A Unifying Response Time Analysis Framework for Self-Suspending Tasks

Jian-Jia Chen*, Geoffrey Nelissen†, Wen-Hung Huang*
* TU Dortmund University, Germany
Emails: {jian-jia.chen, wen-hung.huang}@tu-dortmund.de
† CISTER/INESC-TEC, ISEP, Polytechnic Institute of Porto, Portugal
Email: grrpn@isep.ipp.pt

***Abstract*—**

1 Introduction

The periodic/sporadic task model has been recognized as the basic model for real-time systems with recurring executions. A sporadic real-time task τ_i is characterized by its *worst-case execution time* C_i, its *minimum inter-arrival time* T_i and its *relative deadline* D_i. A sporadic task defines an infinite sequence of task instances, also called *jobs*, that arrive with the minimum inter-arrival time constraint. When a job of task τ_i arrives at time t, the job should finish no later than its *absolute deadline* $t + D_i$, and the next job of task τ_i can only be released no earlier than $t + T_i$. For the periodic task model, the next job is released at time $t + T_i$, in which T_i is also referred to as the *period* of task τ_i.

The seminal work by Liu and Layland [14] considered the scheduling of periodic tasks and presented the schedulability analyses based on utilization bounds to verify whether the deadlines are met or not. For over decades, researchers in real-time systems have devoted themselves to effective design and efficient analyses of different recurrent task models to ensure that tasks can meet their specified deadlines. In most of these studies, *a task usually does not suspend itself*. That is, after a job is released, the job is either executed or stays in the ready queue, but it is not moved to the suspension state. Such an assumption is valid only under the following conditions: (1) the latency of the memory accesses and I/O peripherals is considered to be part of the worst-case execution time of a job, (2) there is no external device for accelerating the computation, and (3) there is no synchronization between different tasks on different processors in a multiprocessor or distributed computing platform.

If a job can suspend itself before it finishes its computation, self-suspension behaviour has to be considered. Due to the interaction with other system components and synchronization, self-suspension behaviour has become more visible in designing real-time embedded systems. Typically, the resulting suspension delays range from a few microseconds (e.g., a write operation on a flash drive [9]) to a few hundreds of milliseconds (e.g., offloading computation to GPUs [10], [16]).

There are two typical models for self-suspending sporadic task systems: 1) the dynamic self-suspension task model, and 2) the segmented self-suspension task model. In the *dynamic* self-suspension task model, in addition the worst-case execution time C_i of sporadic task τ_i, we have also the worst-case self-suspension time S_i of task τ_i. In the *segmented* self-suspension task model, the execution behaviour of a job of task τ_i is specified by interleaved computation segments and self-suspension intervals. From the system designer's perspective, the dynamic self-suspension model provides a simple specification by ignoring the juncture of I/O access, computation offloading, or synchronization. However, if the suspending behaviour can be characterized by using a segmented pattern, the segmented self-suspension task model can be more appropriate.

In this paper, we focus on preemptive fixed-priority scheduling for the dynamic self-suspension task model on a uniprocessor platform. To verify the schedulability of a given task set, this problem has been specifically studied in [1], [2], [8], [12], [17]. The recent report by Chen et al. and the report by Bletsas et al. [3] have shown that the analysis by introducing the suspension time of a higher-priority task as its arrival jitter in [1], [2], [12], [17] is unsafe. This misconception was unfortunately adopted in [4], [5], [7], [11], [13], [20]–[22] to analyze the worst-case response time for partitioned multiprocessor real-time locking protocols.

Moreover, one concept to consider suspension-time as blocking time was used by Jane W. S. Liu in her book titled "Real-Time Systems" [15, Pages 164-165], and was also implicitly used by Rajkumar, Sha, and Lehoczky [19, Page 267] for analyzing the self-suspending behaviour due to synchronization protocols in multiprocessor systems. However, there is no proof in [15], [19] to support the correctness of the provided schedulability tests.

The contributions of this paper are as follows:

- We provide a general analysis framework in Theorem 1 for dynamic self-suspending sporadic real-time tasks on a uniprocessor platform. This theorem analytically dominates all the existing results in [3], [8] and [15, Pages 164-165], excluding the flawed ones. The key observation in the analysis framework is that the *interference from higher-priority self-suspending tasks can be arbitrarily modelled as jitter or carry-in terms*. Moreover, the proof of Theorem 1 also supports the correctness of the analysis in [15, Pages 164-165] and [19, Page 267].[1]
- We develop a few strategies to decide which higher-priority tasks should be classified to associate with jitter

[1] A simplified version of the proof of Theorem 1 to support the correctness of [15, Pages 164-165] and [19, Page 267] is provided in [6].

terms and which higher-priority tasks should be classified to associate with carry-in terms. The methods are presented in Section **??**.
- utilization bounds..
- evaluation results...

2 Task Model

We assume a system τ composed of n sporadic self-suspending tasks. A sporadic task τ_i is released repeatedly, with each such invocation called a job. The j^{th} job of τ_i, denoted by $\tau_{i,j}$, is released at time $r_{i,j}$ and has an absolute deadline at time $d_{i,j}$. Each job of task τ_i is assumed to have a worst-case execution time C_i. Furthermore, a job of task τ_i may suspend itself for at most S_i time units (across all of its suspension phases). When a job suspends itself, it releases the processor and another job can be executed. The response time of a job is defined as its finishing time minus its release time. Successive jobs of the same task are required to execute in sequence.

Associated with each task τ_i are a period (or minimum inter-arrival time) T_i, which specifies the minimum time between two consecutive job releases of τ_i, and a relative deadline D_i, which specifies the maximum amount of time a job can take to complete its execution after its release. It results that for each job $\tau_{i,j}$, there is $d_{i,j} = r_{i,j} + D_i$ and $r_{i,j+1} \geq r_{i,j} + T_i$. In this paper, we focus on constrained-deadline tasks, for which $D_i \leq T_i$. The utilization of a task τ_i is defined as $U_i = C_i/T_i$.

The worst-case response time R_i of a task τ_i is the maximum response time among all its jobs. A schedulability test for a task τ_k is therefore to verify whether its worst-case response time is no more than its associated relative deadline D_k.

In this paper, we only consider *preemptive fixed-priority scheduling running on a single processor platform*, in which each task is assigned with a unique priority level. We assume that the priority assignment is given beforehand and that the tasks are numbered in a decreasing priority order. That is, a task with a smaller index has a higher priority than any task with a higher index, i.e., task τ_i has a higher-priority than task τ_j if $i < j$.

When performing the schedulability analysis of a specific task τ_k, we will implicitly assume that all the higher priority tasks (i.e., $\tau_1, \tau_2, \ldots, \tau_{k-1}$) are already verified to meet their deadlines, i.e., that $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$.

