

# Interference from Higher-Priority Self-Suspending Tasks Can Be Arbitrarily Modelled as Jitter or Carry-In Terms

Jian-Jia Chen, Wen-Hung Huang, and Geoffrey Nelissen

TU Dortmund University, Germany

Email: jian-jia.chen@tu-dortmund.de, 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  $\tau_i$  is characterized by 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  $\tau_i$  arrives at time  $t$ , the job should finish no later than its *absolute deadline*  $t + D_i$ , and the next job of task  $\tau_i$  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  $\tau_i$ .

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 all the computation, the above assumption becomes invalid. Self-suspension behaviour has become more visible in designing real-time embedded systems. The suspension-oblivious approach, which considers the suspension time as computation, can be very pessimistic if the suspension time is long. Self-suspensions have been even more pervasive in many emerging embedded cyber-physical systems in which the computational components frequently interact with external and physical devices [9], [10]. 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, each sporadic task  $\tau_i$  is further specified with its worst-case execution time  $C_i$  and its worst-case self-suspension time  $S_i$ . In the *segmented* self-suspension task model, the execution behaviour of a job of task  $\tau_i$  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 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. 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].<sup>1</sup>

<sup>1</sup>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].

- We develop a few strategies to decide which higher-priority tasks should be classified to associate with jitter terms and which higher-priority tasks should be classified to associate with carry-in terms. The methods are presented in Section 5.
- utilization bounds..
- evaluation results...

## 2 Task Model

The system model and terminologies are defined as follows: We assume a system composed of  $n$  sporadic self-suspending tasks. A sporadic task  $\tau_i$  is released repeatedly, with each such invocation called a job. The  $j^{th}$  job of  $\tau_i$ , denoted  $\tau_{i,j}$ , is released at time  $r_{i,j}$  and has an absolute deadline at time  $d_{i,j}$ . Each job of any task  $\tau_i$  is assumed to have a worst-case execution time  $C_i$ . Each job of task  $\tau_i$  suspends for at most  $S_i$  time units (across all of its suspension phases). When a job suspends itself, the processor can execute another job. 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  $\tau_i$  are a period (or minimum inter-arrival time)  $T_i$ , which specifies the minimum time between two consecutive job releases of  $\tau_i$ , and a relative deadline  $D_i$ , which specifies the maximum amount of time a job can take to complete its execution after its release, i.e.,  $d_{i,j} = r_{i,j} + D_i$ . The utilization of a task  $\tau_i$  is defined as  $U_i = C_i/T_i$ .

In this paper, we focus on constrained-deadline task systems, in which  $D_i \leq T_i$  for every task  $\tau_i$ . We only consider *preemptive fixed-priority scheduling on a single processor*, in which each task is assigned with a unique priority level. We assume that the priority assignment is given. The worst-case response time  $R_i$  of a task  $\tau_i$  is the maximum response time among all its jobs. A schedulability test (analysis) of a task  $\tau_k$  is to verify whether the worst-case response time of the task is no more than the associated relative deadline  $D_k$ .

We assume that the tasks are numbered in a decreasing priority order. That is, a task with a smaller index has higher priority than any task with a higher index, i.e., task  $\tau_i$  has a higher-priority level than task  $\tau_{i+1}$ . *When performing the schedulability analysis of a specific task  $\tau_k$ , we will implicitly assume that  $\tau_1, \tau_2, \dots, \tau_{k-1}$  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 Existing Analyses

To analyze the worst-case response time (or the schedulability) of task  $\tau_k$ , we usually need to quantify the worst-case interference caused by the higher-priority tasks on the execution of any job of task  $\tau_k$ . In the ordinary sequential sporadic real-time task model, i.e., when  $S_i = 0$  for every task  $\tau_i$ , the so-called critical instant theorem by Liu and Layland [14] is commonly adopted. That is, the worst-case response time of task  $\tau_k$  (if it is less than or equal to its period) happens for the first job of task  $\tau_k$  when  $\tau_k$  and all the higher-priority tasks release a job synchronously and the subsequent jobs are released as early as possible (i.e., with a rate equal to their period). However, as proven in [18], this definition of the critical instant does not hold for self-suspending sporadic tasks.

