Efficient Scheduling of Weakly-Hard Real-Time Tasks with Sufficient Schedulability Condition

V. Gabriel Moyano
German Aerospace Center (DLR)
Braunschweig, Germany
gabriel.moyano@dlr.de

Zain A. H. Hammadeh
German Aerospace Center (DLR)
Braunschweig, Germany
zain.hajhammadeh@dlr.de

Selma Saidi
Technische Universität Braunschweig
Braunschweig, Germany
saidi@ida.ing.tu-bs.de

Daniel Lüdtke
German Aerospace Center (DLR)
Braunschweig, Germany
daniel.luedtke@dlr.de

Abstract

Many real-time tasks, particularly control tasks, can accommodate occasional missed deadlines thanks to robust algorithms. These tasks can be effectively modeled using the weakly-hard model, which specifies the maximum number of tolerable deadline misses, denoted as m_i, within a sequence of K_i executions. Research indicates that utilizing the weakly-hard model can significantly reduce the over-provisioning typically required in the design of real-time systems. Therefore, different scheduling algorithms and schedulability analyses have been proposed in the last few years. However, state-of-the-art scheduling analyses do not scale with larger values of K_i. We present a new job-level fixed priority scheduling algorithm whose schedulability analysis scales with K_i. Furthermore, our scheduling algorithm leverages the tolerable continuous deadline misses to assigning priorities to jobs. Schedulability analyses show that the computation time of our analysis is up to 100 times faster comparing to the approaches in literature improving also the schedulability ratio for total utilization of 0.9.

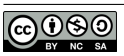
CCS Concepts

• **Computer systems organization → Real-time system specification**.

Keywords

Weakly-Hard, Real-Time Scheduling

ACM Reference Format:
V. Gabriel Moyano, Zain A. H. Hammadeh, Selma Saidi, and Daniel Lüdtke. 2025. Efficient Scheduling of Weakly-Hard Real-Time Tasks with Sufficient Schedulability Condition. In *The 40th ACM/SIGAPP Symposium on Applied Computing (SAC '25), March 31-April 4, 2025, Catania, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3672608.3707844



SAC '25, March 31-April 4, 2025, Catania, Italy

ACM ISBN 979-8-4007-0629-5/25/03
https://doi.org/10.1145/3672608.3707844

1 Introduction

Advanced and computationally demanding control algorithms take an important role in automotive and aerospace systems nowadays. These control algorithms require real-time guarantees which computation usually consider worst-case scenarios. The use of these considered scenarios comes at the cost of exacerbating over-provisioning in such new software-based embedded systems, which implies, for example, higher power consumption. Moreover, studies, including references [9, 11, 13], have demonstrated that control systems can withstand occasional deadline misses with only minimal performance degradation.

The weakly-hard real-time model [2] broadens the range of schedulable tasks defined by the hard real-time model by effectively leveraging tolerable deadline misses. In the weakly-hard real-time model, the notation $\overline{\binom{m_i}{K_i}}$ defines the maximum number of tolerable deadline misses m_i in a sequence of K_i executions. To compute weakly-hard real-time guarantees, the developed analysis should consider all possible combinations of jobs within a window of K consecutive jobs. That makes computing the weakly-hard real-time guarantees more complicated and subject for more pessimism. However, literature [7] shows that leveraging the weakly-hard model can relax the over-provisioning associated with designed real-time systems.

In recent years, weakly-hard real-time systems have garnered significant attention, leading to the development of various schedulability analyses [5, 12, 16]. Choi et al. proposed a job-level fixed priority scheduling approach for single-core systems in [4, 5], where jobs are assigned varying priorities based on whether they meet or miss their deadlines. The scheduling analysis proposed by Choi et al. requires a reachability tree-based analysis, which complexity increases exponentially with K.

