Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions

Jian-Jia Chen
Department of Informatics
TU Dortmund University, Germany
jia.chen@tu-dortmund.de

Cong Liu
Department of Computer Science
The University of Texas at Dallas
cong@utdallas.edu

***Abstract*—In many real-time systems, tasks may experience self-suspension delays when accessing external devices. The problem of scheduling such self-suspending tasks to meet hard deadlines on a uniprocessor is known to be $\mathcal{NP}$-hard in the strong sense. Current solutions including the common suspension-oblivious approach of treating all suspensions as computation can be quite pessimistic. This paper shows that another category of scheduling algorithms, namely fixed-relative-deadline (FRD) scheduling, may yield better performance than classical schedulers such as EDF and RM, for real-time tasks that may experience one self-suspension during the execution of a task instance. We analyze a simple FRD algorithm, namely EDA, and derive corresponding pseudo-polynomial-time and linear-time schedulability tests. To analyze the quality of EDA and its schedulability tests, we analyze their resource augmentation factors, with respect to the speed-up factor that is needed to ensure the schedulability and feasibility of the resulting schedule. Specifically, the speed-up factor of EDA is 2 and 3, when referring to the optimal FRD scheduling and any feasible arbitrary scheduling, respectively. Moreover, the speed-up factor of the proposed linear-time schedulability test is 2.787 and 4.875, when referring to the optimal FRD scheduling and any feasible arbitrary scheduling, respectively. Furthermore, extensive experiments presented herein show that our proposed linear-time schedulability test improves upon prior approaches by a significant margin. To our best knowledge, for the scheduling of self-suspending tasks, these are the first results of any sort that indicate it might be possible to design good approximation algorithms.**

1 Introduction

In many real-time systems, tasks may self-suspend when accessing external devices such as disks and GPUs. The resulting suspension delays typically range from a few microseconds (e.g., a read operation on a flash drive [6]) to a few seconds (e.g., accessing a GPU [7], using computation offloading for speeding-up [22]). Such suspension delays cause intractabilities in hard real-time (HRT) schedulability analysis [20]. The unsolved problem of efficiently supporting self-suspensions has impeded research progress on many related research topics such as analyzing and implementing I/O-intensive applications in multiprocessor systems as well as computation offloading in real-time systems.

Due to the fact that the problem of scheduling HRT self-suspending task systems on a uniprocessor is $\mathcal{NP}$-hard in the strong sense [20], it is unlikely to design optimal polynomial-time solutions. 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* have attracted much attention (e.g., real-time task partitioning on multiprocessors [2], [4]). If an algorithm $\mathcal{A}$ has a *speed-up factor* ρ, then it guarantees that *the schedule derived from the algorithm $\mathcal{A}$ is always feasible by running at speed ρ, if the input task set admits a feasible schedule on a unit-speed processor*. In other words, by taking the negation of the above statement, if an algorithm $\mathcal{A}$ has a *speed-up factor* ρ, then it guarantees that *if the schedule derived from the algorithm $\mathcal{A}$ is not feasible, then the input does not admit a feasible schedule by running at speed $\frac{1}{\rho}$*. Therefore, designing scheduling algorithms and schedulability tests with bounded speed-up factors (resource augmentation factors, equivalently) also ensures their qualities for such $\mathcal{NP}$-hard problems.

Resource-augmentation-based approximations assume a certain speedup of the processor. For ordinary real-time task systems without self-suspensions, it has been shown in [18] that well-known priority-based real-time scheduling algorithms, e.g., earliest-deadline-first (EDF) scheduling policy, which have poor performance on a multiprocessor from an absolute worst-case perspective, are good when allowing moderately faster resources. Unfortunately, for the self-suspending task scheduling problem, there does not exist any of such approximation results. Besides the fact that current techniques [8]–[12], [15], [17], [19], [21] for dealing with self-suspensions can be quite pessimistic, none of the existing work provides us a good understanding on how to quantify the quality of such scheduling algorithms.

In this paper, we study the fundamental problem of scheduling an HRT self-suspending task system on a uniprocessor. Classical job-level and task-level fixed-priority scheduling algorithms such as EDF and Rate-Monotonic (RM) may not be suitable for scheduling self-suspending task systems. Several well-known negative results have been shown on supporting the *single-segment-suspending* task model, where each task contains two computation phases with one suspension phase in between, under job- or task-level fixed-priority scheduling algorithms including EDF and RM [20]. Our key observation herein is that when considering self-suspensions, job priorities should be determined not solely by traditional parameters such as periods or deadlines, but also the suspension length. Consider, for example, the uniprocessor task system, scheduled by EDF or RM, shown in Fig. 1. This system consists of two tasks: task τ_1 is an ordinary sporadic

This paper has been supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/), and the priority program "Dependable Embedded Systems" (SPP 1500 - http://spp1500.itec.kit.edu).

task with an execution cost of 1 time unit, a period and a relative deadline of 5 time units; task τ_2 is a self-suspending task with a period and a relative deadline of 10 time units that first executes for 1 time unit, then self-suspends for 8 time units, and finally executes for another 1 time unit. As seen from the figure, τ_2 misses its deadline under either EDF or RM. However, if we prioritize the first computation phase of τ_2 over the computation phase of τ_1, then both tasks can meet their deadlines. Although the job of τ_2 has a longer relative deadline than the job of τ_1, its first computation segment actually must meet an "invisible" hard deadline at time 1 in order for the entire job to meet its deadline.

Motivated by this, we show in this paper that for scheduling single-segment-suspending task systems, another category of scheduling algorithms, namely *fixed-relative-deadline* (FRD) scheduling, may yield better performance than traditional job-level or task-level fixed priority schedulers. An FRD scheduler assigns a separate relative deadline to each computation phase of a task and prioritizes different computation phases by these relative deadlines. This approach has been adopted by Liu et al. [16] for computation offloading. The approach in [16] greedily assigns the relative deadline proportionally to the execution time. We will show in this paper that such a proportional approach is not good w.r.t. speed-up factor. We further observe and prove that, surprisingly, a rather simple FRD scheduling policy, namely equal-deadline assignment (EDA) that assigns relative deadlines equally to both computation phases of a self-suspending task and uses EDF for scheduling the computation phases, yields good performance w.r.t. speed-up factor.

Overview of related work. Recently the problem of scheduling soft real-time (with guaranteed bounded response times) self-suspending task systems on multiprocessor has received much attention [10], [11]. For the HRT case, besides the suspension-oblivious approach of treating all suspensions as computation [15], several schedulability tests have been presented for analyzing periodic single-segment-suspending tasks on a uniprocessor [8], [9], [17], [19], [21]. The result from [9] can only be applied when the system has only one self-suspended task. Unfortunately, these tests are rather pessimistic as their techniques involve straightforward execution control mechanisms, which divide a self-suspending task into two subtasks with appropriately shortened deadlines and modified release times (often known as the *end-to-end* approach [15]). *The FRD scheduling policy can be considered as an end-to-end approach with static assignments of the relative deadline of the first sub-task and (relative) release time of the second subtask of a self-suspended task.*

For the more general self-suspending task model where a task is allowed to suspend multiple times, [14] presents a uniprocessor utilization-base test under RM and a multiprocessor utilization-based test under partitioned approach where RM is applied as the per-processor scheduler. However, the analysis techniques and the tests presented in [14] only applies to *synchronous periodic* task systems with *harmonic* periods. On multiprocessors, [12] presents the only existing global suspension-aware analysis for periodic self-suspending task systems scheduled under global EDF and global fixed-priority schedulers. In another recent work [13], we consider the general case of the self-suspension task model, where no restriction is placed on the number of per-job suspension segments and the computation and suspension pattern. We develop a general interference-based analysis framework that can be applied to derive sufficient utilization-based tests for the general self-suspending task model. This paper is completely different from the above-mentioned work [13], w.r.t. the targeted problem, the proposed analysis technique, and the format of the solution.

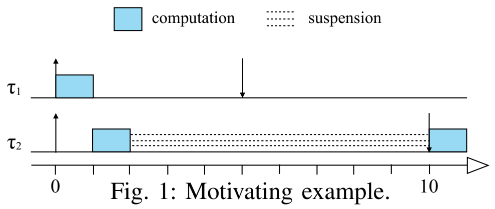


Fig. 1: Motivating example.

Contributions. We answer an important question in this research: for a given sporadic single-segment-suspending task system, if a feasible schedule exists upon a unit-speed processor, can we design a scheduling algorithm that will lead to a feasible schedule when being allowed with moderately faster resources? To answer this question, we first show that classical uniprocessor schedulers including EDF and RM yield poor performance for scheduling self-suspending task systems, as they yield a resource augmentation bound, that is infinite. As an alternative, we observe and prove that a rather simple FRD scheduler, EDA, yields non-trivial resource-augmentation performance guarantees w.r.t. any FRD scheduler and any arbitrary scheduler. Specifically, we derive a pseudo-polynomial-time schedulability test for EDA that is exact and has a resource-augmentation bound of 2 and 3 w.r.t. any FRD scheduler and any arbitrary scheduler, respectively. To reduce the time complexity, we further present a linear-time schedulability test for EDA, which yields a resource-augmentation bound of 2.787 and 4.875, w.r.t. any FRD scheduler and any arbitrary scheduler, respectively. Furthermore, experiments presented herein show that our proposed schedulability tests improve upon prior tests by a large margin in all cases. Moreover, our linear-time schedulability test under EDA achieves little or even no utilization loss in many cases.

2 System Model

We consider the problem of scheduling a set $\mathbf{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ of n independent sporadic self-suspending (SSS) tasks on one processor. Each task is released repeatedly, with each of such invocations called a job. Jobs alternate between computation and suspension phases. We assume that each job of τ_i contains at most two computation phases separated by one suspension phase. This suspending task model (as considered in numerous prior work [8], [15], [19], [21]) actually covers a large set of real-world applications that involve self-suspension behaviors. For example, in many multimedia applications, a common task is to initialize the video processing code, then fetch video data from the disk (which can be modeled by a self-suspension phase), and finally process the video data.