3 Background

To analyze the worst-case response time (or the schedulability) of a task τ_k, one usually needs to quantify the worst-case interference exerted by the higher-priority tasks on the execution of any job of task τ_k. In the ordinary sequential sporadic real-time task model, i.e., when $S_i = 0$ for every task τ_i, the so-called critical instant theorem by Liu and Layland [14] is commonly adopted. That is, the worst-case response time of task τ_k (if it is less than or equal to its period) happens for the first job of task τ_k when (i) τ_k and all the higher-priority tasks release their first job synchronously and (ii) all their subsequent jobs are released as early as possible (i.e., with a rate equal to their period). However, this definition of the critical instant does not hold for self-suspending sporadic tasks.

There exist three different approaches in the state-of-the-art that are potentially sound to perform the schedulability analysis of self-suspending tasks:

- modeling the suspension as execution, also known as the suspension-oblivious analysis (see Section 3.1);
- modeling the suspension as a release jitter (see Section 3.2);
- modeling the suspension as blocking time (see Section 3.3).

We later prove in Section 6 that all these approaches are analytically correct.

3.1 Suspension-Oblivious Analysis

The simplest analysis consists in converting the suspension time S_i of each task τ_i as a part of its computation time. Therefore, a constrained-deadline task τ_k can be feasibly scheduled by a fixed-priority scheduling algorithm if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \leq t. \quad (1)$$

3.2 Modeling the Suspension as a Release Jitter

Another approach consists in modeling the impact of the self-suspension S_i of each higher priority task τ_i as a release jitter J_i. Several works in the state-of-the-art [1], [2], [12], [17] upper bounded J_i with S_i. However, it has been recently shown in [3] that this upper bound is unsafe and J_i can in fact be larger than S_i.

Nevertheless, it was proven in the same document [3] that the jitter of a higher-priority task τ_i can be safely upper bounded by $R_i - C_i$. It results that a task τ_k with a constrained deadline can be feasibly scheduled under fixed-priority if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + R_i - C_i}{T_i} \right\rceil C_i \leq t. \quad (2)$$

3.3 Modeling the Suspension as Blocking Time

In [15, p. 164-165], Liu proposed a solution to study the schedulability of a self-suspending task τ_k by modeling the extra delay suffered by τ_k due to the self-suspension behavior of each task in τ as a blocking time. This blocking time has been defined as follows:

- The blocking time contributed from task τ_k is S_k.
- A higher-priority task τ_i can block the execution of task τ_k for at most $\min(C_i, S_i)$ time units.

An upper bound on the blocking time is therefore given by:

$$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i). \quad (3)$$

In [15], the blocking time is then used to derive a utilization-based schedulability test for rate-monotonic scheduling. Namely, it is stated that, if $T_i = D_i$ for every task $\tau_i \in \tau$ and $\frac{C_k+B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$, then τ_k can be feasibly scheduled with rate-monotonic.

The same concept was also implicitly used by Rajkumar, Sha, and Lehoczky in [19, p. 267] for analyzing the impact of the self-suspendion of a task due to the utilization of synchronization protocols in multiprocessor systems. The statement in [19] reads as follows:[2]

> *"For each higher priority job $\tau_{i,j}$ that suspends on global semaphores or for other reasons, add the term $\min(C_i, S_i)$ to B_k, where S_i is the maximum duration that $\tau_{i,j}$ can suspend itself. [...] The sum [...] yields B_k, which in turn can be used in $\frac{C_k+B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$ to determine whether the current task allocation to the processor is schedulable."*

If the above argument is correct, we can further prove that a constrained-deadline task τ_k can be feasibly scheduled under fixed-priority scheduling if

$$\exists t \mid 0 < t \leq D_k, \quad C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t. \tag{4}$$

However, there is no proof in [15] nor in [19] to support the correctness of those tests. Therefore, in Section 6, we provide a proof (see Theorem 3) of the correctness of Equation (4).

4 Rationale

Even though it can be proven that the response time analysis associated with Eq.(4) dominates the suspension oblivious one (see Lemma 3 in Section 6), none of the analyses presented in Section 3 dominates all the others. Hence, Eq. (4) and 2 are incomparable. That is, in some cases Eq. (4) performs better than Eq. (2), while in others Eq. (2) outperforms Eq. (4).

Example 1. *Consider the two tasks $\tau_1 = (1, 8, 10, 10)$ and $\tau_2 = (1, 0, 15, 15)$. The worst-case response time of τ_1 is obviously 9 whatever the analysis employed. However, the upper bound on the WCRT of τ_2 obtained with Eq. (2) is 2, while the upper bound on the WCRT of τ_2 obtained with Eq. (4) is 3.*

Example 2. *Consider the three tasks $\tau_1 = (xxx)$, $\tau_2 = (yyyy)$ and $\tau_3 = (zzz)$. With both Eqs (2) and (4), the upper bounds computed on the WCRT of τ_1 and τ_2 are x and y, respectively.*

Using Eq. (2), the worst-case response time of task τ_3 is upper bounded by z. However, with Eq. (4), the computed upper bound on the WCRT of τ_3 turns out to be v. Therefore, in this example, the upper bound on the WCRT of τ_3 computed with Eq. (4) is smaller than the value obtained with Eq. (2), while the opposite was true in Example 1.

Similarly, the analysis associated with Eq. (2) cannot be compared with the suspension oblivious approach.

Example 3. *An example where Eq. (2) performs better than oblivious.*

[2] We rephrased the wording and notations in order to be consistent with the rest of this paper.

Example 4. *An example where oblivious performs better than Eq. (2)*

In this paper, we derive a response time analysis that draws inspiration from both Eq. (2) and Eq. (4), combining the best of each of them. As further proven in Section 6, the resulting schedulability test dominates all the tests discussed in Section 3.

5 A Unifying Analysis Framework

5.1 A new Response time Analysis

We can greedily convert the suspension time of task τ_k to its computation time. For the sake of notational brevity, let C'_k be $C_k + S_k$. We call this converted version of task τ_k as task τ'_k. Suppose that R'_k is the worst-case response time in the task system $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k\}$. It was already shown in the previous works, e.g., Lemma 3 in [16] and Theorem 2 in [18], that R'_k is a safe upper bound on the worst-case response time of task τ_k in the original task system.

Note that for the rest of this section we implicitly assume that $R_i \leq D_i, \forall \tau_i \mid 1 \leq i \leq k-1$. Our key result in this paper is the following theorem:

Theorem 1. *Suppose that $R_k \leq T_k$. For any arbitrary vector assignment $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$, in which x_i is either 0 or 1, the worst-case response time R_k is upper bounded by the minimum t (with $t > 0$) that satisfies*

$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \leq t, \tag{5}$$

where $Q_i^{\vec{x}}$ is $\sum_{j=i}^{k-1} S_j \cdot x_j$.

We will explain the resulting properties from Theorem 1 first, by leaving the proof to Section 5.2 since it is pretty long. With Theorem 1, we can directly have the following corollary.

Corollary 1. *If there is a vector $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$ with $x_i \in \{0, 1\}$, such that*

$$\exists t | 0 < t \leq D_k,$$
$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil C_i \leq t \tag{6}$$

where $Q_i^{\vec{x}} \stackrel{\text{def}}{=} \sum_{j=i}^{k-1} S_j \cdot x_j$, then a constrained-deadline task τ_k is schedulable under fixed-priority.