### 3.1 Suspension-Oblivious Analysis

The simplest approach is to convert the suspension time to computation time. Therefore, a constrained-deadline task  $\tau_k$  can be feasibly scheduled by the fixed-priority scheduling 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 Model the Back-to-Back Hit as Jitter

Another approach is to model the impact due to self-suspension as jitter. 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. A rather safe way is to quantify the jitter of a higher-priority task  $\tau_i$  as  $D_i - C_i$ . Therefore, a constrained-deadline task  $\tau_k$  can be feasibly scheduled by the fixed-priority scheduling if

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

### 3.3 Model Suspension Time as Blocking Time

In [15, Pages 164-165], Jane W. S. Liu proposed a solution to study the schedulability of self-suspending tasks by modeling the *extra delay* suffered by a task  $\tau_k$  due to the self-suspending behavior of the tasks as a blocking time denoted as  $B_k$  and defined as follows:

- The blocking time contributed from task  $\tau_k$  is  $S_k$ .
- A higher-priority task  $\tau_i$  can only block the execution of task  $\tau_k$  by at most  $b_i = \min(C_i, S_i)$  time units.

Therefore,

$$B_k = S_k + \sum_{i=1}^{k-1} b_i. \quad (3)$$

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  $\frac{C_k + B_k}{T_k} + \sum_{i=1}^{k-1} U_i \leq k(2^{\frac{1}{k}} - 1)$ , then task  $\tau_k$  can be feasibly scheduled by using rate-monotonic scheduling if  $T_i = D_i$  for every task  $\tau_i$  in the given task set.

The same concept was also implicitly used by Rajkumar, Sha, and Lehoczky [19, Page 267] for analyzing self-suspending behaviour due to synchronization protocols in multiprocessor systems. To account for the self-suspending behaviour, it reads as follows:<sup>2</sup>

*For each higher priority job  $J_i$  on the processor 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  $J_i$  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.*

<sup>2</sup>We rephrased the wordings and notation to be consistent with this paper.

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

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

However, there is no proof in [15], [19] to support the correctness of the above tests. It is not difficult to see that the test in Eq. (4) dominates that in Eq. (1).

**Lemma 1.** *The schedulability test of task  $\tau_k$  in Eq. (4) dominates that in Eq. (1).*

*Proof:* This is rather trivial, and the proof is omitted. ■

## 4 Our General Analysis Framework

We can greedily convert the suspension time of task  $\tau_k$  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  $\tau_k$  as task  $\tau'_k$ . Suppose that  $R'_k$  is the worst-case response time in the task system  $\{\tau_1, \tau_2, \dots, \tau_{k-1}, \tau'_k\}$ . 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  $\tau_k$  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  $\vec{x} = (x_1, x_2, \dots, x_{k-1})$ , 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*

$$C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq t, \quad (5)$$

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

We will explain the resulting properties from Theorem 1 first, by leaving the proof to Section 4.2 since it is pretty long. With Theorem 1, we can directly have the following corollary.

**Corollary 1.** *If there exists a vector assignment  $\vec{x} = (x_1, x_2, \dots, x_{k-1})$ , in which  $x_i$  is either 0 or 1, such that*

$$\exists t \mid 0 < t \leq D_k, C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_i^{\vec{x}} + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq t, \quad (6)$$

where  $Q_i^{\vec{x}} = \sum_{j=i}^{k-1} S_j \cdot x_j$ , then a constrained-deadline task  $\tau_k$  can be feasibly scheduled by the fixed-priority scheduling.

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

### 4.1 An Illustrative Example and Dominance

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  $\tau_1$  and  $\tau_2$  can be verified to be schedulable under the fixed-priority scheduling by using Eq. (2).

