Self-Suspension Real-Time Tasks under Fixed-Relative-Deadline Fixed-Priority Scheduling

Wen-Hung Huang and Jian-Jia Chen
Department of Computer Science
TU Dortmund University, Germany

ABSTRACT

Self-suspension is becoming a prominent characteristic in real-time systems such as: (i) I/O-intensive systems (ii) multi-core processors, and (iii) computation offloading systems with coprocessors, like Graphics Processing Units (GPUs). In this work, we study self-suspension systems under fixed-priority (FP) fixed-relative-deadline (FRD) algorithm by using release enforcement to control self-suspension tasks' behavior. Specifically, we use equal-deadline assignment (EDA) to assign the release phases of computations and suspensions. We provide analysis for deriving the speedup factor of the FP FRD scheduler using suspension-laxity-monotonic (SLM) priority assignment. This is the first positive result to provide bounded speedup factor guarantees for general multi-segment self-suspending task systems.

1. INTRODUCTION

In many real-time and embedded systems, tasks may be suspended by the operating system when accessing external devices such as disks, graphical processing units (GPUs), or synchronizing with other tasks. This behavior is often known as *self-suspension*. Self-suspensions are even more pervasive in many emerging embedded cyber-physical systems in which the computation components frequently interact with external and physical devices. Such suspension delays have negative impact on the timing predictability and cause intractability in real-time scheduling [17].

Self-suspending tasks can be classified into two models: *dynamic* self-suspension and *multi-segment* self-suspension models. The dynamic self-suspension sporadic task model characterizes the execution of a job of each task τ_i with its upper bound on the (total) execution time C_i and its upper bound on (total) self-suspension time S_i. That is, self-suspension under such a model can happen at any point obliviously during the execution of a job as long as the suspension time does not exceed the maximum self-suspension time S_i. On the other hand, the multi-segment sporadic task suspending model characterizes the execution of a job of a task τ_i by specifying its computation segments and suspension intervals as an array $(C_i^0, S_i^0, C_i^1, S_i^1, ..., S_i^{m_i-2}, C_i^{m_i-1})$ composed of m_i computation segments separated by $m_i - 1$ suspension intervals.

From the system designer's perspective, the dynamic self-suspension model provides an easy way to specify self-suspending systems without considering the juncture of I/O access or computation offloading. However, from the analysis perspective, such a dynamic model leads to quite pessimistic results in terms of schedulability since the location of suspensions within a job is oblivious. Therefore, if the suspending patterns are well-defined and characterized with known suspending intervals, the multi-segment self-suspension task model is more appropriate.

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. If an algorithm $\mathcal{A}$ has a *speedup 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*. Therefore, designing scheduling algorithms and schedulability tests with bounded speedup factors (resource augmentation factors, equivalently) also ensures their qualities for such $\mathcal{NP}$-hard problems.

Related work. Self-suspending real-time tasks have been studied in the literature [5,7–13,15,16]. Unfortunately, a few results in the literature are recently shown flawed. Please refer to [4] and [15] for some of the existing flaws that have been formally reported. Thus, none of the results with wrong analysis will be further referred in this paper.

Recently, there have been some results [8, 11–13] on the *dynamic* self-suspension task model [14]. The state-of-the-art approach [8] is guaranteed to find a feasible fixed-priority assignment on a speed-2 uniprocessor, if a feasible fixed-priority assignment exists on a unit-speed processor. It is evident that the multi-segment self-suspension task is a special case of the dynamic self-suspension task, where the locations and the number of suspension intervals can vary from one job to another. However, this also implies that the speedup factor 2 for dynamic self-suspension tasks does not hold further when we consider multi-segment self-suspension tasks. Some details about the above discussion can be found in Section 4.

The multi-segment self-suspension task model has been specifically studied in [5, 15, 16]. Palencia and Harbour [16] study fixed-priority scheduling for tasks with dynamic offsets, which can also be adopted to analyze the schedulability of fixed-priority self-suspending task systems. Nelissen et al. [15] propose a method based on a mixed integer linear programming (MILP) formulation to calculate the worst-case response time of a multi-segment self-suspension task. In these two results, the authors assume a given priority assignment, and perform schedulability analysis. The analysis in [16] can be reduced to pseudopolynomial time complexity (with some approximations), but the analysis in [15] requires exponential time complexity.

The only approximation result (with speedup factor guarantees) for multi-segment self-suspension scheduling problems is presented by Chen and Liu [5], for a special case when *there is only at most one self-suspending interval per task*. In their result, each computation segment is assigned a relative deadline according to the equal-deadline assignment (EDA), and then scheduled under earliest-deadline-first (EDF) scheduling. Chen and Liu [5] prove that the

978-3-9815370-6-2/DATE16/

speedup factor of EDA under EDF scheduling is 3. However, the schedulability analysis and the speedup factor analysis in [5] are restrictive since they can only be applied under dynamic priority scheduling and with at most one self-suspending interval per task.