We will later prove in Section 6, that Corollary 1 in fact dominates all the analyses discussed in Section 3.

Example 5. *We use an example to demonstrate how Corollary 1 can be applied. Suppose that we have three tasks*

- $C_1 = 4, S_1 = 5, T_1 = D_1 = 10,$
- $C_2 = 6, S_2 = 1, T_2 = D_2 = 19,$ *and*
- $C_3 = 4, S_3 = 0, T_3 = D_3 = 35.$

Tasks τ_1 and τ_2 can be verified to be schedulable under the fixed-priority scheduling by using Eq. (2).

We focus on task τ_3. For task τ_3, the blocking term B_3 is $4 + 1 = 5$ by Eq. (3). The minimum t to satisfy $C_k +$

$B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t$ *happens when* $t = 37$, *i.e.,* $4 + 5 + \lceil \frac{37}{10} \rceil 4 + \lceil \frac{37}{19} \rceil 6 = 37$. *Therefore, task* τ_3 *cannot pass the schedulability test in Eq. (4). There are four possible vector assignments* $\vec{x}$ *when we consider the schedulability of task* τ_3 *with Corollary 1:*

- *Case 1.* $\vec{x} = (0, 0)$*: In this case, Theorem 1 states that* R'_k *is upper bounded by the minimum* t *under* $0 < t \leq T_3$ *that* $4 + \lceil \frac{t+6}{10} \rceil 4 + \lceil \frac{t+13}{19} \rceil 6 \leq t$. *Such* t *does not exist for this case.*
- *Case 2.* $\vec{x} = (0, 1)$*: In this case, Theorem 1 states that* R'_k *is upper bounded by the minimum* t *under* $0 < t \leq T_3$ *that satisfies* $4 + \lceil \frac{t+7}{10} \rceil 4 + \lceil \frac{t+1}{19} \rceil 6 \leq t$. *Therefore,* $R'_k \leq 32$ *due to* $4 + \lceil \frac{32+7}{10} \rceil 4 + \lceil \frac{32+1}{19} \rceil 6 = 32$.
- *Case 3.* $\vec{x} = (1, 0)$*: In this case, there is no value of* t *that satisfies Eq.* (5).
- *Case 4.* $\vec{x} = (1, 1)$*: In this case, Theorem 1 states that* R'_k *is upper bounded by the minimum* t *under* $0 < t \leq T_3$ *that satisfies* $4 + \lceil \frac{t+6}{10} \rceil \cdot 4 + \lceil \frac{t+1}{19} \rceil \cdot 6 \leq t$. *Therefore,* $R'_k \leq 32$ *due to* $4 + \lceil \frac{32+6}{10} \rceil 4 + \lceil \frac{32+1}{19} \rceil 6 = 32$.

Among the above four cases, the tests in Cases 2 and 4 are tighter. By Corollary 1, task τ_3 *is schedulable under fixed-priority.*

5.2 Proof of Correctness

We now provide the proof to support the correctness of the test in Theorem 1. Our proof strategy is to show that the worst-case response time of task τ_k can be safely upper-bounded by any assignment of $\vec{x}$ of the $k-1$ higher-priority tasks when adopting Eq. (5) as the response time analysis.

Throughout the proof, we consider any arbitrary assignment $\vec{x}$, in which x_i is either 0 or 1. For the sake of notational brevity, we classify the $k-1$ higher-priority tasks into two sets: $\mathbf{T}_0$ and $\mathbf{T}_1$. A task τ_i is in $\mathbf{T}_0$ if x_i is 0; otherwise, it is in $\mathbf{T}_1$.

Our analysis is also based on very simple properties and lemmas enunciated as follows:

Property 1. *In a preemptive fixed-priority schedule, the lower-priority jobs do not impact the schedule of the higher-priority jobs.*

Lemma 1. *In a preemptive fixed-priority schedule, if the worst-case response time of task* τ_i *is no more than its period* T_i*, preventing the release of a job of task* τ_i *does not affect the schedule of any other job of task* τ_i*.*

Proof: Since the worst-case response time of task τ_i is no more than its period, any job $\tau_{i,j}$ of task τ_i completes its execution before the release of the next job $\tau_{i,j+1}$. Hence, the execution of $\tau_{i,j}$ does not directly interfere with the execution of any other job of τ_i, which then depends only on the schedule of the higher priority jobs. Furthermore, as stated in Property 1, the removal of $\tau_{i,j}$ has no impact on the schedule of the higher-priority jobs, thereby implying that the other jobs of task τ_i are not affected by the removal of $\tau_{i,j}$. ■

With the above properties, we can present the detailed proof of Theorem 1. However, the proof involves several transformation steps. To illustrate some important steps in the proof, we also provide one concrete example. Consider a task system with the following 4 tasks:

- $T_1 = 6, C_1 = 1, S_1 = 1, x_1 = 1,$
- $T_2 = 10, C_2 = 1, S_2 = 6, x_2 = 0,$
- $T_3 = 18, C_3 = 4, S_3 = 1, x_3 = 1,$
- $T_4 = 20, C_4 = 5, S_4 = 0.$

Figure 1 demonstrates a schedule for the jobs of the above 4 tasks. We assume that the first job of task τ_1 arrives at time $4+\epsilon$ with a very small $\epsilon > 0$. The first job of task τ_2 suspends itself from time 0 to time $5+\epsilon$, and is blocked by task τ_1 from time $5+\epsilon$ to time $6+\epsilon$. After some very short computation with ϵ amount of time, the first job of task τ_2 suspends itself again from time $6+2\epsilon$ to 7.

Proof of Theorem 1. Let us consider the task set τ' composed of $\{\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k, \tau_{k+1}, \ldots\}$ and let Ψ be a schedule of τ', in which $R'_k \leq T_k$ by our assumption. Suppose that a job J_k of task τ'_k arrives at time r_k and finishes at time f_k. We will prove that the response time analysis in Eq. (5) gives us a safe upper bound on $f_k - r_k$ for any job J_k in Ψ.

The proof is built upon the three following steps:

1) We discard all the jobs that do not contribute to the response time of J_k in the schedule Ψ. We follow an inductive strategy by iteratively inspecting the schedule of the higher priority tasks in Ψ, starting with τ_{k-1} until the highest priority task τ_1. At each iteration, a time instant t_j is identified such that $t_j \leq t_{j+1}$ $(1 \leq j < k)$. Then, all the jobs of task τ_j released before t_j are removed from the schedule and, if needed, replaced by an artificial job mimicking the interference caused by the residual workload of task τ_j at time t_j on the response time of job J_k.
2) The final reduced schedule is analyzed so as to characterize the worst-case response time of τ'_k in Ψ by using workload functions.
3) We then prove that the response time analysis in Eq. (5) is indeed an upper bound on the worst-case response time R'_k of τ'_k.