Each task τ_i is characterized by the following parameters.

- minimum inter-arrival time (also called period) T_i,
- worst-case execution time $C_{i,1}$ on the first computation phase,

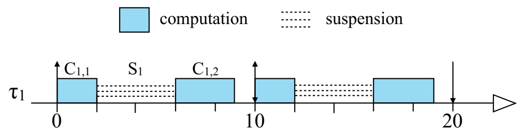


Fig. 2: An example SSS task τ_1 with $C_{i,1} = 2$, $S_1 = 4$, $C_{i,2} = 3$, $D_i = T_i = 10$, and a utilization of 0.5.

- worst-case suspension time S_i of a task instance, and
- worst-case execution time $C_{i,2}$ on the second computation phase,
- relative deadline D_i.

If $C_{i,2}$ is 0, it means that there is only one computation phase for task τ_i. If S_i is 0, we also implicitly assume that $C_{i,2}$ is 0 and task τ_i does not self-suspend.

An SSS task set $\mathbf{T}$ is said to be an *implicit-deadline* task set if $D_i = T_i$ holds for each τ_i. Due to space constraints, we limit our attention to implicit-deadline SSS tasks in this paper.

According to our SSS task model, each task τ_i can be considered as two subtasks representing the two computation phases, denoted $\tau_{i,1}$ and $\tau_{i,2}$. An example SSS task is given in Fig. 2. The j^{th} job of τ_i, denoted τ_i^j, is released time r_i^j and has a deadline at time $D_i^j = r_i^j + D_i$. Similarly, each job τ_i^j consists of two subjobs (separated by a suspension), denoted $\tau_{i,1}^j$ and $\tau_{i,2}^j$. Successive jobs of the same task are required to execute in sequence. Note that, when a job of a task misses its deadline, the release time of the next job of that task is not altered. The utilization of τ_i is defined as $U_i = C_i/T_i$, and the utilization of the task set $\mathbf{T}$ as $U_{sum} = \sum_{\tau_i \in \mathbf{T}} U_i$. We require $C_{i,1} + S_i + C_{i,2} \leq D_i$, $U_i \leq 1$, and $U_{sum} \leq 1$; otherwise, deadlines can be missed. We denote $\max\{C_{i,1}, C_{i,2}\}$ as $C_{i,\max}$ and $\min\{C_{i,1}, C_{i,2}\}$ as $C_{i,\min}$, respectively.

3 Our Scheduling Policy and Speed-Up Factors

3.1 Fixed-Relative-Deadline Scheduling

As discussed in Sec. 1, it is not wise to schedule SSS tasks under EDF and RM. A major reason is because the per-job deadline parameter cannot accurately represent the urgency of a self-suspending job. For an ordinary sporadic task τ_i, the time available for its completion is given by D_i. However, for a self-suspending task τ_i, the time available for its completion is actually given by $D_i - S_i$. For instance, for the task system shown in Fig. 1, τ_2 has only two time units available for its completion of its two subtasks. Subjob $\tau_{2,1}^1$ needs to be completed by time 1 in order for job $\tau_{2,1}$ to meet the deadline, which implies that $\tau_{2,1}^1$ has a deadline at time one.

Motivated by the above observation, a better alternative is to set fixed relative deadlines for each subtask. An FRD policy is to set relative deadlines $D_{i,1}$ and $D_{i,2}$ for the executions of the first subtask and the second subtask of τ_i, respectively. When a job of task τ_i arrives at time t,

- the release time and the absolute deadline of the first subjob (i.e., the first computation phase) are t and $t+D_{i,1}$, respectively,
- the suspension has to be finished before $t + D_{i,1} + S_i$,
- the release time and the absolute deadline of the second subjob (i.e., the second computation phase) is $t+D_{i,1}+S_i$ and $t + D_{i,1} + S_i + D_{i,2}$, respectively.

For the rest of this paper, we call such a scheduling policy a *fixed-relative-deadline scheduler* (FRD scheduler). After the relative deadlines are assigned, we will use EDF to schedule the subjobs by using dynamic-priority scheduling.

The fixed-relative-deadline scheduling has a feasible schedule if the worst-case response time of the first (second, respectively) computation phase 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 $D_{i,1} + D_{i,2} + S_i \leq T_i$.

Theorem 1. *For a given task set* $\mathbf{T}$*, deciding whether there exists a relative-deadline assignment for fixed-relative-deadline scheduling is* $\mathcal{NP}$*-complete in the strong sense.*

Proof: The reduction, from the 3-PARTITION problem, is the same as the proof for the preemptive case in Theorem 1 in [20]. We would also like to point out a minor issue in the proof for the only-if part of the preemptive case in Theorem 1 in [20]. Here, we use the same terminologies in [20] to patch the proof. The proof does not consider a possible case, in which four subtasks start in one block k but only 3 subtasks are completed in this block. In such a case, in the block $k+m$, the workload that can be processed is strictly less than the block size B. Then, the contradiction in the proof for the only-if statement remains. ■

3.2 Resource Augmentation Factor

Due to Theorem 1, a common approach for quantifying the quality of scheduling algorithms and schedulability tests is to quantitatively bound the degree to which the algorithm under consideration may under-perform a hypothetical optimal one. To obtain such a bound, we adopt the concept of the resource augmentation factor [2], [18]. When the speed of the system is 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, S_i remains the same.

Typically, the resource augmentation factor is defined, by referring to any arbitrarily feasible schedule:

- **Scheduling algorithm with respect to arbitrary schedules**: For notational brevity, we call such a factor the *arbitrary speed-up factor*. Provided that the task set $\mathbf{T}$ can be feasibly scheduled, an algorithm $\mathcal{A}$ is said to have an α arbitrary speed-up factor when algorithm $\mathcal{A}$ guarantees to derive a feasible schedule by speeding up the system with a factor α.
- **Schedulability test with respect to arbitrary schedules**: A schedulability test is with an α arbitrary speed-up factor for a scheduling algorithm $\mathcal{A}$: if the test fails, i.e., the test returns "infeasibility", then the task set is also not schedulable (under any scheduling policy) by slowing to run at speed $\frac{1}{\alpha}$.

Moreover, the above definition can also be extended by referring to the optimal fixed-relative deadline schedules:

- **Scheduling algorithm with respect to FRD schedules**: For notational brevity, we call such a factor the *FRD* *speed-up factor*. Provided that the task set $\mathbf{T}$ can be feasibly scheduled under a fixed-relative-deadline schedule, an algorithm $\mathcal{A}$ is said to have an α FRD speed-up factor when algorithm $\mathcal{A}$ guarantees to derive a feasible fixed-relative-deadline schedule by speeding up the system with a factor α.

- **Schedulability tests with respect to FRD schedules**: A schedulability test is with an α FRD speed-up factor for an FRD scheduling algorithm $\mathcal{A}$: if the test fails, i.e., the test returns "infeasibility", then the task set is also not schedulable under any FRD schedules by slowing to run at speed $\frac{1}{\alpha}$.

3.3 Speed-Up Factors of EDF and RM

Classical priority-based dynamic scheduling algorithms such as EDF and RM are able to deliver good performance on a uniprocessor for ordinary real-time task systems without self-suspensions (e.g., EDF is optimal on a uniprocessor). Unfortunately, the following theorem shows that these algorithms yield rather poor performance in the presence of self-suspensions w.r.t. speed-up factors if we do not set individual relative deadlines for the computation phases.

Theorem 2. *The arbitrary and the FRD speed-up factors under EDF and RM are ∞.*

Proof: The proof is similar to the proof of Theorem 6 in [20] with minor changes. The details are in the Appendix. ■

4 Necessary Conditions for Schedulability

This section presents the necessary conditions for the schedulability of FRD schedules and any arbitrary schedules. The following lemma gives the necessary condition of any FRD schedule (by considering the worst-case job arrivals).

Lemma 1. *No matter how $D_{i,1}$ and $D_{i,2}$ are assigned under the condition $D_{i,1} + D_{i,2} \leq T_i - S_i$, the necessary condition for the schedulability of any FRD scheduler is*

$$\forall t > 0, \qquad \sum_{\tau_i \in \mathbf{T}} dbf_i(t) \leq t, \tag{1}$$

where

$$dbf_i(t) = \begin{cases} 0 & 0 \leq t < T_i - S_i \\ (C_{i,1} + C_{i,2}) & t \geq T_i - S_i \\ + \left\lfloor \frac{t-(T_i-S_i)}{T_i} \right\rfloor (C_{i,1} + C_{i,2}) & \end{cases} \tag{2}$$

Proof: This lemma is proved in sketch by evaluating the computation demand incurred by the FRD scheduler in any given time interval length t. It is not difficult to prove that we only have to consider the case that $D_{i,1} + D_{i,2} = T_i - S_i$ for evaluating the necessary condition for the schedulability of the FRD scheduling policy.

Now, let us consider a fixed time instant x. Assume that the first job of task τ_i is released at time $x-(D_{i,1}+S_i)$ and all the corresponding jobs are released as soon as the minimum inter-arrival time constraint is met. Therefore, we know that task τ_i has to finish $dbf_i(t)$ in time interval $(x, t+x]$ for any $t > 0$. As a result, the necessary condition is proved. ■

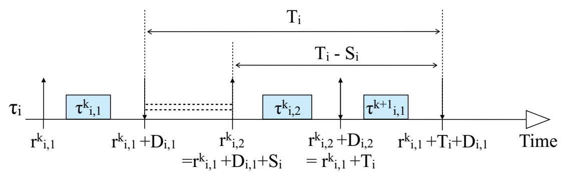


Fig. 3: An illustration of (2) as a necessary condition for the schedulability under any FRD scheduling policy.

