t, \forall t \in \mathbf{D}.$$

Proof: This is rather obvious since $\text{DBF}_i^{\text{lin}}(t, D_{i,1})$ can be presented by a linear function if $t \notin \mathbf{D}$ and $t \ge T_i - S_i$. Therefore, we only have to check whether $\sum_{i=1}^n \text{DBF}_i^{\text{lin}}(t, D_{i,1}) \le t, \forall t \in \mathbf{D}$. If this holds, the condition $\sum_{i=1}^n U_i \le 1$ implies that $\sum_{i=1}^n \text{DBF}_i^{\text{lin}}(t, D_{i,1}) \le t, \forall t > 0$ and $t \notin \mathbf{D}$. Therefore, by Lemma 7, we reach the schedulability condition in Theorem 1. ■