Contributions. In this work, we analyze simple scheduling strategies and derive the first bounded speedup factor guarantees for the multi-segment self-suspension scheduling problem under fixed-priority scheduling. The contributions of this paper are summarized as follows:

- In Section 3, we explain how fixed-relative-deadline (FRD) scheduling with release time enforcement works. We show that tasks under the FRD scheduler can be equivalently transformed to the well-known generalized multiframe (GMF) [2] tasks. Specifically, we study the equal deadline assignment (EDA) with multiple suspension intervals per task under the suspension-laxity-monotonic (SLM) priority assignment: the smaller the suspension laxity, the higher the priority level.
- Prior to this paper, only very negative results are known in [17] for general cases. That is, the speedup factor for EDF and rate-monotonic scheduling (RM) can be ∞. In Section 4, we show that EDA together with SLM has a processor speedup factor M^2, where $M = \max_{\tau_i} max(2, m_i)$. Note that there is no better lower bound or upper bound with respect to the quality (speedup factor) of the scheduling policy and the schedulability analysis. Although this factor is not a constant, it provides the first evidence that simple strategies can still work pretty well when the number of computation segments is small. Therefore, the scheduling problem remains open when M is relatively large.
- We empirically show that our proposed approach is highly effective if the number of suspending intervals is small (< 5), in terms of the number of task sets that are deemed schedulable, despite that it is impractical for task sets with many suspension intervals (≥ 5), in Section 5.
- Although this paper focuses on fixed-priority scheduling, we can conclude that the speedup factor M^2 also holds for EDF scheduling (under FRD and release time enforcement). This will be discussed in the conclusion.

2. SYSTEM MODEL AND NOTATIONS

We consider a real-time system to execute a set of n independent, preemptive, multi-segment self-suspension real-time tasks $\tau = \{\tau_1, \tau_2, ..., \tau_n\}$ on a uniprocessor. Each task can release an infinite number of jobs under minimum inter-arrival time (temporal) constraints. Similar to sporadic tasks, a multi-segment self-suspension sporadic task releases jobs sporadically, but the execution of each job of τ_i alternates between pre-defined computation and suspension phases in an interleaving manner. The execution of each job of τ_i is composed of m_i *computation* segments separated by $m_i - 1$ *suspension* intervals. A computation segment is eligible to execute only after the completion of the previous suspension interval. A multi-segment self-suspension task τ_i is characterized by 3 tuples:

$$\tau_i = \big((C_i^0, S_i^0, C_i^1, S_i^1, ..., S_i^{m_i-2}, C_i^{m_i-1}), T_i, D_i\big)$$

where T_i denotes the minimum inter-arrival time of τ_i, D_i denotes the relative deadline of task τ_i; C_i^j denotes the upper bound on the execution time of the $(j+1)$th- computation segment; and S_i^j denotes the upper bound on the suspension time of the $(j+1)$th- suspension interval.

For the simplicity of presentations, for the rest of this paper, we will implicitly call such tasks as self-suspension tasks as the context is clear. In this work, we restrict our attention to *constrained-deadline* task systems, i.e., $D_i \leq T_i$.

If m_i is 1, there is only one computation segment of task τ_i, which is equivalent to the conventional sporadic task model. We further define such a task as $\tau_i = (C_i, T_i, D_i)$. If $m_i \geq 2$, we denote the amount of total computation segment lengths $\sum_{j=0}^{m_i-1} C_i^j$ as C_i and the amount of total suspension interval lengths $\sum_{j=0}^{m_i-2} S_i^j$ as S_i. The computation segment with the maximum execution time is denoted as $C_i^{max} = max_{0 \leq j \leq m_i-1}\{C_i^j\}$. We assume that $C_i + S_i \leq D_i$ for any task $\tau_i \in \tau$. The utilization of task τ_i is defined as $U_i = C_i/T_i$. We further assume that $\sum_{i=1}^{n} U_i \leq 1$.

In this paper, we focus on fixed-priority scheduling, in which each task is associated with a unique priority level. For a task τ_k in τ, we denote the set of the tasks with higher priority than task τ_k as $hp(k)$. More precisely, all the jobs of a task have the same priority level, and the system always selects the job in the ready queue with the highest-priority level to execute. Clearly, if a job suspends itself, it is no longer in the ready queue. On the other hand, when a job resumes from its self-suspension, it is put into the ready queue again.