$dbf_i(t)$	necessary condition for a feasible FDR schedule (used as a lower bound)	Equation (2)
$dbf_i^*(t)$	necessary condition for any feasible schedule (used as a lower bound)	Equation (4)
$dbf_i^{EDA}(t)$	necessary and sufficient condition of Algorithm EDA (used as an upper bound)	Equation (7)
$dbf_i^{EDA^\dagger}(t)$	sufficient condition of Algorithm EDA with a linear approximation (used as an upper bound)	Equation (11)
$dbf_i^{EDA^\flat}(t)$	transformation of $dbf_i^{EDA\dagger}(t)$ for analyzing the speed-up factors of the linear approximation	Equation (14)

TABLE I: Variant symbols related to the demand bound functions, where EDA is an FRD algorithm studied in Sec. 5.

Fig. 3 illustrates how the demand bound function in (2) is defined for any FRD scheduler. Next we provide a necessary condition for scheduability under any scheduling policy.

Lemma 2. *The necessary condition for the schedulability under any scheduling policy is*

$$\forall t > 0, \qquad \sum_{\tau_i \in \mathbf{T}} dbf_i^*(t) \leq t. \tag{3}$$

where

$$dbf_i^*(t) = \begin{cases} 0 & 0 \leq t < T_i - S_i \\ C_{i,\max} + & \\ \left\lfloor \frac{t-(T_i-S_i)}{T_i} \right\rfloor (C_{i,1} + C_{i,2}) & t \geq T_i - S_i \end{cases} \tag{4}$$

Proof: The proof is similar to the proof of Lemma 1, but has to consider the arbitrary scheduling policy. Let $\mathbf{T}_1$ be the tasks in $\mathbf{T}$ in which $C_{i,1} > C_{i,2}$ and $\mathbf{T}_2$ be $\mathbf{T} \setminus \mathbf{T}_1$.

Now, let's fix a time instant x. For each task τ_i, let the first job of task τ_i be released at time x if τ_i is in $\mathbf{T}_1$, and at time $x - S_i - C_{i,1}$ if τ_i is in $\mathbf{T}_2$. Moreover, all the corresponding jobs are released as soon as the minimum inter-arrival time constraint is met. Therefore, we know that task τ_i has to finish at least $dbf_i^*(t)$ in time interval $(x, t+x]$ for any $t > 0$. As a result, the necessary condition is proved. ■

Therefore, the functions $dbf_i(t)$ and $dbf_i^*(t)$ provide the lower bounds for feasible FRD and arbitrary schedules. According to Lemma 1, if $\max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i(t)}{t} > f$, the task set $\mathbf{T}$ cannot be feasibly scheduled by any FRD schedule if the speed is lower than or equal to f. Similarly, due to Lemma 2, if $\max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(t)}{t} > f'$, $\mathbf{T}$ cannot be feasibly scheduled by any schedule if the speed is lower than or equal to f'.

Notations Related to DBF The rest of this paper will use several definitions related to the demand bound functions that represent the *minimum* or the *maximum* demand that has to be finished within a specified length t of time intervals. To explain the difference among these terms, we provide a table (in Tab. I) and an illustrative summary (in Fig. 4) here.

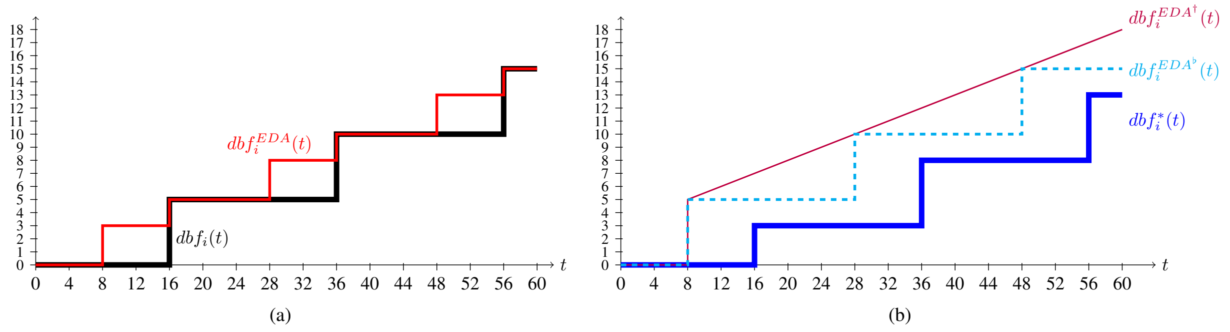


Fig. 4: An example of the listed functions in Tab. I, when $C_{i,1} = 3$, $C_{i,2} = 2$, $S_i = 4$, and $T_i = 20$. For clarity, two sub-figures are presented to avoid overlapping of the curves.

5 FRD Scheduling Algorithms

In this section, we first present an equal-deadline assignment (EDA) algorithm that assigns equal relative deadlines to subtasks of each task and all subjobs are scheduled by EDF. We then derive a pseudo-polynomial-time schedulability test for this algorithm and prove corresponding non-trivial resource-augmentation performance guarantees. At the end of this section, we also explain why another greedy approach, used in [16], by assigning the relative deadline proportionally to the execution time is indeed a bad approach by providing an example.

5.1 Algorithm EDA

For each task τ_i, $D_{i,1} + D_{i,2} \leq T_i - S_i$ must hold. EDA makes the following assignment

$$D_{i,1} = D_{i,2} = \frac{T_i - S_i}{2}. \tag{5}$$

As $D_{i,1} = D_{i,2}$, for notational brevity, we denote $D_{i,1}$ and $D_{i,2}$ as Δ_i.

The following theorem gives a schedulability test for EDA that can be checked in pseudo-polynomial time.

Theorem 3. *Algorithm EDA generates a feasible fixed-relative-deadline schedule for the input task set* $\mathbf{T}$ *if and only if*

$$\forall t > 0, \qquad \sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA}(t) \leq t, \tag{6}$$

where

$$dbf_i^{EDA}(t) = \begin{cases} 0 & 0 \leq t < \frac{T_i - S_i}{2} \\ C_{i,\max} & \frac{T_i-S_i}{2} \leq t < T_i - S_i \\ C_{i,1} + C_{i,2} & t = T_i - S_i \\ dbf_i^{EDA}(t - \left\lfloor \frac{t-(T_i-S_i)}{T_i} \right\rfloor T_i) + \\ (\left\lfloor \frac{t-(T_i-S_i)}{T_i} \right\rfloor + 1)(C_{i,1} + C_{i,2}) & t > T_i - S_i. \end{cases} \tag{7}$$

Proof: Due to space limitation, we only provide the concepts behind the test. The function described in (7) is basically similar to that in (2) except the ranges $t \in [\frac{T_i-S_i}{2} + v \cdot T_i, T_i - S_i + v \cdot T_i)$ for any non-negative integer v, e.g., in Fig. 4a. Such a range is due to the fact that EDA assigns these two subtasks of task τ_i with the same relative deadline $\frac{T_i-S_i}{2}$, in which one of them requires $C_{i,\max}$ amount of time for execution and the other requires $C_{i,\min}$. For task τ_i in any interval length in $[\frac{T_i-S_i}{2} + v \cdot T_i, T_i - S_i + v \cdot T_i)$, in the worst case, there are $v+1$ subjobs with execution time $C_{i,\max}$ and v subjobs with execution time $C_{i,\min}$. These subjobs of task τ_i do not have any overlap in their release times and absolute deadlines. Therefore, the if and only-if conditions can be proved by using the same strategy to prove the exact schedulability test for the ordinary sporadic real-time tasks in [3]. ∎

Theorem 3 also implies that the condition in (6) is an exact schedulability test, that requires pseudo-polynomial time, of Algorithm EDA.

5.2 Quantitative Evaluation of EDA

We now offer a quantitative evaluation of the efficacy of Algorithm EDA. Specifically, we derive properties that are used to provide a quantitative measure w.r.t. resource augmentation factors of how effective Algorithm EDA is compared to an optimal fixed-realative-deadline scheduler (Theorem 4) and an optimal arbitrary scheduler (Theorem 5).

Lemma 3. *For any t for a task $\tau_i \in \mathbf{T}$, we have*

$$dbf_i^{EDA}(t) - dbf_i(t) \leq C_{i,\max}$$

and $$dbf_i^{EDA}(t) - dbf_i^*(t) \leq C_{i,\max}.$$

Lemma 4. *For any $t \geq 0$ for a task $\tau_i \in \mathbf{T}$, we have*

$$dbf_i(2t) \geq dbf_i^{EDA}(t).$$

Proof: The proofs of Lemma 3 and Lemma 4 are by simple arithmetics and the definitions of these three functions. Specifically, the proof of Lemma 4 is in the Appendix. ∎

Both of the above properties can be also found in Fig. 4.

Theorem 4. *The FRD speed-up factor of Algorithm EDA is 2.*

Proof: Suppose that at time $t^* > 0$, we have $\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA}(t^*) > t^*$. Let us now classify the tasks in $\mathbf{T}$ into three task sets:

- $\mathbf{T}_1$: if $t^* < \frac{T_i-S_i}{2}$, task τ_i is in $\mathbf{T}_1$.
- $\mathbf{T}_2$: if $T_i - S_i > t^* \geq \frac{T_i-S_i}{2}$, task τ_i is in $\mathbf{T}_2$.

- $\mathbf{T}_3$: if $t^* \geq T_i - S_i$, task τ_i is in $\mathbf{T}_3$.