We focus on task  $\tau_3$ . For task  $\tau_3$ , 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.,  $4 + 5 + \left\lceil \frac{37}{10} \right\rceil \cdot 4 + \left\lceil \frac{37}{19} \right\rceil \cdot 6 = 37$ . Therefore, task  $\tau_3$  cannot pass the schedulability test in Eq. (4). There are four possible vector assignments  $\vec{x}$  when we consider the schedulability of task  $\tau_3$ :

- 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 satisfies

$$4 + \left\lceil \frac{t+6}{10} \right\rceil \cdot 4 + \left\lceil \frac{t+13}{19} \right\rceil \cdot 6 \leq t. \quad (7)$$

Such a value  $t$  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

$$4 + \left\lceil \frac{t+7}{10} \right\rceil \cdot 4 + \left\lceil \frac{t+1}{19} \right\rceil \cdot 6 \leq t. \quad (8)$$

Therefore,  $R'_k \leq 32$  due to  $4 + \left\lceil \frac{32+7}{10} \right\rceil \cdot 4 + \left\lceil \frac{32+1}{19} \right\rceil \cdot 6 = 32$ .

- Case 3  $\vec{x} = (1, 0)$ : In this case, Theorem 1 states that  $R'_k$  is upper bounded by the minimum  $t$  under  $0 < t \leq T_3$  that satisfies

$$4 + \left\lceil \frac{t+5}{10} \right\rceil \cdot 4 + \left\lceil \frac{t+13}{19} \right\rceil \cdot 6 \leq t. \quad (9)$$

Such a value  $t$  does not exist for this case.

- 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

$$4 + \left\lceil \frac{t+6}{10} \right\rceil \cdot 4 + \left\lceil \frac{t+1}{19} \right\rceil \cdot 6 \leq t. \quad (10)$$

Therefore,  $R'_k \leq 32$  due to  $4 + \left\lceil \frac{32+6}{10} \right\rceil \cdot 4 + \left\lceil \frac{32+1}{19} \right\rceil \cdot 6 = 32$ .

Among the above four cases, the tests in Cases 2 and 4 are tighter. By Corollary 1, task  $\tau_3$  is schedulable by the fixed-priority scheduling policy.

In fact, the following theorem shows that the test in Corollary 1 analytically dominates the existing tests in Eq. (1), Eq. (2) and Eq. (4).

**Theorem 2.** *The schedulability test in Corollary 1 dominates the schedulability tests in Eq. (1), Eq. (2), and Eq. (4).*

*Proof:* The dominance of Eq. (2) can be easily seen by considering the vector assignment  $x_1 = x_2 = \dots = x_{k-1} = 0$ . The resulting test in Eq. (6) is identical to Eq. (4) for this vector assignment. Since Eq. (4) dominates Eq. (1), we only have to prove that Corollary 1 dominates Eq. (4).

We now prove that Eq. (4) is dominated by considering the vector assignment  $\vec{x}$  in which

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

for  $i = 1, 2, \dots, k-1$ . By the fact that  $Q_i^{\vec{x}} \leq Q_1^{\vec{x}}$  for  $i = 1, 2, \dots, k-1$ , we know that it is more pessimistic if we test  $C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + Q_1^{\vec{x}} + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq t$  instead of testing

Eq. (6). Let  $\theta$  be  $t + Q_1^{\vec{x}}$ . Therefore, we know that  $R'_k$  is upper bounded by the minimum  $\theta - Q_1^{\vec{x}} > 0$  such that

$$C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{\theta + (1-x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq \theta - Q_1^{\vec{x}} \quad (11)$$

$$\Rightarrow C'_k + Q_1^{\vec{x}} + \sum_{i=1}^{k-1} \left\lceil \frac{\theta + (1-x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq \theta. \quad (12)$$