Our proposed schedulability analysis works for any fixed-priority assignment. However, it has been shown in [5, 17] that rate-monotonic (RM) and deadline-monotonic (DM) can have very bad performance. Alternatively, we will consider two heuristic fixed-priority assignments, one is suspension-laxity-monotonic (SLM), and another is the optimal priority assignment (OPA) approach proposed by Audsley [1]. The suspension-laxity-monotonic (SLM) scheduling prioritizes the tasks according to their suspension laxity $D_i - S_i$: the smaller the suspension laxity $D_i - S_i$, the higher the priority level. The ties are broken arbitrarily. The OPA priority assignment [1] starts from the lowest-priority level iteratively to the highest-priority level based on an OPA-compatible schedulability test.

We define the following feasibility and schedulability to be used for the rest of the paper.

- A schedule is *feasible* if there is no deadline miss and all the scheduling constraints are respected.
- A self-suspension task system τ is said to be *schedulable* if there exists a feasible schedule for the task system for any release patterns under the temporal constraints.
- A self-suspension task system τ is said to be *schedulable* under a scheduling algorithm if the schedule produced by the algorithm for the task system is always feasible.

3. FIXED-RELATIVE-DEADLINE (FRD)

As discussed in Section 1, the *fixed-relative-deadline* (FRD) scheduler proposed by Chen and Liu [5] is now the only scheduling algorithm that has speedup factor guarantees for multi-segment self-suspension sporadic task systems. The FRD scheduler works based on a release (scheduling) enforcement that assigns each computation segment C_i^j with a relative deadline D_i^j, as also shown in Figure 1:

- The release times between two consecutive computation segments C_i^j, C_i^{j+1} are separated by *exactly* D_i^j time units plus the upper bound time on the suspen-

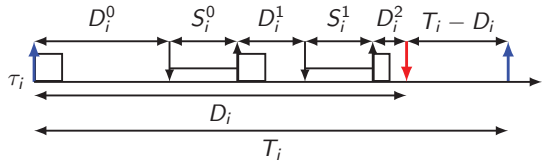

Figure 1: A self-suspension task τ_i under FRD scheduler where $D_i^0 + D_i^1 + D_i^2 = D_i - S_i$. The blue upward and the red downward arrows indicate the release time and the relative deadline, respectively, of task τ_i.

sion interval, i.e. $D_i^j + S_i^j$. For completeness, we introduce a auxiliary variable $S_i^{m_i-1} \equiv T_i - D_i \geq 0$.

- Each suspension interval S_i^{j+1} is released at the absolute deadline of the previous computation segment. That is, if a computation segment C_i^j is released at time t and is set an absolute deadline $t+D_i^j$ under FRD scheduling, then the follow-up suspension interval is released at $t + D_i^j$.
- The sum of the assigned relative deadlines of task τ_i's computation segments cannot exceed $D_i - S_i$, i.e., $\sum_{j=0}^{m_i-1} D_i^j \leq D_i - S_i$.

Although the FRD concept can be generalized to multiple computation segments, the (sufficient test) analysis by Chen and Liu [5] only works for task systems with at most one self-suspension interval per task under EDF.

Note that the release times of the computation segments by using above release enforcement have to be strictly enforced. Otherwise, the resulting early release of the subsequence computation phases may introduce more interference than the above release enforcement, due to jitters. The above release enforcement can be achieved by designing the suspensions properly. For example, every suspension has to initialize a timer (interrupt) to set a predefined suspension length. Even if self-suspension returns earlier, the following computation segment is not placed back to the ready-queue until the timer is triggered.

With the above treatment, every computation segment of the task has to finish its execution before the assigned relative deadline. In a nutshell, the FRD scheduler is a release (scheduling) enforcement that releases computation segments and suspension intervals by strictly following the assigned time instants, irrespective of any early completion during run-time. The concept of the FRD scheduler is very simple. *The key design challenge is to determine the priority assignment and the relative deadlines of the computation segments.* If the priority assignment and relative deadline assignment are both given, the schedulability test can be easily done as in Section 3.1 by transforming the problem to the generalized multiframe task model. In Sections 3.2, we will discuss about two simple heuristics.

3.1 Schedulability Analysis of FRD

The generalized multiframe (GMF) task model was first introduced by Baruah et al. [2]. A GMF task ψ_i consisting of m_i frames is characterized by the 3-tuple $(\vec{C_i}, \vec{D_i}, \vec{T_i})$, where $\vec{C_i}$,$\vec{D_i}$, and $\vec{T_i}$ are m_i-ary vectors $(C_i^0, C_i^1, ..., C_i^{m_i-1})$ of execution requirements, $(D_i^0, D_i^1, ..., D_i^{m_i-1})$ of relative deadlines, $(T_i^0, T_i^1, ..., T_i^{m_i-1})$ of minimum inter-arrival times, respectively. In fact, from the analysis perspective, a self-suspension task τ_i under FRD scheduler is equivalent to a GMF task ψ_i, by considering the computation segments as the frames with different separation times:

Lemma 1. *The schedulability analysis problem under FRD scheduling is equivalent to the schedulability analysis of the following generalized multiframe task model by converting each self-suspending task τ_i into a generalized multiframe ψ_i in which*

$$\vec{C_i} = \left(C_i^0, C_i^1, ..., C_i^{m_i-1}\right), \qquad \vec{D_i} = \left(D_i^0, D_i^1, ..., D_i^{m_i-1}\right) \tag{1}$$

and

$$\vec{T_i} = \left(D_i^0 + S_i^0, D_i^1 + S_i^1, \ldots, D_i^{m_i-1} + S_i^{m_i-1}\right), \tag{2}$$

where $S_i^{m_i-1} \equiv T_i - D_i$ for completeness.

Proof. Due to space limitation, we only provide a sketched proof. Computation segments under FRD scheduling are released according to the assigned time instants, which are exactly separated by one assigned relative deadline and one suspension interval, i.e., $D_i^j + S_i^j$. Therefore, when a job of task τ_i is released to the system at time θ_a, under the FRD scheduling with release enforcement, (1) the first computation segment has an absolute deadline $\theta_a + D_i^0$ to finish C_i^0 amount of execution, (2) the second computation segment C_i^1 is released at time $\theta_a + D_i^0 + S_i^0$, has an absolute deadline $\theta_a + D_i^0 + S_i^0 + D_i^1$, (3) the third computation segment C_i^2 is released at time $\theta_a + D_i^0 + S_i^0 + D_i^1 + S_i^1$, has an absolute deadline $\theta_a + D_i^0 + S_i^0 + D_i^1 + S_i^1 + D_i^2$, (4) etc. Since $\sum_{j=0}^{m_i-1} D_i^j \leq D_i - S_i$ in an FRD assignment, the absolute deadline of the last computation segment in the above release is $\theta_a + D_i$. Therefore, the earliest arrival time of the next job of task τ_i after time $\theta_a + D_i$ is at time $\theta_a + T_i$, which results in the last auxiliary setting by setting $S_i^{m_i-1}$ to $T_i - D_i$. □

It has been shown in [18] that the interference (the demand requested by a GMF task τ_i in an interval length t) from a GMF task ψ_i can be efficiently calculated and upper-bounded, as paraphrased in the following lemma:

Lemma 2 (Takada and Sakamura [18]). *The maximum execution time (demand) $W_i(t)$ that higher-priority task ψ_i interferes with lower-priority tasks within an interval of length t is upper bounded by*

$$W_i(t) = max_{0 \leq h \leq m_i-1} E_i^h(t) \tag{3}$$

where

$$E_i^h(t) = \sum_{j=h}^{h+\ell-1} C_i^{j \bmod m_i} + min\left(C_i^{(h+\ell) \bmod m_i}, t - \sum_{j=h}^{h+\ell-1} T_i^{j \bmod m_i}\right) \tag{4}$$

and ℓ is the maximum integer under the following condition:

$$\sum_{j=h}^{h+\ell-1} T_i^{j \bmod m_i} \leq t \tag{5}$$

Proof. This comes from Section 4 in [18]. □

We then make use of the GMF interference function to analyze the schedulability of a GMF task (equivalently, a self-suspension segment under FRD scheduler), by adopting the time-demand analysis (TDA).

Lemma 3 (Takada and Sakamura [18]). *The $(j+1)$-th frame of a GMF task ψ_k can meet its relative deadline D_k^j if*

$$\exists 0 < t \leq D_k^j, \ s.t. \ C_k^j + \sum_{\psi_i \in hp(k)} W_i(t) \leq t,$$

where $hp(k)$ is the set of the higher-priority GMF tasks than ψ_k.

3.2 Fixed-Relative-Deadline Assignment

Proportional Assignment: Intuitively, one may assign the relative deadline of the computation segments proportional to their computation times, i.e. $D_i^j = (D_i - S_i) \cdot C_i^j / C_i$. However, this assignment performs rather poor in terms of processor speedup factor, as shown in [5] for one-segment self-suspension systems under dynamic scheduling. Likewise, under fixed-priority scheduling, the same example would also lead to very poor speedup factor results. Therefore, we will not further consider this assignment.

Equal-Deadline Assignment: It has been shown in [5] that Equal-Deadline Assignment (EDA) can provide a non-trivial resource-augmentation performance guarantee for self-suspension systems (with $m_i \leq 2$ for each $\tau_i \in \tau$) scheduled by dynamic scheduling, e.g. *earliest-deadline-first* (EDF). EDA assigns equal relative deadlines to computation segments, considering from a total slack $D_i - S_i$:

$$D_i^0 = D_i^1 = \cdots = D_i^{m_i - 1} = \frac{D_i - S_i}{m_i} \tag{6}$$

In this paper, we intend to answer an open question to know whether EDA works under fixed-priority scheduling with multiple self-suspension intervals. In the following theorem, we can summarize the schedulability test for a constrained-deadline self-suspension task.