Clearly, each task τ_i in $\mathbf{T}$ is either in $\mathbf{T}_1, \mathbf{T}_2$, or $\mathbf{T}_3$. Now suppose that $\frac{\sum_{\tau_i \in \mathbf{T}_2} dbf_i^{EDA}(t^*)}{t^*}$ is x and $\frac{\sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*)}{t^*}$ is y. Since $\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA}(t^*) > t^*$, we have

$$x + y > 1.$$

By Lemma 4, we have

$$\sum_{\tau_i \in \mathbf{T}_2 \cup \mathbf{T}_3} dbf_i^{EDA}(t^*) \leq \sum_{\tau_i \in \mathbf{T}_2 \cup \mathbf{T}_3} dbf_i(2t^*). \tag{8}$$

Therefore, by (8), we have

$$\begin{aligned} \max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i(t)}{t} &\geq \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i(2t^*)}{2t^*} \\ &\geq \frac{\sum_{\tau_i \in \mathbf{T}_2 \cup \mathbf{T}_3} dbf_i(2t^*)}{2t^*} \\ &\geq \frac{\sum_{\tau_i \in \mathbf{T}_2 \cup \mathbf{T}_3} dbf_i^{EDA}(t^*)}{2t^*} \\ &= \frac{x+y}{2} > 0.5. \end{aligned} \tag{9}$$

Therefore, this implies that the task set $\mathbf{T}$ cannot be feasibly scheduled by any fixed-relative-deadline schedule when the speed of the system is 0.5. ■

With a similar proof, we can also show the arbitrary speed-up factor for EDA.

Theorem 5. *The arbitrary speed-up factor of Algorithm EDA is* 3.

Proof: The proof strategy is similar to the proof of Theorem 4. The major difference in the analysis is to further consider a subset of tasks in which $\frac{3}{2}T_i - \frac{1}{2}S_i > t^* \geq T_i - S_i$ and a property revised from Lemma 4 for function $dbf_i^*(t)$. The detailed proof is in the Appendix. ■

5.3 Algorithm Proportional: A Bad Approach

It may seem to be very pessimistic to simply use EDA. Another greedy approach is to assign the relative deadline proportionally to the execution time, i.e., $D_{i,1} = \frac{C_{i,1}}{C_{i,1}+C_{i,2}} \cdot (T_i - S_i)$, as used in [16]. However, this approach can be shown to be quite bad in the worst cases by using the following example: We are given n tasks, in which $C_{i,1} = 1$, $C_{i,2} = 2^i - 1$, $T_i - S_i = 2^i * (n-1)$, and $T_i \gg 2^n$ for $i = 1, 2, \ldots, n$. By the proportional relative deadline algorithm, $D_{i,1}$ is set to $n-1$, and $D_{i,2}$ is set to $(2^i - 1) * (n-1)$ for each task τ_i.

The demand bound function of the above algorithm at time $n-1$ is n. Therefore, the above task set is not schedulable by EDF under the proportional relative deadline assignment. By evaluating $\sum_{i=1}^n dbf_i^{EDA}(t)$, we can conclude that $\sum_{i=1}^n dbf_i^{EDA}(t) \leq \frac{4t}{n-1}$. Therefore, the above task set remains schedulable under Algorithm EDA even when the speed of the system is $\frac{4}{n-1}$. As a result, the speed-up factor of the proportional relative deadline algorithm is at least $\Omega(n)$, which is quite bad.

6 Linear-Time Schedulability Test

Sec. 5 presents EDA and schedulability analysis in pseudo polynomial-time. This section further presents a linear-time schedulability test, under the assumption that the sorting of the tasks according to $T_i - S_i$ is done in advance.

6.1 Density-Based Schedulability Analysis

By simple arithmetics, it can be easily shown that $\frac{dbf_i^{EDA}(t)}{t} \leq \frac{C_{i,\max}}{(T_i - S_i)/2}$ for any t. Therefore, to achieve a fast schedulability analysis, if $\sum_{\tau_i \in \mathbf{T}} \frac{2C_{i,\max}}{T_i - S_i} \leq 1$, EDA provides a feasible schedule under EDF. However, such a schedulability test $\sum_{\tau_i \in \mathbf{T}} \frac{2C_{i,\max}}{T_i - S_i} \leq 1$ can be very pessimistic.

Theorem 6. *The FRD speed-up factor of the schedulability test by verifying whether $\sum_{\tau_i \in \mathbf{T}} \frac{2C_{i,\max}}{T_i - S_i} \leq 1$ holds for n tasks is at least H_n, where $H_n = \sum_{i=1}^n \frac{1}{i}$.*

Proof: To prove such a lower bound, we just have to provide a concrete case. Consider a special input instance with n tasks as follows:

- $C_{i,1} = C_{i,2} = \frac{1}{f}$, $S_i = 2n - 2i + 1$ and $T_i = 2n + 1$, for task τ_i,

in which f is $\frac{H_n}{1+\epsilon}$ with $\epsilon > 0$. For this input instance, we have $\sum_{\tau_i \in \mathbf{T}} \frac{2}{f \cdot 2i} = \frac{1}{f} \sum_{i=1}^n \frac{1}{i} = 1 + \epsilon > 1$. The schedulability test here shows that this input instance is not schedulable under EDA. However, it is very clear that the resulting solution by EDA is feasible even if the system is slowed down to run at speed $\frac{1}{f}$. ■

6.2 Linear Approximation

We now present a linear-time schedulability test for Algorithm EDA and analyze the speed-up factors for the schedulability test. The key idea is to use a linear curve to (upper) bound the function $dbf_i^{EDA}(t)$ for all t, which has been used in [4]. When the linear curve is too far away from the function $dbf_i^{EDA}(t)$, the error becomes too large. Therefore, the design philosophy is to minimize the gap between the linear curve and the function $dbf_i^{EDA}(t)$.

The increase of the function $dbf_i^{EDA}(t)$ becomes periodic when t is large enough, i.e., $dbf_i^{EDA}(t) = dbf_i^{EDA}(t - T_i) + C_{i,1} + C_{i,2}, \forall t > 2T_i - S_i$. Therefore, we should use the utilization $U_i = \frac{C_{i,1}+C_{i,2}}{T_i}$ as the slope of the linear approximation, so that the approximation becomes more precise when t is large enough. That is, this linear approximation should be able to bound the function $dbf_i^{EDA}(t)$, under the slope U_i.

To provide a safe upper bound of the function $dbf_i^{EDA}(t)$ at any interval length t, we can start with a linear segment at interval length Δ_i, which is defined as $\frac{T_i - S_i}{2}$, with a value $C_i' = \max\{C_{i,\max}, C_{i,1} + C_{i,2} - U_i \Delta_i\}$. That is, this segment is with a slope equals to U_i. The above setting of C_i' comes from two different specified values in the linear segment: (1) $C_{i,\max}$ at length Δ_i or (2) $C_{i,1} + C_{i,2}$ at length $2\Delta_i$ (which implies that the segment starts with $C_{i,1} + C_{i,2} - U_i \Delta_i$ at length Δ_i).

Since $\max\{C_{i,\max}, C_{i,1} + C_{i,2} - U_i \Delta_i\} \leq C_{i,1} + C_{i,2}$, setting C_i' to $C_{i,1} + C_{i,2}$ with a slope U_i, starting at length

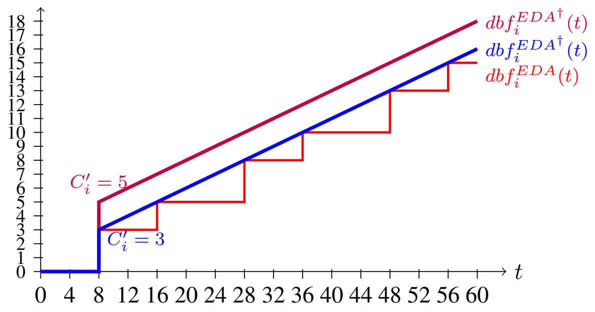


Fig. 5: An example of the linear approximation by using different C' values, where $C_{i,1} = 3$, $C_{i,2} = 2$, $S_i = 4$, and $T_i = 20$. Therefore, $C_{i,1} + C_{i,2} = 5$ and $\max\{C_{i,\max}, C_{i,1} + C_{i,2} - U_i\Delta_i\}$ is 3.

Δ_i, is also a feasible upper bound of $dbf^{EDA^\dagger}(t)$. Fig. 5 presents the above two different linear approximations. For the simplicity of presentation for the analysis in Sec. 6.4, we will consider that C_i' is equal to $C_{i,1} + C_{i,2}$ for the rest of this section, while the experimental results in Sec. 7 will be based on $C_i' = \max\{C_{i,\max}, C_{i,1} + C_{i,2} - U_i\Delta_i\}$.

Therefore, for the rest of this section, suppose that[1]

$$C_i' \overset{\text{def}}{=} C_{i,1} + C_{i,2}. \tag{10}$$

Moreover, the linear approximation is

$$dbf_i^{EDA^\dagger}(t) = \begin{cases} 0 & 0 \le t < \Delta_i \\ C_i' + (t - \Delta_i)U_i & \Delta_i \le t. \end{cases} \tag{11}$$

Then, we have the following lemma.

Lemma 5. *For any given $t > 0$ and $\tau_i \in \mathbf{T}$,*

$$dbf_i^{EDA^\dagger}(t) \ge dbf_i^{EDA}(t).$$

Proof: This is based on simple arithmetics, as also shown in Fig. 4. ■

Theorem 7. *Algorithm EDA generates a feasible fixed-relative-deadline schedule for the input task set if*

$$\forall t > 0, \qquad \sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA^\dagger}(t) \le t. \tag{12}$$

Proof: This comes directly from Lemma 5 and Theorem 3. ■

For notational brevity, for the rest of this section, we order the tasks such that $T_1 - S_1 \le T_2 - S_2 \le \ldots \le T_n - S_n$. That is, $\Delta_i \le \Delta_{i+1}$ for $i = 1, 2, \ldots, n-1$.