Step 1: Reducing the schedule Ψ

During this step, we iteratively build an artificial schedule Ψ^j from Ψ^{j+1} (with $1 \leq j < k$) so that the response time of τ'_k remains identical. At each iteration, we define t_j for task τ_j in the schedule Ψ^{j+1} (with $j = k-1, k-2, \ldots, 1$) and build Ψ^j by removing all the jobs released by τ_j before t_j.

Basic step (definition of Ψ^k *and* t_k*):*

Recall that the job J_k of task τ'_k arrives at time r_k and finishes at time f_k in schedule Ψ. We know by Property 1 that the lower priority tasks $\tau_{k+1}, \tau_{k+2}, \ldots, \tau_n$ do not impact the response time of J_k. Moreover, since we assume that the worst-case response time of task τ'_k is no more than T_k, Lemma 1 proves that removing all the jobs of task τ'_k but J_k has no impact on the schedule of J_k. Therefore, let Ψ^k be a schedule identical to Ψ but removing all the jobs released by the lower priority tasks $\tau_{k+1}, \ldots, \tau_n$ as well as all the jobs released by

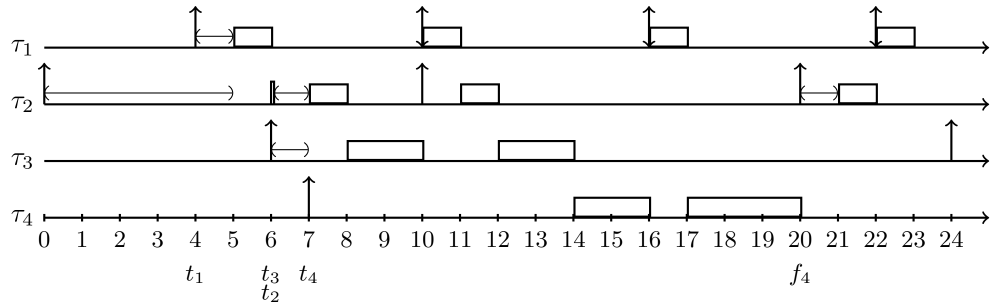

Fig. 1: An illustrative example of Step 1 in the proof of Theorem 1.

τ'_k at the exception of J_k. The response time of J_k in Ψ^k is thus identical to the response time of J_k in Ψ.

We define t_k as the release time of J_k (i.e., $t_k = r_k$).

Induction step (definition of Ψ^j and t_j with $1 \leq j < k$):

Let r_j be the arrival time of the last job released by τ_j before t_{j+1} in Ψ^{j+1} and let J_j denote that job. Removing all the jobs of task τ_j arrived before r_j has no impact on the schedule of any other job released by τ_j (Lemma 1) or any higher priority job released by $\tau_1, \ldots, \tau_{j-1}$ (Property 1). Moreover, because by the construction of Ψ^{j+1}, no task with a priority lower than τ_j executes jobs before t_{j+1} in Ψ^{j+1}, removing the jobs released by τ_j before t_{j+1} does not impact the schedule of the jobs of $\tau_{j+1}, \ldots, \tau_k$. Therefore, we can safely remove all the jobs of task τ_j arrived before r_j without impacting the response time of J_k. Two cases must then be considered:

(a) $\tau_j \in \mathbf{T}_1$. In this case, we analyze two different subcases:
- J_j completed its execution before or at t_{j+1}. By Lemma 1 and Property 1, removing all the jobs of task τ_j arrived before t_{j+1} has no impact on the schedule of the higher-priority jobs (jobs released by $\tau_1, \ldots, \tau_{j-1}$) and the jobs of τ_j released after or at t_{j+1}. Moreover, because no task with lower priority than τ_j executes jobs before t_{j+1} in Ψ^{j+1}, removing the jobs released by τ_j before t_{j+1} does not impact the schedule of the jobs of $\tau_{j+1}, \ldots, \tau_k$. Therefore, t_j is set to t_{j+1} and Ψ^j is generated by removing all the jobs of task τ_j arrived before t_{j+1}. The response time of J_k in Ψ^j thus remains unchanged in comparison to its response time in Ψ^{j+1}.
- J_j did not complete its execution by t_{j+1}. For such a case, t_j is set to r_j and hence Ψ^j is built from Ψ^{j+1} by removing all the jobs released by τ_j before r_j.

Note that because by the construction of Ψ^{j+1} and hence Ψ^j there is no job with priority lower than τ_j available to be executed before t_{j+1}, the maximum amount of time during which the processor remains idle within $[t_j, t_{j+1})$ is at most S_j time units.

(b) $\tau_j \in \mathbf{T}_0$. For such a case, we set t_j to t_{j+1}. Let c^*_j be the remaining execution time for the job of task τ_j at time t_j. We know that c^*_j is at most C_j. Since by the construction of Ψ^j, all the jobs of τ_j released before t_j are removed, the job of task τ_j arrived at time r_j $(< t_j)$ is replaced by a new job released at time t_j with execution time c^*_j and the same priority than τ_j. Clearly, this has no impact on the execution of any job executed after t_j and thus on the response time of J_k. The remaining execution time c^*_j of τ_j at time t_j is called the *residual workload* of task τ_j for the rest of the proof.

The above construction of $\Psi^{k-1}, \Psi^{k-2}, \ldots, \Psi^1$ is repeated until producing Ψ^1. The procedures are well-defined. Therefore, it is guaranteed that Ψ^1 can be constructed. Note that after each iteration, the number of jobs considered in the schedule have been reduced, yet without affecting the response time of J_k.

An example of the procedures in Step 1: In this schedule illustrated in Figure 1, f_k is set to $20 - \epsilon$. We define t_4 as 7. Then, we set t_3 to 6. When considering task τ_2, since it belongs to $\mathbf{T}_0$, we greedily set t_2 to $t_3 = 6$ and the residual workload c^*_2 is 1. Then, t_1 is set to $4 + \epsilon$. In the above schedule, the idle time from $4 + \epsilon$ to $20 - \epsilon$ is at most $2 = S_1 + S_3$. We have to further consider one job of task τ_2 arrived before time t_1 with execution time C_2.

Step 2: Analyzing the final reduced schedule Ψ^1

We now analyze the properties of the final schedule Ψ^1 in which all the unnecessary jobs have been removed. The proof is based on the fact that for any interval $[t_1, t)$, there is

$$\mathrm{idle}(t_1, t) + \mathrm{exec}(t_1, t) = (t - t_1) \tag{7}$$

where $\mathrm{exec}(t_1, t)$ is the amount of time during which the processor executed tasks within $[t_1, t)$, and $\mathrm{idle}(t_1, t)$ is the amount of time during which the processor remained idle within the interval $[t_1, t)$.

If $t_i < t_{i+1}$, the processor may idle in the time interval $[t_i, t_{i+1})$ in Ψ^1. Suppose that σ_i is the sum of the idle time in this interval $[t_i, t_{i+1})$ in Ψ^1. If t_i is equal to t_{i+1}, then σ_i is set to 0. Therefore, we have

$$\mathrm{idle}(t_1, t) \leq \sum_{i: t_i < t} \sigma_i. \tag{8}$$

From case (a) of Step 1, we know that $\sigma_i \leq S_i$.