Moreover, by the fact that  $D_i \leq T_i$  and  $x_i \in \{0, 1\}$  for  $i = 1, 2, \dots, k-1$ , we also have  $\left\lceil \frac{\theta + (1-x_i)(D_i - C_i)}{T_i} \right\rceil C_i \leq \left\lceil \frac{\theta + (1-x_i)T_i}{T_i} \right\rceil C_i = (1-x_i)C_i + \left\lceil \frac{\theta}{T_i} \right\rceil C_i$ . Therefore, we know that  $R'_k$  is upper bounded by the minimum  $\theta - Q_1^{\vec{x}} > 0$  such that

$$C_k + S_k + \sum_{i=1}^{k-1} (x_i S_i + (1-x_i)C_i) + \sum_{i=1}^{k-1} \left\lceil \frac{\theta}{T_i} \right\rceil C_i \leq \theta. \quad (13)$$

By the fact that  $B_k$  is defined as  $S_k + \sum_{i=1}^{k-1} (x_i S_i + (1-x_i)C_i)$ , and  $Q_1^{\vec{x}} \geq 0$ , the above test in Eq. (13) is analytically tighter than or the same as that in Eq. (4), which concludes the proof. ■

## 4.2 Proof of Theorem 1

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  $\tau_k$  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  $\tau_i$  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 2.** *In a preemptive fixed-priority schedule, if the worst-case response time of task  $\tau_i$  is no more than its period  $T_i$ , preventing the release of a job of task  $\tau_i$  does not affect the schedule of any other job of task  $\tau_i$ .*

*Proof:* Since the worst-case response time of task  $\tau_i$  is no more than its period, any job  $\tau_{i,j}$  of task  $\tau_i$  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  $\tau_i$ , 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  $\tau_i$  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  $\tau_1$  arrives at time  $4 + \epsilon$  with a very small  $\epsilon > 0$ . The first job of task  $\tau_2$  suspends itself from time 0 to time  $5 + \epsilon$ , and is blocked by task  $\tau_1$  from time  $5 + \epsilon$  to time  $6 + \epsilon$ . After some very short computation with  $\epsilon$  amount of time, the first job of task  $\tau_2$  suspends itself again from time  $6 + 2\epsilon$  to 7.

**Proof of Theorem 1.** Let us consider the task set  $\tau'$  composed of  $\{\tau_1, \tau_2, \dots, \tau_{k-1}, \tau'_k, \tau_{k+1}, \dots\}$  and let  $\Psi$  be a schedule of  $\tau'$ , in which  $R'_k \leq T_k$  by our assumption. Suppose that a job  $J_k$  of task  $\tau'_k$  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  $\Psi$ .

The proof is built upon the two following steps:

- 1) We discard all the jobs that do not contribute to the response time of  $J_k$  in the schedule  $\Psi$ . We follow an inductive strategy by iteratively inspecting the schedule of the higher priority tasks in  $\Psi$ , starting with  $\tau_{k-1}$  until the highest priority task  $\tau_1$ . 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  $\tau_j$  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  $\tau_j$  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  $\tau'_k$  in  $\Psi$ . 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  $\tau'_k$ .

### Step 1: Reducing the schedule $\Psi$

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

*Basic step (definition of  $\Psi^k$  and  $t_k$ ):*

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



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

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

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

Let  $r_j$  be the arrival time of the last job released by  $\tau_j$  before  $t_{j+1}$  in  $\Psi^{j+1}$  and let  $J_j$  denote that job. Removing all the jobs of task  $\tau_j$  arrived before  $r_j$  has no impact on the schedule of any other job released by  $\tau_j$  (Lemma 2) or any higher priority job released by  $\tau_1, \dots, \tau_{j-1}$  (Property 1). Moreover, because by the construction of  $\Psi^{j+1}$ , no task with a priority lower than  $\tau_j$  executes jobs before  $t_{j+1}$  in  $\Psi^{j+1}$ , removing the jobs released by  $\tau_j$  before  $t_{j+1}$  does not impact the schedule of the jobs of  $\tau_{j+1}, \dots, \tau_k$ . Therefore, we can safely remove all the jobs of task  $\tau_j$  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 2 and Property 1, removing all the jobs of task  $\tau_j$  arrived before  $t_{j+1}$  has no impact on the schedule of the higher-priority jobs (jobs released by  $\tau_1, \dots, \tau_{j-1}$ ) and the jobs of  $\tau_j$  released after or at  $t_{j+1}$ . Moreover, because no task with lower priority than  $\tau_j$  executes jobs before  $t_{j+1}$  in  $\Psi^{j+1}$ , removing the jobs released by  $\tau_j$  before  $t_{j+1}$  does not impact the schedule of the jobs of  $\tau_{j+1}, \dots, \tau_k$ . Therefore,  $t_j$  is set to  $t_{j+1}$  and  $\Psi^j$  is generated by removing all the jobs of task  $\tau_j$  arrived before  $t_{j+1}$ . The response time of  $J_k$  in  $\Psi^j$  thus remains unchanged in comparison to its response time in  $\Psi^{j+1}$ .
  - $J_j$  did not complete its execution by  $t_{j+1}$ . For such a case,  $t_j$  is set to  $r_j$  and hence  $\Psi^j$  is built from  $\Psi^{j+1}$  by removing all the jobs released by  $\tau_j$  before  $r_j$ .