Theorem 8. *Suppose that $\Delta_i \le \Delta_{i+1}$ for $i = 1, 2, \ldots, n-1$. Algorithm EDA generates a feasible fixed-relative-deadline schedule for the input task set if $U_{sum} = \sum_{i=1}^n U_i \le 1$ and*

$$\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) \le \Delta_\ell, \qquad \forall \ell = 1, 2, \ldots, n. \tag{13}$$

Proof: It is clear that $\sum_{i=1}^n U_i \le 1$ is a necessary condition for feasible schedules. For notational brevity, let Δ_{n+1} be ∞. With $\sum_{i=1}^n U_i \le 1$ and (13), for any $\Delta_\ell < t \le \Delta_{\ell+1}$ and $\ell = 1, 2, \ldots, n$, we know that

$$\sum_{i=1}^{n} dbf_i^{EDA^\dagger}(t) = \sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) + (t - \Delta_\ell)\sum_{i=1}^{\ell} U_i$$
$$\le \Delta_\ell + t - \Delta_\ell = t.$$

Therefore, by testing only these n points in (13) under the assumption that $\sum_{i=1}^n U_i \le 1$, we know that the condition in (12) in Theorem 7 holds for all $t > 0$, which concludes the proof. ■

[1]We will use the term C_i' to make distinguishes to C_i.

The following corollary provides the time complexity analysis for the schedulability test.

Corollary 1. *The schedulability test in Theorem 8 can be done in linear time provided that the tasks are ordered in a non-decreasing order with respect to Δ_i.*

Proof: The test can be done by incrementally evaluating $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell)$ for $\ell = 1, 2, \ldots, n$. The details are provided in the Appendix. ■

Note that the above linear approximation, as presented in Fig. 5, has one jump at time Δ_i and has a constant slope after time Δ_i. *Such a type of linear approximation has been used to partition sporadic real-time tasks to identical multiprocessor systems [2], [4], [5]. The same task partitioning strategy in [2], [4], [5] can be used to assign the self-suspended tasks to multiprocessor systems by adopting the linear approximation here for testing whether it is feasible to assign a self-suspended task onto a processor.* The details of the analysis are not presented here due to the space limitations.

For the rest of this section, we will prove the arbitrary and FRD speed-up factors of the linear approximation in the schedulability test in Theorem 8. We denote the schedulability test in Theorem 8 as *schedulability analysis (test) LA*.

6.3 Speed-Up Factor Analysis: Basic Analyses

Here, we will first provide the basic analysis by using a simpler proof strategy. We define the following step function $dbf_i^{EDA^\flat}(t)$ to represent the lower bound of the linear approximation $dbf_i^{EDA^\dagger}(t)$.

$$dbf_i^{EDA^\flat}(t) = \begin{cases} 0 & 0 \le t < \Delta_i \\ C_i' + \left(\left\lfloor \frac{t-\Delta_i}{T_i} \right\rfloor (C_{i,1} + C_{i,2})\right) & \Delta_i \le t \end{cases} \tag{14}$$

By the above definition, we have the following lemma:

Lemma 6. *For any given $t > 0$ and $\tau_i \in \mathbf{T}$,*

$$dbf_i^{EDA^\flat}(t) \le dbf_i(2t). \tag{15}$$

Proof: It clearly holds when $t < \Delta_i$, since $dbf_i^{EDA^\flat}(t) = 0$. By the definition of C_i', we have $C_i' \le C_{i,1} + C_{i,2}$. Moreover, with the definition of $dbf_i(t)$, we know that $dbf_i^{EDA^\flat}(t) \le dbf_i(t + \Delta_i)$. (These two inequalities happen when C_i' is set to $\max\{C_{i,\max}, C_{i,1} + C_{i,2} - U_i\Delta_i\}$ instead of $C_{i,1} + C_{i,2}$.) Therefore, when $t \ge \Delta_i$, we have $dbf_i^{EDA^\flat}(t) \le dbf_i(2t)$. ■

We can now prove the FRD speed-up factor for the schedulability analysis in Theorem 8.

Theorem 9. *The FRD speed-up factor for the schedulability analysis LA (in Theorem 8) is* 3.

Proof: The schedulability test in Theorem 8 can fail by two cases: (1) $\sum_{i=1}^{n} U_i > 1$ or (2) $\exists \ell$ such that $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) > \Delta_\ell$. For the former case, it is clear that the task set is not schedulable for any scheduling policy. To prove the FRD speed-up factor, we only have to focus on the latter case, in which the error comes from the linear approximation as well as the scheduling policy EDA.

Suppose that ℓ is the smallest index such that $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) > \Delta_\ell$. For $i = 1, 2, \ldots, \ell$, let δ_i be $\left\lfloor \frac{\Delta_\ell - \Delta_i}{T_i} \right\rfloor T_i + \Delta_i$. That is, δ_i is set such that $\left\lfloor \frac{\Delta_\ell - \Delta_i}{T_i} \right\rfloor$ is equal to $\frac{\delta_i - \Delta_i}{T_i}$. As a result, $dbf_i^{EDA^\dagger}(\Delta_\ell)$ is equal to $dbf_i^{EDA^\flat}(\delta_i) + U_i \cdot (\Delta_\ell - \delta_i)$. Therefore, we have

$$\begin{aligned}
\Delta_\ell &< \sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) \\
&= \left(\sum_{i=1}^{\ell} dbf_i^{EDA^\flat}(\delta_i)\right) + \left(\sum_{i=1}^{\ell} U_i \cdot (\Delta_\ell - \delta_i)\right) \\
&\leq_1 \left(\sum_{i=1}^{\ell} dbf_i^{EDA^\flat}(\Delta_\ell)\right) + (\sum_{i=1}^{\ell} U_i \Delta_\ell) \\
&\leq_2 \left(\sum_{i=1}^{\ell} dbf_i(2\Delta_\ell)\right) + (\sum_{i=1}^{\ell} U_i \Delta_\ell),
\end{aligned}$$

where $\leq_1$ comes from the fact that $\delta_i \leq \Delta_\ell$ and $dbf_i^{EDA^\flat}(t)$ is a non-decreasing function of t for each $i < \ell$, and $\leq_2$ comes from Lemma 6.

By dividing Δ_ℓ in both sides, we have

$$1 < 2\left(\frac{\sum_{i=1}^{\ell} dbf_i(2\Delta_\ell)}{2\Delta_\ell}\right) + (\sum_{i=1}^{\ell} U_i).$$

Therefore, either $\sum_{i=1}^{\ell} U_i > \frac{1}{3}$ or $\frac{\sum_{i=1}^{\ell} dbf_i(2\Delta_\ell)}{2\Delta_\ell} > \frac{1}{3}$. This also implies that, by slowing down the system to speed $\frac{1}{3}$, there does not exist any feasible FRD schedule since the necessary condition in Lemma 1 cannot be satisfied. Therefore, we reach the conclusion. ■

Similar to the proofs in Lemma 6 and Theorem 9, the following lemma and theorem can be achieved by using a very similar strategy.

Lemma 7. *For any given $t > 0$ and $\tau_i \in \mathbf{T}$,*

$$dbf_i^{EDA^\flat}(t) \leq 2dbf_i^*(2t). \tag{16}$$

Proof: It clearly holds when $t < \Delta_i$, since $dbf_i^{EDA^\flat}(t) = 0$. By the definition of C_i' in (10), we have $C_i' \leq C_{i,1} + C_{i,2}$. Moreover, with the definition of $dbf_i^*(t)$, we know that $dbf_i^{EDA^\flat}(t) \leq dbf_i^*(t+\Delta_i) + C_{i,\min}$. Therefore, when $t \geq \Delta_i$, we have $dbf_i^{EDA^\flat}(t) \leq dbf_i^*(2t) + C_{i,\min} \leq 2dbf_i^*(2t)$. ■

Theorem 10. *The arbitrary speed-up factor for the schedulability analysis LA (in Theorem 8) is* 5.

Proof: By using the same strategy in the proof in Theorem 9 by using $dbf_i^*()$ instead of $dbf_i()$, we reach

$$\Delta_\ell < 2\left(\sum_{i=1}^{\ell} dbf_i^*(2\Delta_\ell)\right) + (\sum_{i=1}^{\ell} U_i \Delta_\ell).$$

By dividing Δ_ℓ in both sides, we have

$$1 < 4\left(\frac{\sum_{i=1}^{\ell} dbf_i^*(2\Delta_\ell)}{2\Delta_\ell}\right) + (\sum_{i=1}^{\ell} U_i).$$

Therefore, either $\sum_{i=1}^{\ell} U_i > \frac{1}{5}$ or $\frac{\sum_{i=1}^{\ell} dbf_i^*(2\Delta_\ell)}{2\Delta_\ell} > \frac{1}{5}$, which concludes the proof. ■

6.4 Speed-Up Factor Analysis: Tighter Analysis

Following the proofs in Theorems 9 and 10, we focus ourselves to provide tighter analysis in this subsection. The analysis extends the analysis developed in [5] for the speed up factor of the linear approximate demand bound function for ordinary sporadic real-time tasks (without self-suspensions). With the setting of $C_i' = C_{i,1} + C_{i,2}$ in (10), the connection between the two functions $dbf_i^{EDA^\flat}(t)$ and $dbf_i^{EDA^\dagger}(t)$ is identical to the linear approximation when considering normal sporadic tasks in [5].

Now, let us look at the linear approximation in (11) and the necessary condition based on $dbf_i()$ in Lemma 1 more closely. Based on Lemma 5, we can observe the following theorem.