Because there is no job released by lower priority tasks than τ'_k in Ψ^1, we only focus on the execution patterns of the tasks $(\tau_1, \tau_2, \ldots, \tau_{k-1}, \tau'_k)$. According to Step 1, we should consider two cases:

- If task τ_j is in $\mathbf{T}_1$, there is no job of task τ_j arrived before t_j in Ψ^1. This corresponds to both subcases in case (a) in

Step 1. In this case, for any $\Delta \geq 0$, the workload, defined as $W_j^1(\Delta)$, contributed from task τ_j from t_j to $t_j + \Delta$ that is executed on the processor is at most

$$W_j^1(\Delta) = \left\lfloor \frac{\Delta}{T_j} \right\rfloor C_j + \min\left\{\Delta - \left\lfloor \frac{\Delta}{T_j} \right\rfloor T_j, C_j\right\}. \quad (9)$$

- If task τ_j is in $\mathbf{T}_0$, there may be a job arrived before t_j with residual workload c_j^* at time t_j. This corresponds to case (b) in Step 1. In this case, for any $\Delta \geq 0$, the workload, defined as $\widehat{W}_j^0(\Delta, c_j^*)$, contributed from task τ_j from t_j to $t_j + \Delta$ has to consider two subcases:
  - If the residual workload c_j^* of task τ_j is 0, the earliest arrival time of task τ_j can be any time point at or after t_j. In this case, for any $\Delta \geq 0$, the workload contributed from task τ_j from t_j to $t_j + \Delta$ that is executed on the processor is at most
  $$\widehat{W}_j^0(\Delta, 0) = W_j^1(\Delta). \quad (10)$$
  - If the residual workload c_j^* of task τ_j is positive, the absolute deadline of the job corresponding to the residual workload must be at least $t_j + c_j^*$; otherwise, the job corresponding to the residual workload would miss its deadline. Therefore, the earliest arrival time of task τ_j arriving *strictly after* t_j is at least $t_j + (T_j - D_j + c_j^*)$ in Ψ^1. For notational brevity, let ρ_j be $(T_j - D_j + c_j^*)$. In this case, for any $\Delta \geq 0$ and $c_j^* > 0$, the workload contributed from task τ_j from t_j to $t_j + \Delta$ that is executed on the processor is at most
  $$\widehat{W}_j^0(\Delta, c_j^*) = \begin{cases} \Delta & \text{if } \Delta \leq c_j^* \\ c_j^* & \text{if } c_j^* < \Delta \leq \rho_j \\ c_j^* + W_j^1(\Delta - \rho_j) & \text{otherwise.} \end{cases} \quad (11)$$

  It is proved in Lemma 2 that the worst case residual workload in $\widehat{W}_j^0(\Delta, c_j^*)$ by considering both Eq. (10) and Eq. (11) is to have $c_j^* = C_j$, i.e., for all $\Delta \geq 0$, we have $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$. For the sake of notational brevity, let
  $$W_j^0(\Delta) \stackrel{\text{def}}{=} \widehat{W}_j^0(\Delta, C_j) \quad (12)$$

Putting the execution time from the tasks in $\mathbf{T}_0$ and $\mathbf{T}_1$ together, we have for $i = 2, 3, \ldots, k-1$, $\forall t \mid t_{i-1} \leq t < t_i$

$$\text{exec}(t_1, t) \leq \sum_{j=1}^{i-1} x_j \cdot W_j^1(t - t_j) + (1 - x_j) \cdot W_j^0(t - t_j). \quad (13)$$

Putting Eqs. (7), (8), (13) together, we have for $i = 2, 3, \ldots, k-1$, $\forall t \mid t_{i-1} \leq t < t_i$

$$\sum_{j=1}^{i-1} x_j \cdot (W_j^1(t-t_j)+\sigma_j)+(1-x_j)\cdot W_j^0(t-t_j) \geq t-t_1. \quad (14)$$

Moreover, $\forall t \mid t_k \leq t < f_k$, we have

$$C_k' + \sum_{j=1}^{k-1} x_j \cdot (W_j^1(t-t_j)+\sigma_j)+(1-x_j)\cdot W_j^0(t-t_j) > t-t_1. \quad (15)$$

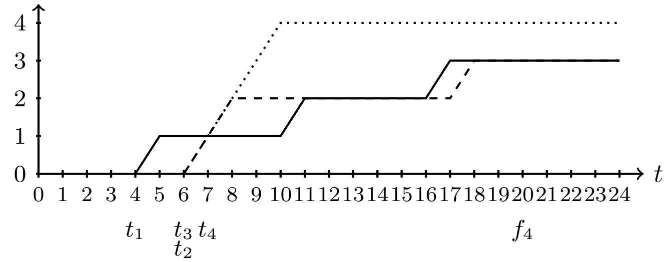

Fig. 2: The workload function for the three higher-priority tasks in Figure 2. Solid line: $W_1^1(t - t_1)$, Dashed line: $W_2^0(t - t_2)$, Dotted line: $W_3^1(t - t_3)$, where the functions are 0 if $t - t_j < 0$ for $j = 1, 2, 3$.

An example of the procedures in Step 2: In the example used in Figure 1, we have $\sigma_1 = 1, \sigma_2 = 0$, and $\sigma_3 = 1$. The corresponding functions $W_1^1(t - t_1)$, $W_2^0(t - t_2)$, $W_3^1(t - t_3)$ are illustrated in Figure 2. Therefore, it is rather clear that all the conditions in Eq. (14) and Eq. (15) hold by simple arithmetics.

Step 3: Creating Safe Response-Time Analysis

This step constructs a safe response-time analysis based on the conditions in Eqs. (14) and (15). We will construct another release pattern which moves t_i to t_i^* for $i = 2, 3, \ldots, k$ such that $t_i^* \leq t_i$ and the corresponding conditions in Eqs. (14) and (15) will become worse when we use t_i^*. We start the procedure as follows:

- Initial Step: Let t_1^* be t_1.
- Iterative steps ($i = 2, 3, \ldots, k$): Let t_i^* be $t_{i-1}^* + x_{i-1} \cdot \sigma_{i-1}$.