Note that because by the construction of  $\Psi^{j+1}$  and hence  $\Psi^j$  there is no job with priority lower than  $\tau_j$  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  $\tau_j$  at time  $t_j$ . We know that  $c_j^*$  is at most  $C_j$ . Since by the construction of  $\Psi^j$ , all the jobs of  $\tau_j$  released before  $t_j$  are removed, the job of task  $\tau_j$  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  $\tau_j$ . 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  $\tau_j$  at time  $t_j$  is called the *residual workload* of task  $\tau_j$  for the rest of the proof.

The above construction of  $\Psi^{k-1}, \Psi^{k-2}, \dots, \Psi^1$  is repeated until producing  $\Psi^1$ . The procedures are well-defined. Therefore, it is guaranteed that  $\Psi^1$  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  $\tau_2$ , 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  $\tau_2$  arrived before time  $t_1$  with execution time  $C_2$ .

## Step 2: Analyzing the final reduced schedule $\Psi^1$

We now analyze the properties of the final schedule  $\Psi^1$  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

$$\text{idle}(t_1, t) + \text{exec}(t_1, t) = (t - t_1) \quad (14)$$

where  $\text{exec}(t_1, t)$  is the amount of time during which the processor executed tasks within  $[t_1, t)$ , and  $\text{idle}(t_1, t)$  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  $\Psi^1$ . Suppose that  $\sigma_i$  is the sum of the idle time in this interval  $[t_i, t_{i+1})$  in  $\Psi^1$ . If  $t_i$  is equal to  $t_{i+1}$ , then  $\sigma_i$  is set to 0. Therefore, we have

$$\text{idle}(t_1, t) \leq \sum_{i: t_i < t} \sigma_i. \quad (15)$$

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  $\tau_k'$  in  $\Psi^1$ , we only focus on the execution patterns of the tasks  $(\tau_1, \tau_2, \dots, \tau_{k-1}, \tau_k')$ . According to Step 1, we should consider two cases:

- If task  $\tau_j$  is in  $\mathbf{T}_1$ , there is no job of task  $\tau_j$  arrived before  $t_j$  in  $\Psi^1$ . 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  $\tau_j$  from  $t_j$  to  $t_j + \Delta$  that is executed on the processor is at most

$$W_j^1(\Delta) = \left\lfloor \frac{\Delta}{T_j} \right\rfloor C_j + \min \left\{ \Delta - \left\lfloor \frac{\Delta}{T_j} \right\rfloor T_j, C_j \right\}. \quad (16)$$