THEOREM 1 (EDAGMF). *A constrained-deadline self-suspension task system τ is schedulable under fixed-priority EDA scheduling if the schedulability test in Lemma 2 by setting D_k^j to $\frac{D_k - S_k}{m_k}$ succeeds for every task $\tau_k \in \tau$.*

PROOF. This comes from Lemmas 1 and 2 under EDA. □

It is not difficult to see that the schedulability test in Theorem 1 is in fact compatible with the well-known *Optimal Priority Assignment* (OPA), proposed by Audsley [1]. The OPA algorithm assigns each priority level k to one of the unassigned tasks that has no deadline miss along with the other unassigned tasks, assumed to have higher priority levels. The iterative priority assignment terminates as soon as either no unassigned task can be assigned at the priority level k or all priority levels are assigned. We will evaluate the performance by using the OPA priority assignment, compared to the SLM priority assignment, in Section 5.

4. SPEEDUP FACTOR OF EDA

Although EDA with the SLM priority assignment is a simple strategy, we will provide the analysis in this section that such a strategy has a bounded speedup factor guarantee. Prior to this result, there was no positive result with theoretical analysis for this studied problem. To prove the speedup factor, we first need the *necessary condition* for the existence of a feasible schedule for self-suspension sporadic real-time tasks. Here, we would like to emphasize that the necessary condition (Theorem 3 in [8]) for fixed-priority scheduling in dynamic self-suspension models cannot be applied for multi-segment self-suspension models. The reason is that the self-suspension can happen arbitrarily in the necessary condition in [8], but here in this paper self-suspensions are segmented. Therefore, the proof in [8] cannot be applied here.

The necessary condition is established upon the concepts of *demand bound function* (dbf) used widespread in real-time schedulability analysis. Roughly speaking, for any $t > 0$, the demand bound function of a task bounds the cumulative execution requirement by its jobs that arrive in, and have to be necessarily finished within any interval of length t. In Lemma 4, we formally prove the demand bound function of a self-suspension task, and then it is used to establish the necessary condition by collecting all the tasks' demand from a self-suspension task system:

LEMMA 4. *If a self-suspension task system τ is schedulable (i.e., there exists a feasible schedule for any release patterns under the temporal constraints), then*

$$\forall t > 0, \sum_{\tau_i \in \tau} dbf_i(t) \leq t \tag{7}$$

where

$$dbf_i(t) = \begin{cases} 0 & \text{if } 0 \leq t < D_i - S_i, \\ C_i^{max} & \text{if } D_i - S_i \leq t < D_i, \\ C_i + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor C_i & \text{if } t \geq D_i \end{cases}$$

PROOF. We prove this lemma by contrapositive: *if there exists $t^* > 0$ such that $\sum_{\tau_i \in \tau} dbf_i(t^*) > t^*$, then there exists a release pattern resulting in an infeasible schedule.* Note that this is a necessary condition for any optimal scheduling. Therefore, any release pattern that does not violate the temporal constraints can be considered.

Let's now examine a specific time interval $[\theta_a, \theta_a + t^*)$. By the value t^*, a task τ_i can be in either of the following three cases exclusively:

- Case 1: $0 \leq t^* < D_i - S_i$. For such a case, we do not release any job from this task τ_i.
- Case 2: $t^* \geq D_i$. For such a case, we release the first job of this task τ_i at time θ_a, and the subsequent jobs of task τ_i as early as possible by respecting the task period, i.e., at $\theta_a, \theta_a + T_i, \theta_a + 2T_i, \ldots$. The workload released no earlier than θ_a and has to be finished by $\theta_a + t^*$ is hence by definition *at least* $C_i + \left\lfloor \frac{t^* - D_i}{T_i} \right\rfloor C_i$.
- Case 3: $D_i - S_i \leq t^* < D_i$. For such a case, we need a pattern to ensure that at least C_i^{max} is released no earlier than θ_a and has to be finished no later than $\theta_a + t^*$. Let's assume that the $(v+1)$-th computation segment C_i^v of task τ_i is the one with the maximum execution time among the computation segments of task τ_i. We consider the following release pattern of task τ_i:
  - A job of task τ_i is released at time $\theta_a - \sum_{j=0}^{v-1} S_i^j$.
  - The execution time of a computation segment C_i^j of this job is 0 for any $j = 0, 1, 2, \ldots, v-1$.
  - The execution time of a computation segment C_i^j of this job is its worst-case execution time for any $j = v, v+1, v+2, \ldots, m_i - 1$.
  - The suspension time of a suspending interval of this job is always equal to its worst case S_i^j for any $j = 0, 1, 2, \ldots, m_i - 1$.