Theorem 11. *The FRD speed-up factor for the schedulability analysis LA (in Theorem 8) is $\frac{2}{\alpha}$ if there exists $t > 0$ with $\frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA^\flat}(t)}{t} > \alpha$ or $\sum_{\tau_i \in \mathbf{T}} U_i > 0.5\alpha$.*

Proof: By Lemma 6, the condition the existence of t with $\frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA^\flat}(t)}{t} > \alpha$ implies that $\frac{\sum_{\tau_i \in \mathbf{T}} dbf_i(2t)}{2t} > \frac{\alpha}{2}$. Therefore, if such a condition holds or $\sum_{\tau_i \in \mathbf{T}} U_i > 0.5\alpha$, by definition, the FRD speed-up factor for the schedulability analysis LA is $\frac{2}{\alpha}$. ■

For the rest of this subsection, let ℓ be the smallest index such that $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) > \Delta_\ell$. Moreover, δ_i is defined as $\left\lfloor \frac{t - \Delta_i}{T_i} \right\rfloor T_i + \Delta_i$, as in the proof of Theorem 9.

The key difference between the tighter analyses here and those in Sec. 6.3 is due to the range of t for evaluating $\frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA^\flat}(t)}{t}$. In the proofs of Theorems 9 and 10, we only evaluate one single value, in which $t = \Delta_\ell$, whereas the tighter analyses here will adopt Theorem 11 and evaluate the maximum value of $\frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA^\flat}(t)}{t}$ for $0 < t \leq \Delta_\ell$.

For notational simplicity, we define $\beta, k^\dagger$, and k as follows:

$$C'_\ell = \beta \sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i), \tag{17}$$

$$\sum_{i=1}^{\ell-1} (\Delta_\ell - \delta_i) U_i = k^\dagger \sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i). \tag{18}$$

$$k \sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i) = \Delta_\ell - (1+\beta)\sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i). \tag{19}$$

That is, by taking $\sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i)$ as the basis, β defines the ratio of C'_ℓ to $\sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i)$, $k^\dagger$ defines the ratio of the partially released workload $\sum_{i=1}^{\ell-1} (\Delta_\ell - \delta_i) U_i$ in time interval length Δ_ℓ with respect to $\sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i)$, and k is defined to set $\Delta_\ell = (1+k+\beta)\sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i)$.

Based on the above definition and the fact that $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) > \Delta_\ell$, we know that $k^\dagger > k$ and

$$\begin{aligned} \max_{t\geq 0} \frac{\sum_{i=1}^{n} dbf_i^{EDA^\flat}(t)}{t} &\geq \frac{\sum_{i=1}^{\ell} dbf_i^{EDA^\flat}(\Delta_\ell)}{\Delta_\ell} \\ &= \frac{1+\beta}{1+k+\beta}. \end{aligned} \tag{20}$$

Lemma 8. *If* $k \leq \frac{(e^{0.5}-1)}{e^{0.5}}(1+\beta)$*, then*

$$\max\left\{\max_{t>0} \frac{\sum_{i=1}^{\ell} dbf_i(t)}{t}, \sum_{i=1}^{\ell-1} U_i\right\} > \frac{1+\beta}{2(1+k+\beta)}.$$

Proof: This comes directly from the above analysis in (20). ■

Therefore, when k is small, the bound in Lemma 8 can be used. The following analysis moves further by considering larger k, in which the corresponding proofs are very similar to the proofs in [5] and the major differences are provided in the Appendix, for completeness.

Lemma 9. *For any non-negative parameters,* k, β, x, α*, if* $x^* \leq \frac{\alpha(1+k+\beta)(e^{0.5}-1)}{e^{0.5}}$*, then* $\int_0^{x^*} \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx \leq \frac{\alpha}{2}$.

Lemma 10. *Suppose that* ℓ *is the smallest index such that* $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell) > \Delta_\ell$ *and* α *is* $\frac{k+\beta}{1+k+\beta}\frac{(e^{0.5})}{(e^{0.5}-1)}$*. If, for all* $0 < t \leq \Delta_\ell$*, we have* $\frac{\sum_{i=1}^{\ell} dbf_i^{EDA^\flat}(t)}{t} \leq \alpha$*, then, the total utilization* $\sum_{i=1}^{\ell-1} U_i$ *for* $\tau_1, \tau_2, \ldots, \tau_{\ell-1}$ *is larger than* $\int_0^k \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx = \frac{\alpha}{2}$.

Lemma 11. *If* $k > \frac{(e^{0.5}-1)}{e^{0.5}}(1+\beta)$*, then*

$$\max\left\{\max_{t>0} \frac{\sum_{i=1}^{\ell} dbf_i(t)}{t}, \sum_{i=1}^{\ell-1} U_i\right\} > \frac{\frac{e^{0.5}}{(e^{0.5}-1)}k}{2(1+k+\beta)}. \tag{21}$$

We can now conclude the analysis by providing the corresponding speed-up factors for the schedulability analysis in Theorem 8.

Theorem 12. *The FRD speed-up factor for the schedulability analysis LA (in Theorem 8) is* $\frac{2(2e^{0.5}-1)}{e^{0.5}} < 2.787$.

Proof: By Lemma 8, the function $\frac{1+\beta}{2(1+k+\beta)}$ is a decreasing function with respect to k when $k \geq 0$. By Lemma 11, the function $\frac{\frac{e^{0.5}}{(e^{0.5}-1)}k}{2(1+k+\beta)}$ is an increasing function with respect to k. Therefore, the only intersection when k is equal to $\frac{e^{0.5}-1}{e^{0.5}}(1+\beta)$ defines which part should be used for bounding the speed-up factor.

There are two cases:

- When $k \leq \frac{e^{0.5}-1}{e^{0.5}}(1+\beta)$, by using Lemma 8, we know that the FRD speed-up factor is at most $2 \cdot \frac{\frac{e^{0.5}-1}{e^{0.5}}(1+\beta)}{1+\beta} = \frac{2(2e^{0.5}-1)}{e^{0.5}}$.
- When $k > \frac{e^{0.5}-1}{e^{0.5}}(1+\beta)$, by using Lemma 11, we know that the FRD speed-up factor is at most $2 \cdot \frac{\frac{e^{0.5}-1}{e^{0.5}}(1+\beta)}{1+\beta} = \frac{2(2e^{0.5}-1)}{e^{0.5}}$.

Therefore, the FRD speed-up factor for the schedulability analysis in Theorem 8 is $\frac{2(2e^{0.5}-1)}{e^{0.5}} < 2.787$. ■

Theorem 13. *The arbitrary speed-up factor for the schedulability analysis in Theorem 8 is* $\frac{4(2e^{0.25}-1)}{e^{0.25}} < 4.875$.

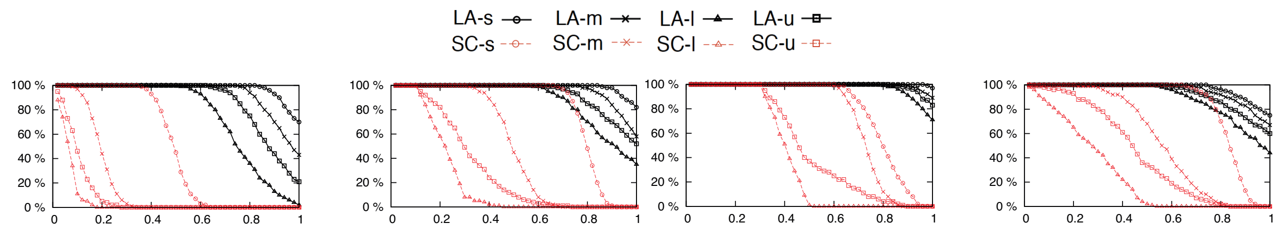
Proof: The proof is identical to the proof of FRD speed-up factor, but has to consider dbf_i^*. Therefore, the corresponding revision of Theorem 11 for the arbitrary speed-up factor becomes $\frac{4}{\alpha}$. By following the same proof procedure, we can reach the conclusion by considering two cases, i.e., whether k is larger than $\frac{e^{0.25}-1}{e^{0.25}}(1+\beta)$. With the same procedure, we can reach the conclusion. Due to space limitation, the details are omitted. ■

7 Experiment

In this section, we conduct extensive experiments using randomly-generated task sets to evaluate the applicability of our linear-time schedulability test (Theorem 8), denoted "LA". We evaluated the effectiveness of LA by comparing it to the suspension-oblivious approach denoted "SC". That is, after transforming all self-suspending tasks into ordinary periodic tasks (no suspensions) using SC, the original task system is schedulable if the total utilization of the transformed task system is no greater than 1. Although the result in [9] is related to this paper, it only works when the system has one self-suspended task, and, hence, is not comparable.

In our experiments, sporadic self-suspending task sets were generated similar to the methodology used in [11], [12]. Task periods, i.e., T_is, were distributed uniformly over [$20ms$, $200ms$]. Task utilizations, i.e., U_is, were distributed differently for each experiment using four uniform distributions. The ranges for the uniform distributions were [0.005, 0.1] (light), [0.1, 0.3] (medium), [0.3, 0.5] (heavy), and [0.005, 0.5] (uniform). Task execution costs were calculated from periods and utilizations.

Suspensions lengths of tasks were also distributed using four uniform distributions: $[0.01\cdot(1-U_i)\cdot T_i, 0.1\cdot(1-U_i)\cdot T_i]$ (short suspension length), $[0.1\cdot(1-U_i)\cdot T_i, 0.3\cdot(1-U_i)\cdot T_i]$ (moderate suspension length), $[0.3\cdot(1-U_i)\cdot T_i, 0.6\cdot(1-U_i)\cdot T_i]$ (long suspension length), and $[0.01\cdot(1-U_i)\cdot T_i, 0.6\cdot(1-$



(a) light task utilization (b) medium task utilization (c) heavy task utilization (d) uniform task utilization

Fig. 6: Schedulability results. In all four graphs, the x-axis denotes the task set utilization cap and the y-axis denotes the fraction of generated task sets that were schedulable. Each graph gives four curves per tested approach for the cases of short, moderate, long, and uniform task suspension lengths, respectively. As seen at the top of the figure, the label "**LA-s(m/h/u)**" indicates the approach of LA assuming short (moderate/long/uniform) task suspension lengths. Similar "**SC**" labels are used for SC.