- If task  $\tau_j$  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  $\tau_j$  from  $t_j$  to  $t_j + \Delta$  has to consider two subcases:
  - If the residual workload  $c_j^*$  of task  $\tau_j$  is 0, the earliest arrival time of task  $\tau_j$  can be any time point at or after  $t_j$ . In this case, for any  $\Delta \geq 0$ , the workload contributed from task  $\tau_j$  from  $t_j$  to  $t_j + \Delta$  that is executed on the processor is at most

$$\widehat{W}_j^0(\Delta, 0) = W_j^1(\Delta). \quad (17)$$

- If the residual workload  $c_j^*$  of task  $\tau_j$  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  $\tau_j$  arriving *strictly after*  $t_j$  is at least  $t_j + (T_j - D_j + c_j^*)$  in  $\Psi^1$ . For notational brevity, let  $\rho_j$  be  $(T_j - D_j + c_j^*)$ . In this case, for any  $\Delta \geq 0$  and  $c_j^* > 0$ , the workload contributed from task  $\tau_j$  from  $t_j$  to  $t_j + \Delta$  that is executed on the processor is at most

$$\widehat{W}_j^0(\Delta, c_j^*) = \begin{cases} \Delta & \text{if } \Delta \leq c_j^* \\ c_j^* & \text{if } c_j^* < \Delta \leq \rho_j \\ c_j^* + W_j^1(\Delta - \rho_j) & \text{otherwise.} \end{cases} \quad (18)$$

It is proved in Lemma 3 that the worst case residual workload in  $\widehat{W}_j^0(\Delta, c_j^*)$  by considering both Eq. (17) and Eq. (18) 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

$$W_j^0(\Delta) \stackrel{\text{def}}{=} \widehat{W}_j^0(\Delta, C_j) \quad (19)$$

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

$$\text{exec}(t_1, t) \leq \sum_{j=1}^{i-1} x_j \cdot W_j^1(t - t_j) + (1 - x_j) \cdot W_j^0(t - t_j). \quad (20)$$

Putting Eqs. (14), (15), (20) together, we have for  $i = 2, 3, \dots, k-1$ ,  $\forall t \mid t_{i-1} \leq t < t_i$

$$\sum_{j=1}^{i-1} x_j \cdot (W_j^1(t - t_j) + \sigma_j) + (1 - x_j) \cdot W_j^0(t - t_j) \geq t - t_1. \quad (21)$$

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

$$C'_k + \sum_{j=1}^{k-1} x_j \cdot (W_j^1(t - t_j) + \sigma_j) + (1 - x_j) \cdot W_j^0(t - t_j) > t - t_1. \quad (22)$$

An example of the procedures in Step 2: In the example used

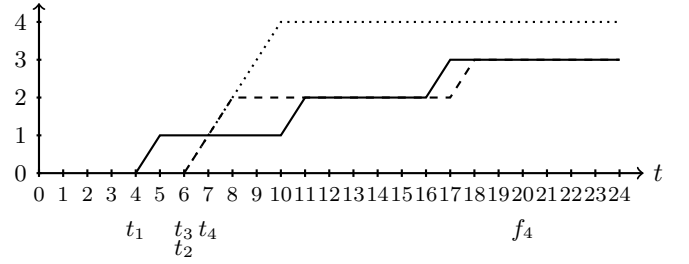


Fig. 2: The workload function for the three higher-priority tasks in Figure 2. Solid line:  $W_1^1(t - t_1)$ , Dashed line:  $W_2^0(t - t_2)$ , Dotted line:  $W_3^1(t - t_3)$ , where the functions are 0 if  $t - t_j < 0$  for  $j = 1, 2, 3$ .

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. (21) and Eq. (22) 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. (21) and (22). We will construct another release pattern which moves  $t_i$  to  $t_i^*$  for  $i = 2, 3, \dots, k$  such that  $t_i^* \leq t_i$  and the corresponding conditions in Eqs. (21) and (22) will become worse when we use  $t_i^*$ . We start the procedure as follows:

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

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

$$W_j^1(\Delta) \leq W_j^1(\Delta + (t_j - t_j^*)). \quad (23)$$

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

$$W_j^0(\Delta) \leq W_j^0(\Delta + (t_j - t_j^*)). \quad (24)$$

Therefore, for any  $j = 1, 2, \dots, k-1$ , 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. (21)  $\forall t \mid t_k^* \leq t < t_k$  leads to

$$\begin{aligned} & \sum_{j=1}^{k-1} x_j \cdot (W_j^1(t - t_j^*) + \sigma_j) + (1 - x_j) \cdot W_j^0(t - t_j^*) \geq t - t_1, \\ \Rightarrow & \sum_{j=1}^{k-1} x_j \cdot W_j^1(t - t_j^*) + (1 - x_j) \cdot W_j^0(t - t_j^*) \geq t - t_k^*. \end{aligned} \quad (25)$$