  Therefore, C_i^v computation segment is released at time θ_a and followed by a total of suspension times $\sum_{j=v}^{m_i-1} S_i^j$ preceding the absolute deadline $\theta_a - \sum_{j=0}^{v-1} S_i^j + D_i$. If C_i^v cannot be finished before $\theta_a - \sum_{j=0}^{v-1} S_i^j + D_i - \sum_{j=v}^{m_i-1} S_i^j = \theta_a + D_i - S_i$, we can already conclude that task τ_i misses its deadline. Otherwise, we can *artificially* set the absolute deadline of C_i^v to some point no later than $\theta_a + t^*$. As a result, for the last case, the workload that is released no earlier than θ_a and has to be finished by $\theta_a + t^*$ is hence *at least* C_i^{max}.

From the above analysis, we know that the existence of t^* with $\sum_{\tau_i \in \tau} dbf_i(t^*) > t^*$ implies that the workload from the generated jobs above, arriving no earlier than θ_a and with absolute deadline no later than $\theta_a + t^*$, is strictly larger than t^*. Therefore, there does not exist any feasible schedule for such jobs. Hence, this lemma is proved. □

With the above necessary condition, we can now provide the speedup factor analysis for the schedulability test of Theorem 1 where tasks are prioritized according to their suspension laxity $D_i - S_i$, called suspension-laxity-monotonic (SLM) scheduling, i.e. the smaller the suspension laxity, the higher the priority (with ties broken arbitrarily). We need the following lemma to safely bound the function $W_i(t)$ in the GMF schedulability test.

LEMMA 5. *The function $W_i(t)$ defined in Eq. (3) is upper bounded by $W_i(t) \leq \left\lceil \frac{t}{T_i} \right\rceil C_i$, where C_i is $\sum_{j=0}^{m_i-1} C_i^j$.*

PROOF. This is based on simple arithmetic and observations. We can imagine a more pessimistic analysis by pushing all the demand of the frames to only one frame. □

THEOREM 2. *The schedulability test of Theorem 1 using SLM has a processor speedup factor M^2, where M is $\max_{\tau_i \in \tau} max(2, m_i)$.*

PROOF. We prove this theorem by contrapositive. If the schedulability test of Theorem 1 using SLM fails for task τ_k, then, we prove that $\sum_{\tau_i \in \tau} dbf_i(D_k - S_k) \cdot M^2 > D_k - S_k$, where $dbf_i(t)$ is defined in Lemma 4. The failure of Theorem 1 (with $D_k^j = \frac{D_k - S_k}{m_k}$) means that

$$\forall 0 < t \leq \frac{D_k - S_k}{m_k}, \qquad C_k^{max} + \sum_{\tau_i \in hp(k)} W_i(t) > t. \quad (8)$$

Therefore, by Lemma 5, we also know that

$$\forall 0 < t \leq \frac{D_k - S_k}{m_k}, \qquad C_k^{max} + \sum_{\tau_i \in hp(k)} \left\lceil \frac{t}{T_i} \right\rceil C_i > t \quad (9)$$

That is, we have $C_k^{max} + \sum_{\tau_i \in hp(k)} \left\lceil \frac{\frac{D_k - S_k}{m_k}}{T_i} \right\rceil C_i > \frac{D_k - S_k}{m_k}$.

By the definition of SLM scheduling, we know that $D_i - S_i \leq D_k - S_k$ for higher-priority task τ_i in $hp(k)$. There are two different cases for higher-priority task τ_i in $hp(k)$:

- Case 1: $0 < D_i - S_i \leq D_k - S_k \leq T_i$. In this case,

$$\left\lceil \frac{(D_k - S_k)/m_k}{T_i} \right\rceil C_i \leq \left\lceil \frac{T_i}{m_k \cdot T_i} \right\rceil C_i = C_i = m_i \frac{C_i}{m_i}$$
$$\leq m_i C_i^{max} = m_i \cdot dbf_i(D_i - S_i) \leq m_i \cdot dbf_i(D_k - S_k) \quad (10)$$

- Case 2: $D_k - S_k > T_i$. By the assumption of the constrained-deadline task systems, we have $D_i \leq T_i$ and

$$\left\lceil \frac{(D_k - S_k)/m_k}{T_i} \right\rceil C_i$$
$$\leq \left(\left\lfloor \frac{(D_k - S_k)/m_k}{T_i} \right\rfloor + 1 \right) C_i \leq \left(\left\lfloor \frac{D_k - S_k}{T_i} \right\rfloor + 1 \right) C_i$$
$$= \left(\left\lfloor \frac{D_k - S_k - T_i}{T_i} \right\rfloor + 2 \right) C_i \leq 2 \left(\left\lfloor \frac{D_k - S_k - D_i}{T_i} \right\rfloor + 1 \right) C_i$$
$$= 2 dbf_i(D_k - S_k) \quad (11)$$

The second inequality comes from $m_k \geq 1$.