This results in $t_i^* \leq t_i$ for $i = 2, 3, \ldots, k$. Moreover, by definition, t_j^* is $t_1^* + \sum_{i=1}^{j-1} x_i \cdot \sigma_i$ for $j = 2, 3, \ldots, k$. For any task τ_j in $\mathbf{T}_1$, $\forall \Delta \geq 0$, since $t_j \geq t_j^*$, we have

$$W_j^1(\Delta) \leq W_j^1(\Delta + (t_j - t_j^*)). \quad (16)$$

For any task τ_j in $\mathbf{T}_0$, $\forall \Delta \geq 0$, since $t_j \geq t_j^*$, we have

$$W_j^0(\Delta) \leq W_j^0(\Delta + (t_j - t_j^*)). \quad (17)$$

Therefore, for any $j = 1, 2, \ldots, k-1$, the contribution $W_j^1(t - t_j) \leq W_j^1(t - t_j^*)$ and $W_j^0(t - t_j) \leq W_j^0(t - t_j^*)$ for any $t \geq t_j$. Putting these into Eqs. (14) $\forall t \mid t_k^* \leq t < t_k$ leads to

$$\sum_{j=1}^{k-1} x_j \cdot (W_j^1(t - t_j^*) + \sigma_j) + (1 - x_j) \cdot W_j^0(t - t_j^*) \geq t - t_1,$$
$$\Rightarrow \sum_{j=1}^{k-1} x_j \cdot W_j^1(t - t_j^*) + (1 - x_j) \cdot W_j^0(t - t_j^*) \geq t - t_k^*. \quad (18)$$

Similarly, putting these into Eqs. (15) $\forall t \mid t_k \leq t < f_k$ leads to

$$C_k' + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t-t_j^*)+(1-x_j)\cdot W_j^0(t-t_j^*) > t-t_k^*. \quad (19)$$

By the assumption that $C_k' \geq C_k > 0$, we can unify the above inequalities in Eq. (18) and Eq. (19) as follows: $\forall t \mid t_k^* \leq t <$

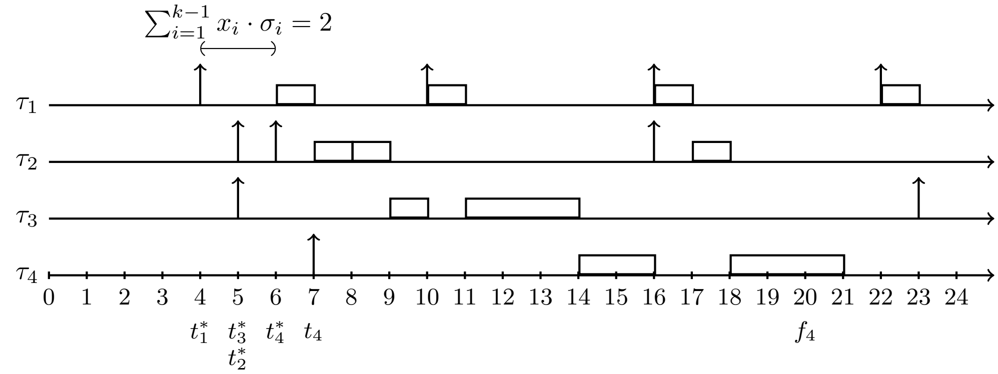

Fig. 3: An illustrative example of Step 3 in the proof of Theorem 1 based on an *imaginary* schedule.

f_k

$$C'_k + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t-t_j^*) + (1-x_j) \cdot W_j^0(t-t_j^*) > t - t_k^*. \quad (20)$$

By definition, $\forall t \mid t_k^* \leq t < f_k$, we have $t - t_j^* = t - t_k^* + \sum_{\ell=j}^{k-1} x_\ell \sigma_\ell$ for every $j = 1, 2, \ldots, k-1$. Therefore, we know that $W_j^1(t - t_j^*) \leq \left\lceil \frac{t - t_k^* + \sum_{\ell=j}^{k-1} x_\ell \sigma_\ell}{T_j} \right\rceil C_j$ for task τ_j in $\mathbf{T}_1$. Moreover, $\forall t \mid t_k^* \leq t < f_k$, we have $W_j^0(t - t_j^*) \leq \left\lceil \frac{t - t_k^* + \sum_{\ell=j}^{k-1} x_\ell \sigma_\ell + (1-x_j)(D_j - C_j)}{T_j} \right\rceil C_j$ for task τ_j in $\mathbf{T}_0$. Therefore, we can conclude that $\forall t \mid t_k^* \leq t < f_k$

$$C'_k + \sum_{j=1}^{k-1} \left\lceil \frac{t - t_k^* + X_j + (1-x_j)(D_j - C_j)}{T_j} \right\rceil C_j > t - t_k^*, \quad (21)$$

where X_j is $\sum_{\ell=j}^{k-1} x_\ell \sigma_\ell$. We replace $t - t_k^*$ with θ. The above inequation implies that the minimum θ with $\theta > 0$ such that $C'_k + \sum_{j=1}^{k-1} \left\lceil \frac{\theta + X_j + (1-x_j)(D_j - C_j)}{T_j} \right\rceil C_j = \theta$ is larger than or equal to $f_k - t_k^* \geq f_k - t_k$.

However, the above condition requires the knowledge of σ_i. It is straightforward to see that $\sum_{j=1}^{k-1} \left\lceil \frac{\theta + X_j + (1-x_j)(D_j - C_j)}{T_j} \right\rceil C_j$ reaches the worst case if X_j is the largest. Since X_j is upper bounded by $Q_j^{\vec{x}}$ defined in Theorem 1, we reach the conclusion.

<u>An example of the procedures in Step 3:</u> This can be demonstrated in Figure 3 based on the previous example in Figure 1. Figure 3 provides the imaginary workload and an imaginary execution plan based on the test behind the condition in Eq. (20). *Note that this is not an actual schedule since task τ_2 is artificially alerted to release two jobs within a short time interval. This is only for illustrative purposes.* For such a case, $t_1^* = 4, t_2^* = 5, t_3^* = 5$, and $t_4^* = 6$. The two idle time units are used between time 4 and time 6. The accumulated workload is then started to be executed at time 6 and the processor does not idle after time 6. Over here, we see that two jobs of task τ_2 are executed back to back from time 7 to time 9. As shown in the imaginary schedule in Figure 3, the processor is busy executing the workload from time 6 to time 21, which is more pessimistic than the actual in Figure 1. The conclusion we have in the final statement of the theorem is that $20 - 7 = f_k - r_k \leq 21 - 6$. $\square$

Lemma 2. *$\forall \Delta \geq 0$ and $\forall C_j \geq c_j^* \geq 0$,*

$$\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*),$$

where $\widehat{W}_j^0(\Delta, 0)$ is defined in Eq. (10) and $\widehat{W}_j^0(\Delta, c_j^*)$ is defined in Eq. (11) if $c_j^* > 0$.

Proof: The proof is based on simple observations of the workload function. We first prove that $\widehat{W}_j^0(\Delta, C_j) \geq W_j^1(\Delta)$ defined in Eq. (10). By the definition of $\rho_j = T_j - D_j + C_j$ when c_j^* is C_j and the assumption $C_j \leq D_j \leq T_j$, we have $0 \leq \rho_j \leq T_j$. Therefore, for $\Delta \geq T_j$, we have $W_j^1(\Delta) = C_j + W_j^1(\Delta - T_j) \leq C_j + W_j^1(\Delta - \rho_j) \leq \widehat{W}_j^0(\Delta, C_j)$. For $0 \leq \Delta < T_j$, it is also obvious that $\widehat{W}_j^0(\Delta, C_j) \geq \min\{\Delta, C_j\} = W_j^1(\Delta)$.