$U_i) \cdot T_i]$ (uniform suspension length).[2] For each combination of task utilization distribution, suspension length distribution, and U_{sum}, 10,000 task sets were generated. Each such task set was generated by creating tasks until total utilization exceeded the corresponding utilization cap, and by then reducing the last task's utilization so that the total utilization equaled the utilization cap.

The obtained schedulability results are shown in Fig. 6 (the organization of which is explained in the figure's caption). Each curve plots the fraction of the generated task sets the corresponding approach successfully scheduled, as a function of total utilization. As seen, in all tested scenarios, LA clearly improves upon SC by a substantial margin. For example, as seen in Fig 6(a), when task utilizations are light, LA can achieve 100% schedulability when $U_{sum} \leq 0.82$, $U_{sum} \leq 0.76$, $U_{sum} \leq 0.62$, and $U_{sum} \leq 0.5$ with short, moderate, uniform, and heavy suspension lengths, respectively; while SC fails to do so when U_{sum} merely exceeds 0.36, 0.1, 0.02, 0.02, respectively. Moreover, as seen in all four inset of Fig 6, when task suspension lengths are long or uniform, SC fails to schedule most of the generated task sets while LA is able to deliver good performance. For instance, as seen in Fig 6(c), when task utilizations are heavy, LA is able to achieve 100% schedulability when $U_{sum} \leq 0.8$ with heavy or uniform suspension lengths; while SC fails to do so when U_{sum} merely exceeds 0.3. This is because in these cases, the utilization loss due to the conversion of long suspensions into computation under SC is too significant. Another interesting observation is that both methods perform better when task utilizations are heavier. This is due to the fact that with heavier task utilizations, a less number of self-suspending tasks can be generated and the suspension lengths of such tasks are shorter compared to the case with lighter task utilizations. This clearly alleviates the negative impact due to suspensions in the schedulability.

8 Conclusion

For a given sporadic self-suspending task system, if a feasible schedule exists upon a unit-speed processor, can we design a scheduling algorithm that will lead to a feasible schedule when allowed moderately faster resources? To answer this question, we present an FRD scheduling algorithm EDA, that assigns relative deadlines equally to computation phases of self-suspending tasks. We prove that EDA yields non-trivial resource-augmentation performance guarantees. Specifically, we derive a pseudo-polynomial-time schedulability test for EDA that is exact and yields a resource-augmentation bound of 2 and 3 w.r.t. any FRD scheduler and any arbitrary scheduler, respectively. To reduce the time complexity, we further present a linear-time schedulability test for EDA, which yields a resource-augmentation bound of 2.787 and 4.875, w.r.t. any FRD scheduler and any arbitrary scheduler, respectively. Furthermore, experiments presented herein show that our proposed schedulability tests improve upon prior tests by a large margin.

Based on Theorems 4 and 5, we can also adopt the approach proposed in [1] to provide approximate schedulability in polynomial-time complexity. That is, we take a predefined number of (different) discrete values in (7) at beginning and use a linear approximation after the last discrete value. With such an approach, it is also not difficult to show that the studied problem also admits schedulability tests with FRD and arbitrary speed-up factors $2+\epsilon$ and $3+\epsilon$ with polynomial time complexity proportional to $O(\frac{1}{\epsilon})$, respectively.

While we have assumed implicit-deadline self-suspending task systems, we observe that our results can be directly applied to the constrained-deadline cases, and can also be extended to apply to the arbitrary-deadline case. The intuitive reason is because our analysis techniques do not rely on the assumption that $D_i = T_i$ holds for any task τ_i. For future research, we plan to further tighten the analysis, and consider more general self-suspending task models. We also plan to evaluate our proposed schedulability tests using real-world self-suspending workloads, e.g., video processing tasks that may experience suspension delays when accessing GPUs.

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *ECRTS*, pages 187–195, 2004.
[2] S. Baruah and N. Fisher. The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. In *Proc. of the 26th IEEE Real-Time Systems Symp.*, pages 321–329, 2005.
[3] S. K. Baruah, A. K. Mok, and L. E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *IEEE Real-Time Systems Symposium*, pages 182–190, 1990.
[4] J. Chen and S. Chakraborty. Resource Augmentation Bounds for Approximate Demand Bound Functions. In *Proc. of the 32nd IEEE Real-Time Systems Symp.*, pages 272–281, 2011.
[5] J.-J. Chen and S. Chakraborty. Resource augmentation for uniprocessor and multiprocessor partitioned scheduling of sporadic real-time tasks. *Real-Time Systems*, 49(4):475–516, 2013.
[6] 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.

[2]Note that any S_i is upper-bounded by $(1 - U_i) \cdot T_i$

[7] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *Proc. of the 32nd IEEE Real-Time Systems Symp.*, pages 57–66, 2011.

[8] I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, pages 54–59, 1995.

[9] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 3–12, 2010.

[10] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th Real-Time Systems Symposium*, pages 425–436, 2009.

[11] C. Liu and J. Anderson. An O(m) analysis technique for supporting real-time self-suspending task systems. In *Proceedings of the 33th IEEE Real-Time Systems Symposium*, pages 373–382, 2012.

[12] C. Liu and J. Anderson. Suspension-aware analysis for hard real-time multiprocessor scheduling. In *Proceedings of the 25th EuroMicro Conference on Real-Time Systems*, pages 271–281, 2013.

[13] C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *RTSS*, 2014.

[14] C. Liu, J. Chen, L. He, and Y. Gu. Analysis techniques for supporting harmonic real-time tasks with suspensions. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014.

[15] J. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[16] 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 *DAC*, pages 1–6, 2014.

[17] J. C. Palencia and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 26–37, 1998.

[18] C. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proc. of the 29th ACM Symposium on Theory of Computing*, pages 140–149, 1997.

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

[20] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th IEEE Real-Time Systems Symposium*, pages 47–56, 2004.

[21] K. Tindell. Adding time-offsets to schedulability analysis. Technical Report 221, University of York, 1994.

[22] A. Toma and J.-J. Chen. Computation offloading for frame-based real-time tasks with resource reservation servers. In *ECRTS*, pages 103–112, 2013.

Appendix

Proof of Theorem 2. Consider a special input instance with two tasks as follows:

- $C_{1,1} = T - 2 \cdot \varepsilon$, $S_1 = C_{1,2} = 0$, $T_1 = T - \varepsilon$, and $C_{2,1} = \varepsilon$, $S_2 = T - \varepsilon$, $C_{2,2} = 0$, $T_2 = T$,

where ε can be arbitrarily small and $T > 2$ is a constant. It is clear that this task set is feasible if we assign higher priority to jobs of τ_2 over τ_1. However, under EDF or RM, τ_1's jobs may get higher priorities and thus cause jobs of τ_2 to miss deadlines. In order for this task system to be schedulable under EDF or RM, on a α-speed processor, the following must hold:

$$\frac{T - 2\cdot\varepsilon}{\alpha} + \frac{\varepsilon}{\alpha} \leq \varepsilon.$$

By rearrangements, we have $\alpha \geq \dfrac{T}{\varepsilon} - 1$. Thus, $\alpha \to \infty$ as $\varepsilon \to 0$. □

Proof of Lemma 4. We consider four cases:

- If $t < \frac{T_i - S_i}{2}$, we have $dbf_i^{EDA}(t) = 0$, and the inequality $dbf_i(2t) \geq dbf_i^{EDA}(t)$ holds since $dbf_i(2t) \geq 0$.
- If $\frac{T_i - S_i}{2} \leq t < T_i - S_i$, we have $dbf_i^{EDA}(t) = C_{i,\max}$ and $dbf_i(2t) \geq C_{i,1} + C_{i,2} \geq C_{i,\max}$. Therefore, the inequality $dbf_i(2t) \geq dbf_i^{EDA}(t)$ holds.
- If $T_i - S_i \leq t < \frac{3T_i}{2} - \frac{S_i}{2}$, we have $dbf_i^{EDA}(t) = C_{i,1} + C_{i,2}$ and $dbf_i(2t) \geq C_{i,1} + C_{i,2} \geq dbf_i^{EDA}(t)$.
- If $\frac{3T_i}{2} - \frac{S_i}{2} \leq t$, we have

$$\begin{aligned} dbf_i(2t) &\geq_1 dbf_i(t + \frac{3T_i}{2} - \frac{S_i}{2}) \\ &\geq_2 dbf_i(t + T_i) \\ &=_3 dbf_i(t) + C_{i,1} + C_{i,2} \\ &\geq_4 (dbf_i^{EDA}(t) - C_{i,\max}) + C_{i,1} + C_{i,2} \\ &\geq_5 dbf_i^{EDA}(t), \end{aligned}$$

where the inequality $\geq_1$ comes from the definition that $t \geq \frac{3}{2}T_i - \frac{1}{2}S_i$, the inequality $\geq_2$ comes from the fact that $T_i > S_i$, the equality $=_1$ comes from the definition of $dbf_i()$, and the inequality $\geq_4$ comes from Lemma 3.