Therefore, with the assumption in Eq. (9), and the above two cases in Eq. (10) and Eq. (11), we can conclude that

$$\frac{D_k - S_k}{m_k} < C_k^{max} + \sum_{\tau_i \in hp(k)} \left\lceil \frac{(D_k - S_k)/m_k}{T_i} \right\rceil C_i$$
$$\leq dbf_k(D_k - S_k) + \sigma_k \cdot \sum_{\tau_i \in hp(k)} dbf_i(D_k - S_k), \quad (12)$$

where σ_k is defined as $max_{\tau_i \in hp(k)} max(2, m_i)$. Therefore, we can have

$$D_k - S_k < \sum_{\tau_i \in \tau} (m_k \sigma_k) \cdot dbf_i(D_k - S_k).$$

This concludes that the speedup factor when the test in Theorem 1 fails for a certain τ_k is $m_k \sigma_k$. Since $m_k \sigma_k \leq M^2$, the speedup factor is M^2 regardless of τ_k. □

5. EXPERIMENTAL RESULTS

In this section, we conduct experiments using synthesized task sets for evaluating the schedulability as follows:

- *Idv-Burst-RM*: the polynomial-time test for dynamic self-suspension tasks under RM scheduling in Corollary 2 in [13].
- *PASS-OPA*: the pesudopolynomial-time algorithm and analysis presented in [8] for handling the general dynamic self-suspension system under fixed-priority scheduling.
- PH-SLM: the pseudopolynomial-time analysis considering jitter, in Section 2.4 in [16], under SLM priority assignment.
- EDAGMF-SLM: Theorem 1 using SLM priority assignment in this paper.
- EDAGMF-OPA: Theorem 1 using OPA priority assignment in this paper.

We generate 100 task sets for each utilization level, from 0.01 to 0.99, in steps of 0.01. The acceptance ratio of a level is said to be the number of task sets that are schedulable divided by the number of task sets for this level, i.e., 100. The metric to compare the results is to measure the *acceptance ratio* of the above tests with respect to a given goal of task set utilization.

We first generated a set of sporadic tasks. The cardinality of the task set was 10. The UUniFast method [3] was adopted to generate a set of utilization values with the given goal. We used the approach suggested by Davis and Burns [6] to generate the task period according to an exponential distribution. The distribution is of two orders of magnitude, i.e., $[10ms - 1000ms]$. The execution time was set accordingly, i.e., $C_i = T_i U_i$. Task relative deadlines are set to their periods, i.e., $D_i = T_i$. We then converted them to self-suspension tasks. Suspension lengths of the tasks were generated similarly to the method in [12]. Suspension lengths of the tasks were generated according to a uniform random distribution, in either of two ranges depending on the self-suspension length (sslen): $[0.01(T_i - C_i), 0.1(T_i - C_i)]$ (short suspension, sslen=S), $[0.1(T_i - C_i), 0.3(T_i - C_i)]$ (medium suspension, sslen=M), and $[0.3(T_i - C_i), 0.6(T_i - C_i)]$ (long suspension, sslen=L).

The number of computation segments m_i was set depending on the following types of self-suspensions: 2 (rare suspension, sstype=R) and 5 (frequent suspension, sstype=F). We then generated every computation segment C_i^j and suspension interval S_i^j with the given C_i and S_i, according to a uniform distribution, like the UUniFast method.

In Figure 2, we show the result for the performance by these tests above in terms of the acceptance ratio, from