We then prove that $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$ for any $0 < c_j^* \leq C_j$ based on the definition in Eq. (11). Figure 4 provides an illustrative example for $\widehat{W}_j^0(\Delta, c_j^*)$. We consider three subcases:

- For $0 \leq \Delta \leq C_j$, it is obvious that $\widehat{W}_j^0(\Delta, C_j) \geq \widehat{W}_j^0(\Delta, c_j^*)$.
- For $C_j < \Delta \leq T_j - D_j + C_j$, we have $\widehat{W}_j^0(\Delta, C_j) = C_j$, and it is obvious that $\widehat{W}_j^0(\Delta, c_j^*) = c_j^* + \max\{0, \Delta - (T_j - D_j + c_j^*)\} \leq c_j^* + C_j - c_j^* = C_j$.
- For $T_j - D_j + C_j < \Delta$, we have $\widehat{W}_j^0(\Delta, C_j) = C_j + W_j^1(\Delta - (T_j - D_j + C_j))$. Moreover, by definition, we also know $\widehat{W}_j^0(\Delta, c_j^*) \leq \delta + \widehat{W}_j^0(\Delta - \delta, c_j^*)$ for any δ with $0 < \delta \leq \Delta$. Therefore, for such a case, we can conclude $\widehat{W}_j^0(\Delta, c_j^*) = c_j^* + W_j^1(\Delta - (T_j - D_j + c_j^*)) \leq C_j + W_j^1(\Delta - (T_j - D_j + C_j))$ by setting δ to $C_j - c_j^*$ with the previous inequality.

$\blacksquare$

6 Dominance over the State of the Art

In this section, we prove that the schedulability test presented in Corollary 1 dominates all the existing tests in the state-of-the-art, in the sense that if a task set is deemed

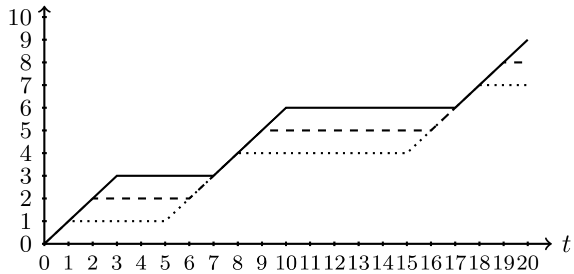

Fig. 4: The workload function $\widehat{W}_j^0(\Delta, c_j^*)$ when $T_j = 10$, $C_j = 3$, and $D_j = 6$. Solid line: c_j^* is 3, Dashed line: c_j^* is 2, Dotted line: c_j^* is 1.

schedulable by either of the tests presented in Section 3, then it is also deemed schedulable by Corollary 1.

Lemma 3. *The schedulability test of task τ_k provided by Eq.* (4) *dominates that of Eq.* (1).

Proof: It is straightforward to see that

$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \\
\geq &C_k + S_k + \sum_{i=1}^{k-1} S_i + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i \\
\geq &C_k + S_k + \sum_{i=1}^{k-1} \min(C_i, S_i) + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i
\end{aligned}$$

and by using the definition of B_k (i.e., Equation (3)), we get

$$C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil (C_i + S_i) \geq C_k + B_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i$$

Therefore, Eq. (4) will always have a solution which is smaller than or equal to the solution of Eq. (1). This proves the lemma. ∎

Lemma 4. *The schedulability test presented in Corollary 1 dominates the schedulability test provided by Eq.* (2).

Proof: Consider the case where $x_1 = x_2 = \cdots = x_{k-1} = 0$. Eq. (6) becomes identical to Eq. (2) for this particular vector assignment. Therefore, if Eq. (2) deems a task set as being schedulable, so does Corollary 1. This proves the lemma. ∎

Lemma 5. *The schedulability test presented in Corollary 1 dominates the schedulability test provided by Eq.* (4).

Proof: In this proof, we first transform the worst-case response time analysis presented in Corollary 1 in a more pessimistic analysis. We then prove that this more pessimistic version of Corollary 1 provides the same solution than Eq. (4), which then proves the lemma.

Since $Q_i^{\vec{x}} \stackrel{\text{def}}{=} \sum_{j=i}^{k-1} S_j \times x_j$, it holds that $Q_i^{\vec{x}} \leq Q_1^{\vec{x}}$ for $i = 1, 2, \ldots, k-1$. It follows that

$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil \\
\stackrel{(Q_i^{\vec{x}} \leq Q_1^{\vec{x}})}{\leq} &C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}} + (1 - x_i)(R_i - C_i)}{T_i} \right\rceil \\
\stackrel{(R_i \leq D_i \leq T_i)}{\leq} &C_k + S_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}} + (1 - x_i)T_i}{T_i} \right\rceil \\
\stackrel{(x_i \in \{0,1\})}{=} &C_k + S_k + \sum_{i=1}^{k-1} (1 - x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}}}{T_i} \right\rceil
\end{aligned}$$

Therefore, the smallest positive value t such that

$$C_k + S_k + \sum_{i=1}^{k-1} (1 - x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}}}{T_i} \right\rceil \leq t \tag{22}$$

is always larger than or equal to the solution to Eq. (6).

Replacing $(t + Q_1^{\vec{x}})$ by θ in Eq. (22), we get that R_k is upper bounded by the minimum value $(\theta - Q_1^{\vec{x}})$ greater than 0 (and therefore by the smallest $\theta > 0$) such that

$$\begin{aligned}
&C_k + S_k + \sum_{i=1}^{k-1} (1 - x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil \leq \theta - Q_1^{\vec{x}} \\
\Leftrightarrow\ &C_k + S_k + Q_1^{\vec{x}} + \sum_{i=1}^{k-1} (1 - x_i) C_i + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil \leq \theta \\
\Leftrightarrow\ &C_k + S_k + \sum_{i=1}^{k-1} (x_i S_i + (1 - x_i) C_i) + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta.
\end{aligned} \tag{23}$$

Now, consider the particular vector assignment $\vec{x}$ in which

$$x_i = \begin{cases} 1 & \text{if } S_i \leq C_i \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, 2, \ldots, k-1$. By the definition of B_k (i.e., Eq. (3)), we get that

$$B_k = S_k + \sum_{i=1}^{k-1} \min(C_i, S_i) = S_k + \sum_{i=1}^{k-1} (x_i S_i + (1 - x_i) C_i)$$

Eq. (23) thus becomes identical to Eq. (4). Therefore, if Eq. (4) deems a task set as being schedulable, so does Corollary 1. ∎

Theorem 2. *The schedulability test presented in Corollary 1 dominates the schedulability tests provided by Equations* (1), (2), *and* (4).

Proof: It ias a direct application of Lemmas 3, 4 and 5. ∎

As a corollary of this theorem, it directly follows that all the response time analyses discussed in Section 3 are in fact correct. This provides the first proof of correctness for Eq. (4), which was initially presented in [15] but never proven correct.

Theorem 3. *The schedulability tests provided by Eqs* (1), (2), *and* (4) *are all correct.*

Proof: It directly results from the two following facts,

(i) by Theorem 2, the schedulability test presented in Corollary 1 dominates the schedulability tests provided by Equations (1), (2), and (4);

(ii) as proven in Section 5.2, Corollary 1 is correct.