Similarly, putting these into Eqs. (22)  $\forall t \mid t_k \leq t < f_k$  leads to

$$C'_k + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t - t_j^*) + (1 - x_j) \cdot W_j^0(t - t_j^*) > t - t_k^*. \quad (26)$$

By the assumption that  $C'_k \geq C_k > 0$ , we can unify the above inequalities in Eq. (25) and Eq. (26) as follows:  $\forall t \mid t_k^* \leq t <$

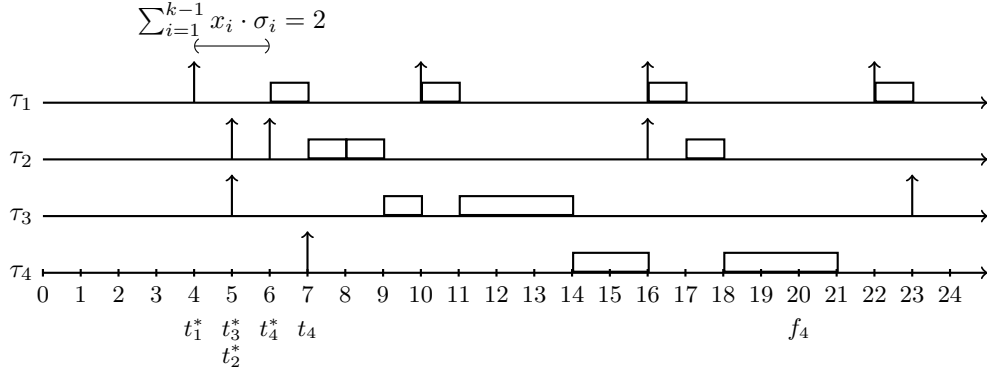


Fig. 3: An illustrative example of Step 3 in the proof of Theorem 1 based on an *imaginary* schedule.

$f_k$

$$C'_k + \sum_{j=1}^{k-1} x_j \cdot W_j^1(t - t_j^*) + (1 - x_j) \cdot W_j^0(t - t_j^*) > t - t_k^*. \quad (27)$$

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  $j = 1, 2, \dots, k-1$ . 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  $\tau_j$  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  $\tau_j$  in  $\mathbf{T}_0$ . Therefore, we can conclude that  $\forall t \mid t_k^* \leq t < f_k$

$$C'_k + \sum_{j=1}^{k-1} \left\lceil \frac{t - t_k^* + X_j + (1 - x_j)(D_j - C_j)}{T_j} \right\rceil C_j > t - t_k^*, \quad (28)$$

where  $X_j$  is  $\sum_{\ell=j}^{k-1} x_\ell \sigma_\ell$ . We replace  $t - t_k^*$  with  $\theta$ . The above inequality implies that the minimum  $\theta$  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  $\sigma_i$ . 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.

An example of the procedures in Step 3: 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. (27). *Note that this is not an actual schedule since task  $\tau_2$  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. Over here, we see that two jobs of task  $\tau_2$  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 3.**  $\forall \Delta \geq 0$  and  $\forall c_j^* \geq 0$ ,

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

where  $\widehat{W}_j^0(\Delta, 0)$  is defined in Eq. (17) and  $\widehat{W}_j^0(\Delta, c_j^*)$  is defined in Eq. (18) 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. (17). 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. (18). 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  $\delta$  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  $\delta$  to  $C_j - c_j^*$  with the previous inequality.  $\blacksquare$

## 5 Testing Different Vector Assignments

To test the schedulability of task  $\tau_k$ , Corollary 1 implies to test all the possible vector assignments  $\vec{x} = (x_1, x_2, \dots, x_{k-1})$ , in which there are  $2^{k-1}$  different combinations. Therefore, the time complexity becomes exponential if