□

Lemma 12. *For any $t \geq 0$ and any task $\tau_i \in \mathbf{T}$, we have*

$$dbf_i^*(t + T_i) \geq dbf_i^{EDA}(t).$$

Proof: The proof is similar to the proof of Lemma 4. By definition, $dbf_i^*(t + T_i) \geq dbf_i^*(t) + C_{i,\max} \geq dbf_i^{EDA}(t) - C_{i,\max} + C_{i,\max} = dbf_i^{EDA}(t)$. ■

Lemma 13.

$$\inf_{x+y+z>1} \left\{ \max\left\{ \frac{y}{2}, \frac{y}{4} + \frac{x+z}{2} \right\} \right\} > \frac{1}{3}.$$

Proof: Suppose that $x + y + z = 1 + \epsilon$ with $\epsilon > 0$. We can then rewrite $\frac{y}{4} + \frac{x+z}{2}$ to $\frac{y}{4} + \frac{1+\epsilon-y}{2} = \frac{2+2\epsilon-y}{4}$. It is clear that $\frac{y}{2}$ is an increasing function with respect to y, while $\frac{2+2\epsilon-y}{4}$ is a decreasing function with respect to y. Therefore, the infimum happens when $\frac{y}{2}$ is equal to $\frac{2+2\epsilon-y}{4}$, i.e., when y is $\frac{2+2\epsilon}{3}$. Therefore, the infimum is $\frac{1+\epsilon}{3}$, which proves the lemma. ■

Proof of Theorem 5. Suppose that at time $t^* > 0$, we have $\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA}(t^*) > t^*$. Let's now classify the tasks in $\mathbf{T}$ into four task sets:

- $\mathbf{T}_1$: if $t^* < \frac{T_i - S_i}{2}$, task τ_i is in $\mathbf{T}_1$.
- $\mathbf{T}_2$: if $T_i - S_i > t^* \geq \frac{T_i - S_i}{2}$, task τ_i is in $\mathbf{T}_2$.
- $\mathbf{T}_3$: if $\frac{3}{2}T_i - \frac{1}{2}S_i > t^* \geq T_i - S_i$, task τ_i is in $\mathbf{T}_3$.
- $\mathbf{T}_4$: if $t^* \geq \frac{3}{2}T_i - \frac{1}{2}S_i$, task τ_i is in $\mathbf{T}_4$.

Clearly, each task τ_i in $\mathbf{T}$ is either in $\mathbf{T}_1$, $\mathbf{T}_2, \mathbf{T}_3$, or $\mathbf{T}_4$. Similarly, we know that $\sum_{\tau_i \in \mathbf{T}_1} dbf_i^{EDA}(t^*) = 0$. Now suppose that $\frac{\sum_{\tau_i \in \mathbf{T}_2} dbf_i^{EDA}(t^*)}{t^*}$ is x, $\frac{\sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*)}{t^*}$ is y and $\frac{\sum_{\tau_i \in \mathbf{T}_4} dbf_i^{EDA}(t^*)}{t^*}$ is z. Since $\sum_{\tau_i \in \mathbf{T}} dbf_i^{EDA}(t^*) > t^*$, we have

$$z + y + x > 1.$$

Based on the definition of $dbf_i^*()$ in (4) and $\frac{3}{2}T_i - \frac{1}{2}S_i > t^* \geq T_i - S_i$ for any task τ_i in $\mathbf{T}_3$, we know that

$$\begin{aligned} \sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*) &= \sum_{\tau_i \in \mathbf{T}_3} (C_{i,1} + C_{i,2}) \\ &\leq 2 \sum_{\tau_i \in \mathbf{T}_3} C_{i,\max} = 2 \sum_{\tau_i \in \mathbf{T}_3} dbf_i^*(t^*). \end{aligned} \tag{22}$$

Therefore, by (22), we know that

$$\max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(t)}{t} \geq \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(t^*)}{t^*}$$
$$\geq \frac{\sum_{\tau_i \in \mathbf{T}_3} dbf_i^*(t^*)}{t^*} \geq \frac{\sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*)}{2t^*} = \frac{y}{2}. \quad (23)$$

For a task τ_i in $\mathbf{T}_4$, we know that

$$\begin{aligned}\forall \tau_i \in \mathbf{T}_4, \quad dbf_i^*(2t^*) &\geq dbf_i^*(t^* + \frac{3}{2}T_i - \frac{1}{2}S_i) \\ &\geq dbf_i^*(t^* + T_i) \\ &\geq dbf_i^{EDA}(t^*), \end{aligned} \quad (24)$$

where the first inequality comes from the definition that $t^* \geq \frac{3}{2}T_i - \frac{1}{2}S_i$, the second inequality comes from the fact that $T_i > S_i$, and the third inequality comes from Lemma 12. Therefore, we have

$$\sum_{\tau_i \in \mathbf{T}_4} dbf_i^{EDA}(t^*) \leq \sum_{\tau_i \in \mathbf{T}_4} dbf_i^*(2t^*). \quad (25)$$

Moreover, since $T_i - S_i > t^* \geq \frac{T_i - S_i}{2}$ for task τ_i in $\mathbf{T}_2$, we know that

$$\begin{aligned}\sum_{\tau_i \in \mathbf{T}_2} dbf_i^{EDA}(t^*) &= \sum_{\tau_i \in \mathbf{T}_2} C_{i,\max} \\ &= \sum_{\tau_i \in \mathbf{T}_2} dbf_i^*(T_i - S_i) \\ &\leq \sum_{\tau_i \in \mathbf{T}_2} dbf_i^*(2t^*). \end{aligned} \quad (26)$$

By (22), we also know that

$$\sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*) \leq 2 \sum_{\tau_i \in \mathbf{T}_3} dbf_i^*(t^*) \quad \leq 2 \sum_{\tau_i \in \mathbf{T}_3} dbf_i^*(2t^*) \quad (27)$$

By combining the above inequalities in (25), (26), and (27),

$$\begin{aligned}&\max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(t)}{t} \\ \geq& \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(2t^*)}{2t^*} \\ \geq& \frac{\sum_{\tau_i \in \mathbf{T}_4 \cup \mathbf{T}_3 \cup \mathbf{T}_2} dbf_i^*(2t^*)}{2t^*} \\ \geq& \frac{\sum_{\tau_i \in \mathbf{T}_4 \cup \mathbf{T}_2} dbf_i^{EDA}(t^*)}{2t^*} + \frac{\sum_{\tau_i \in \mathbf{T}_3} dbf_i^{EDA}(t^*)}{4t^*} \\ =& \frac{z+x}{2} + \frac{y}{4}. \end{aligned} \quad (28)$$

By (23) and (28), we conclude the proof by showing that

$$\begin{aligned}&\max_{t>0} \frac{\sum_{\tau_i \in \mathbf{T}} dbf_i^*(t)}{t} \\ \geq& \max\left\{\frac{y}{2}, \frac{z+x}{2} + \frac{y}{4}\right\} \\ \geq& \inf_{y+x+z>1} \left\{\max\left\{\frac{y}{2}, \frac{z+x}{2} + \frac{y}{4}\right\}\right\} \\ >& \frac{1}{3}, \end{aligned} \quad (29)$$

where the last inequality comes from Lemma 13. □

Proof of Corollary 1. The schedulability test in (13) requires only to test $n+1$ time points, i.e., $\Delta_1, \Delta_2, \ldots, \Delta_n, \infty$. Moreover, since $\sum_{i=1}^{\ell+1} dbf_i^{EDA^\dagger}(\Delta_{\ell+1}) = dbf_{\ell+1}^{EDA^\dagger}(\Delta_{\ell+1}) + ((\Delta_{\ell+1} - \Delta_\ell)\sum_{i=1}^{\ell} U_i) + (\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell))$, calculating $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell)$ can be done in $O(1)$ time complexity by storing $\sum_{i=1}^{\ell} U_i$ incrementally and referring to $\sum_{i=1}^{\ell} dbf_i^{EDA^\dagger}(\Delta_\ell)$. Therefore, the time complexity is $O(n)$, under the assumption that the tasks are ordered in a non-decreasing order with respect to Δ_i. □

Proof of Lemma 9. Since $\frac{1}{1+k+\beta-\frac{x}{\alpha}}$ is non-negative for the given non-negative parameters, k, β, x, α in the integration range, we have

$$\begin{aligned}&\int_0^{x^*} \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx \\ \leq& \int_0^{\frac{\alpha(1+k)(e^{0.5}-1)}{e^{0.5}}} \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx \\ =& \alpha \left(log_e \frac{1+k+\beta}{1+k+\beta-\frac{(1+k+\beta)(e^{0.5}-1)}{e^{0.5}}} \right) = 0.5\alpha \end{aligned}$$

□

Proof of Lemma 10 . The proof is basically identical to the proofs of Lemmas 4, 5, and 7 in [5]. The connection between the two functions $dbf_i^{EDA^\flat}(t)$ and $dbf_i^{EDA^\dagger}(t)$ is identical to the linear approximation when considering normal sporadic tasks in [5]. Let W be the *partially released workload*, in which

$$W \overset{\text{def}}{=} \sum_{i=1}^{\ell-1} (\Delta_\ell - \delta_i) U_i = k^\dagger \sum_{i=1}^{\ell-1} dbf_i^{EDA^\flat}(\delta_i). \quad (30)$$

The proof can be done by reducing the relative deadlines (Lemma 4 in [5]) and increasing the minimum inter-arrival time properly (Lemma 5 in [5]) to increase the value of W.

Under the assumption of α, we know that k is equal to $\frac{\alpha(1+k+\beta)(e^{0.5}-1)}{e^{0.5}}$. Therefore, as in the proof of Lemma 9, we know that $\int_0^k \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx$ is equal to $\frac{\alpha}{2}$. Moreover, based on Lemma 9, it can be proved (as in Lemma 7 in [5]) that the utilization $\sum_{i=1}^{\ell-1} U_i$ must be larger than $\int_0^k \frac{1}{1+k+\beta-\frac{x}{\alpha}} dx$; otherwise W will not be sufficient to make the schedulability test in Theorem 8 fail. □

Proof of Lemma 11. This comes directly from Lemma 10 and Theorem 11 in which $\max\left\{\max_{t>0} \frac{\sum_{i=1}^{\ell} dbf_i(t)}{t}, \sum_{i=1}^{\ell-1} U_i\right\}$ is larger than $\frac{\alpha}{2}$. Moreover, we further drop β in the numerator in (21), due to the fact that α is $\frac{\frac{e^{0.5}}{(e^{0.5}-1)}(k+\beta)}{(1+k+\beta)} \geq \frac{\frac{e^{0.5}}{(e^{0.5}-1)}(k)}{(1+k+\beta)}$. □