■

7 Linear Approximation

To test the schedulability of a task τ_k, Corollary 1 implies to test all the possible vector assignments $\vec{x} = (x_1, x_2, \ldots, x_{k-1})$. 2^{k-1} possible combinations must therefore be tested, implying an exponential time complexity. In this section, we therefore provide a solution to reduce the time complexity associated to Corollary 1. Indeed, using a linear approximation of the test in Eq. (6), a good vector assignment can be derived in linear time.

By definition of the ceiling operator, it holds that:

$$\begin{aligned}
&C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1-x_i)(R_i - C_i)}{T_i} \right\rceil C_i \\
\leq & C'_k + \sum_{i=1}^{k-1} \left(\frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1-x_i)(R_i - C_i)}{T_i} + 1 \right) C_i \\
= & C'_k + \sum_{i=1}^{k-1} \left(U_i \cdot t + C_i + U_i(1-x_i)(R_i - C_i) + U_i \sum_{\ell=i}^{k-1} x_\ell S_\ell \right) \\
= & C'_k + \sum_{i=1}^{k-1} \left(U_i \cdot t + C_i + U_i(1-x_i)(R_i - C_i) + x_i S_i \left(\sum_{\ell=1}^{i} U_\ell \right) \right)
\end{aligned} \tag{24}$$

By observing Eq. (24), the contribution of x_i can be individually determined as $U_i(R_i - C_i)$ when x_i is 0 or $S_i(\sum_{\ell=1}^{i} U_\ell)$ when x_i is 1. Therefore, whether x_i should be set to 0 or 1 can be decided by individually comparing the two constants $U_i(R_i - C_i)$ and $S_i(\sum_{\ell=1}^{i} U_\ell)$. We denote the vector assignment obtained with this technique by $\vec{x}^{lin}$, where, for each higher-priority task τ_i,

$$x_i^{lin} = \begin{cases} 1 & \text{if } U_i(R_i - C_i) > S_i(\sum_{\ell=1}^{i} U_\ell) \\ 0 & \text{otherwise} \end{cases}$$

For notational brevity, we denote the right-hand side of Eq. (24) as $rbf_k(t, \vec{x})$ for any $t > 0$ and given $\vec{x}$.

Theorem 4. *For any $t > 0$, the vector assignment $\vec{x}^{linear}$ minimizes $rbf_k(t, \vec{x})$ among all 2^{k-1} possible vector assignments for the $k-1$ higher-priority tasks. Task τ_k is schedulable under the fixed-priority scheduling if*

$$rbf_k(D_k, \vec{x}^{linear}) \leq D_k. \tag{25}$$

Deriving $\vec{x}^{linear}$ requires $O(k)$ time complexity and testing Eq. (24) also requires only $O(k)$ time complexity.

Proof: The correctness to test Eq. (25) is due to the derivation in Eq. (24) and Corollary 1. The other statements in this theorem are based on the above discussion in this section and simple observations. ■

8 Utilization Bounds and Speedup Factors

Suppose that $S_i \leq \gamma C_i$ for every task $\tau_i \in hp(\tau_k)$. We will present the utilization bounds in this subsection.

We start from the analysis by Liu, which considers the self-suspension time as blocking time for such cases. By using the k2U framework, task τ_k in an implicit deadline system is schedulable by using RM scheduling if

$$(\frac{C_k + S_k}{T_k} + 1 + \gamma) \prod_{i=1}^{k-1} (1 + U_i) \leq 2 + \gamma.$$

That is, $0 < \alpha_i \leq 1+\gamma$ and $0 < \beta_i \leq 1$ for $i = 1, 2, \ldots, k-1$. This gives the immediate utilization bound to find the infimum $\sum_{i=1}^{k} U_k$ such that

$$\begin{aligned}
&(1+\gamma) * (1 + U_k) \prod_{i=1}^{k-1} (1 + U_i) \\
&\geq (\frac{C_k + S_k}{T_k} + 1 + \gamma) \prod_{i=1}^{k-1} (1 + U_i) > 2 + \gamma. \\
&\Rightarrow \prod_{i=1}^{k} (1 + U_i) > \frac{2+\gamma}{1+\gamma}.
\end{aligned}$$

Therefore, the utilization bound for a given $0 \leq \gamma \leq 1$ is $\ln(\frac{2+\gamma}{1+\gamma})$.

9 Conclusion

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

References

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

[2] N. C. Audsley and K. Bletsas. Realistic analysis of limited parallel software / hardware implementations. In *10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 388–395, 2004.

[3] 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, CISTER, July 2015.

[4] B. Brandenburg. Improved analysis and evaluation of real-time semaphore protocols for P-FP scheduling. In *RTAS*, 2013.

[5] A. Carminati, R. de Oliveira, and L. Friedrich. Exploring the design space of multiprocessor synchronization protocols for real-time systems. *Journal of Systems Architecture*, 60(3):258–270, 2014.

[6] J.-J. Chen, W.-H. Huang, and G. Nelissen. A note on modeling self-suspending time as blocking time in real-time systems. Technical report, 2015.

[7] G. Han, H. Zeng, M. Natale, X. Liu, and W. Dou. Experimental evaluation and selection of data consistency mechanisms for hard real-time applications on multicore platforms. *IEEE Transactions on Industrial Informatics*, 10(2):903–918, 2014.

[8] W.-H. Huang, J.-J. Chen, H. Zhou, and C. Liu. PASS: Priority assignment of real-time tasks with dynamic suspending behavior under fixed-priority scheduling. In *Design Automation Conference (DAC)*, 2015.

[9] 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.

[10] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A Responsive GPGPU Execution Model for Runtime Engines. In *2011 IEEE 32nd Real-Time Systems Symposium*, 2011.

[11] H. Kim, S. Wang, and R. Rajkumar. vMPCP: a synchronization framework for multi-core virtual machines. In *RTSS*, 2014.

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

[13] K. Lakshmanan, D. De Niz, and R. Rajkumar. Coordinated task scheduling, allocation and synchronization on multiprocessors. In *RTSS*, 2009.

[14] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, jan 1973.

[15] J. W. S. W. Liu. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 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 *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference (DAC)*, 2014.

[17] L. Ming. Scheduling of the inter-dependent messages in real-time communication. In *Proc. of the First International Workshop on Real-Time Computing Systems and Applications*, 1994.

[18] 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.

[19] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In *Proceedings of the 9th IEEE Real-Time Systems Symposium (RTSS '88)*, pages 259–269, 1988.

[20] M. Yang, H. Lei, Y. Liao, and F. Rabee. PK-OMLP: An OMLP based k-exclusion real-time locking protocol for multi- GPU sharing under partitioned scheduling. In *DASC*, 2013.

[21] M. Yang, H. Lei, Y. Liao, and F. Rabee. Improved blocking timg analysis and evaluation for the multiprocessor priority ceiling protocol. *Jounal of Computer Science and Technology*, 29(6):1003–1013, 2014.

[22] H. Zeng and M. Natale. Mechanisms for guaranteeing data consistency and flow preservation in AUTOSAR software on multi-core platforms. In *SIES*, 2011.