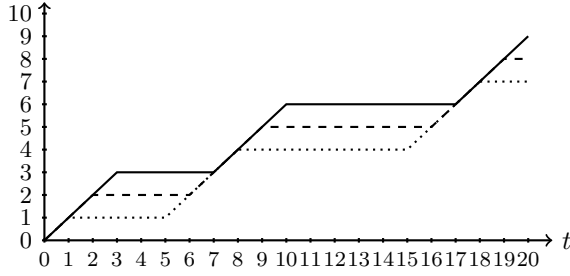


Fig. 4: The workload function  $\widehat{W}_j^0(\Delta, c_j^*)$  when  $T_j = 10$ ,  $C_j = 3$ , and  $D_j = 6$ . Solid line:  $c_j^*$  is 3, Dashed line:  $c_j^*$  is 2, Dotted line:  $c_j^*$  is 1.

we consider all the vector assignments. This section provides a few tricks to reduce the time complexity while adopting Corollary 1.

### 5.1 Linear Approximation

Here, we explain how to use the linear approximation of the test in Eq. (6) to help derive a good vector assignment. By the definition of  $\lceil x \rceil$ , we have the following inequality:

$$\begin{aligned}
 & C'_k + \sum_{i=1}^{k-1} \left\lceil \frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1 - x_i)(D_i - C_i)}{T_i} \right\rceil C_i \\
 & \leq C'_k + \sum_{i=1}^{k-1} \left( \frac{t + \sum_{\ell=i}^{k-1} x_\ell S_\ell + (1 - x_i)(D_i - C_i)}{T_i} + 1 \right) C_i \\
 & = C'_k + \sum_{i=1}^{k-1} \left( U_i \cdot t + C_i + U_i(1 - x_i)(D_i - C_i) + U_i \sum_{\ell=i}^{k-1} x_\ell S_\ell \right) \\
 & = C'_k + \sum_{i=1}^{k-1} \left( U_i \cdot t + C_i + U_i(1 - x_i)(D_i - C_i) + x_i S_i \left( \sum_{\ell=1}^i U_\ell \right) \right) \quad (29)
 \end{aligned}$$

By observing Eq. (29), the contribution of  $x_i$  can be individually determined as  $U_i(D_i - C_i)$  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 easily decided by individually comparing the two constants  $U_i(D_i - C_i)$  and  $S_i(\sum_{\ell=1}^i U_\ell)$ . We denote the vector assignment obtained above by  $\vec{x}^{linear}$ . That is, for each higher-priority task  $\tau_i$ ,

- if  $U_i(D_i - C_i) > S_i(\sum_{\ell=1}^i U_\ell)$ , we greedily set  $x_i^{linear}$  to 1;
- otherwise, we greedily set  $x_i^{linear}$  to 0.

For notational brevity, we denote the right-hand side of Eq. (29) as  $rbf_k(t, \vec{x})$  for any  $t > 0$  and given  $\vec{x}$ .

**Theorem 3.** *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  $\tau_k$  is schedulable under the fixed-priority scheduling if*

$$rbf_k(D_k, \vec{x}^{linear}) \leq D_k. \quad (30)$$

Deriving  $\vec{x}^{linear}$  requires  $O(k)$  time complexity and testing Eq. (29) also requires only  $O(k)$  time complexity.

*Proof:*

**Corollary 2.** *Considering task  $\tau_k$  from  $\tau_1, \tau_2, \dots, \tau_n$ , the time complexity to test the schedulability of all these  $n$  tasks is  $O(n)$  by using the test in Theorem 3. Therefore, the amortized time complexity to test task  $\tau_k$  by using the test in Theorem 3 is constant.*

*Proof:*

## 5.2 Iterative Steps

## 6 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  $\tau_k$  in an implicit deadline system is schedulable by using RM scheduling if

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

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

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

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

## 7 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 time analysis and evaluation for the multiprocessor priority ceiling protocol. *Journal 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.