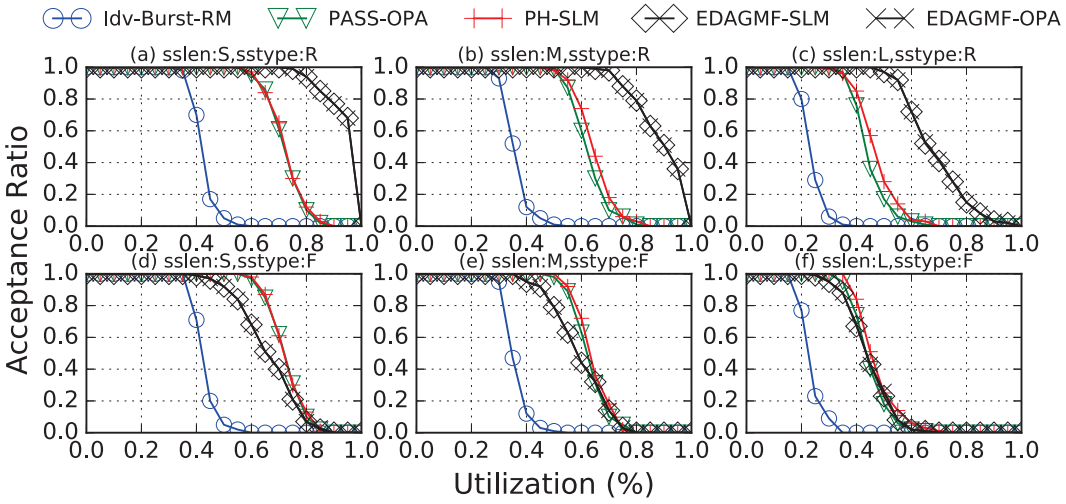

Figure 2: Comparison with different types of suspension lengths (sslength) and different types of suspension frequency (sstype).

different suspension types and lengths. We first notice that the improvement in EDA by using OPA over SLM is very little (almost invisible, but there do exist some task sets schedulable by OPA but not by SLM). The performance by Idv-Burst-RM is inferior to all the others in all the cases. The acceptance ratios by PH-SLM and PASS-OPA are almost identical. The proposed EDAGMF-SLM is far more effective than PASS-OPA and PH-SLM in the case of the rare suspension type (Figure 2a, 2b, and 2c). In the case of the frequent suspension type (Figure 2d, 2e, and 2f), the acceptance by EDAGMF-SLM drops down close to that by PASS-OPA and PH-SLM. Generally speaking, our proposed approach, by using enforcement, is impractical to be adopted for task sets with many suspension intervals (≥ 5); however, it is highly useful for task sets with small numbers of suspension intervals (< 5).

6. CONCLUSION

In this paper, we propose to adopt SLM priority assignment and the FRD scheduler using EDA for scheduling multi-segment self-suspending tasks. This paper *partially solves* this challenging open problem. Although we are not able to provide a constant bound, the speedup factor M^2 in this paper is the only positive and general result. Previously, for this problem, only negative results were known since 2004 [17], and all the factors reported in [17] were ∞ even for one suspension interval. Since there is no lower bound of this problem, further improvement is possible but may require sophisticated approaches. In future work, it is interesting to improve the upper bound of the speedup factor and provide the lower bound.

Although we focus on fixed-priority scheduling, we can conclude that the speedup factor M^2 also holds when we use EDF scheduling under FRD and release time enforcement. This is due to the fact that EDF (under FRD) remains an optimal scheduling policy for GMF tasks, as proved in [2]. Moreover, our analysis uses a necessary condition (Lemma 4) of any arbitrary feasible schedule when analyzing the processor speedup factor. As a result, this paper also provides generalized results of [5] for dynamic-priority scheduling.

7. ACKNOWLEDGMENTS

This paper is supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/).

8. REFERENCES

[1] N. C. Audsley. *Optimal priority assignment and feasibility of static priority tasks with arbitrary start times*. Citeseer, 1991.
[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):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, N. Audsley, W.-H. Huang, J.-J. Chen, and G. Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. Technical report, CISTER-TR-150713, 2015.
[5] J.-J. Chen and C. Liu. Fixed-relative-deadline scheduling of hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium (RTSS)*, 2014.
[6] 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] 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.
[8] W.-H. Hung, J.-J. Chen, H. Zhou, and C. Liu. PASS: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *Design Automation Conference (DAC)*, 2015.
[9] J. Kim, B. Andersson, D. d. Niz, and R. R. Rajkumar. Segment-fixed priority scheduling for self-suspending real-time tasks. In *Real-Time Systems Symposium (RTSS)*, pages 246–257, 2013.
[10] K. Lakshmanan and R. Rajkumar. Scheduling self-suspending real-time tasks with rate-monotonic priorities. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 3–12, 2010.
[11] C. Liu and J. H. Anderson. Supporting sporadic pipelined tasks with early-releasing in soft real-time multiprocessor systems. In *RTCSA*, pages 284–293, 2009.
[12] C. Liu and J. H. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Real-Time Systems Symposium*, pages 425–436, 2009.
[13] C. Liu and J. Chen. Bursty-interference analysis techniques for analyzing complex real-time task models. In *Real-Time Systems Symposium (RTSS)*, pages 173–183, 2014.
[14] J. W. Liu. *Real-time systems. 2000*. Prentice Hall.
[15] 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)*, 2015.
[16] J. C. Palencia and M. Gonzalez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Real-Time Systems Symposium*, pages 26–37, 1998.
[17] 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.
[18] H. Takada and K. Sakamura. Schedulability of generalized multiframe task sets under static priority assignment. In *RTCSA*, pages 80–86, 1997.