This work improves the job class level scheduling by reducing the complexity of the analysis. Our main contributions are as follows:

- We demonstrate that satisfying the weakly-hard constraint $\overline{\binom{w_i}{w_i+h_i}}$ for the highest-priority jobs within their respective priority classes is sufficient to ensure that task τ_i meets the constraint $\overline{\binom{m_i}{K_i}}$, where w_i represents the maximum tolerable consecutive deadline misses and h_i denotes the minimum required deadline hits following w_i.

- We propose a new job-level fixed priority scheduling approach that utilizes predefined priority classes for tasks capable of tolerating a bounded number of deadline misses.
- We show that the complexity of our scheduling analysis scales with K.
- We show that the results of our analysis are less pessimistic than the results in [4, 5].

The rest of this paper is organized as follows: the next section recalls the related work. In Section 3, we present our system model and we elaborate our problem statement. Our contribution starts in Section 4 by showing the new job-class-level scheduling algorithm. Section 5 shows the analysis for the presented scheduling algorithm. In Section 6, we evaluate our proposed scheduling and present the results. Finally, Section 7 concludes our paper.

2 Related Work

The term "weakly-hard" was introduced by Bernat et al. in [2] to characterize systems that can tolerate a limited number of deadline misses while still maintaining predictable performance. According to the weakly-hard model, tasks have the $\overline{\binom{m}{K}}$ constraints where m represents the maximum number of tolerable deadline misses in a sequence of K jobs. The notation $\overline{\binom{m}{K}}$ is older and was defined first in [8] by Hamdaoui et al. as (m, K)-firm.

Hamdaoui et al. presented the approach called Distance-based Priority (DBP) in [8]. In DBP, the priority is assigned at the job-level and its value is based on the amount of deadline misses required to violate the (m, K) constraint, which is known as distance. One issue in DBP is that the priority is assigned by just using the sequence of deadline (hits and misses) of the task without taking into account other tasks.

A Linear Programming (LP) based weakly-hard schedulability analysis for overloaded systems was introduced in [16]. This approach offers two key advantages: 1) It scales effectively with both the number of tasks and the parameter K, as it relies on an LP relaxation, and 2) It can be extended to accommodate additional scheduling policies. However, it is worth noting that this method exhibits significant pessimism when applied to small values of K [6].

Sun et al. introduced a weakly-hard schedulability analysis in [12] that determines the maximum bound on m within a time window of K consecutive jobs using Mixed Integer Linear Programming (MILP). This MILP framework evaluates all potential scenarios within the K job time window, where tasks are activated periodically. Consequently, the analysis presented in [12] can provide tight bounds on m with manageable complexity for small values of $K \leq 10$ [10].

The job-class-level scheduling presented in [4] and [5] recalled the original concept proposed by Hamdaoui et al. [8], in which each task is assigned a different priority upon meeting/missing their deadlines. Every task has a group of priorities which are assignable to its jobs. Each priority of this group is mapped to a job class of the task. A released job is assigned to a job class, and therefore taking its priority, based on the number of deadlines previously met. Every time that a job meets its deadline, the next one will be assigned to a job class with lower priority. After some amount of deadline misses happen, which is calculated based on the weakly-hard constraint, the next job is assigned to the job class with highest priority of the task.

Pazzaglia et al. [11] investigated the performance costs associated with deadline misses in control systems. Their research highlights how the distribution of deadline misses within a sequence of K jobs affects overall system performance.

Recently, Maggio et al. proposed an approach to analyze the stability of control systems under various patterns of deadline misses in [9, 13–15]. This methodology aids in deriving weakly-hard constraints, specifically by bounding m and K, thereby enhancing the understanding of how deadline misses impact system stability.

3 System Model

This paper considers independent sporadic tasks with constrained deadlines, in which a task τ_i is described using 5 parameters:

$$\tau_i \doteq (C_i, D_i, T_i, \overline{\binom{m_i}{K_i}})$$

Where C_i is the worst-case execution time of τ_i, D_i is the relative deadline of each job of τ_i (since tasks have a constrained deadline $D_i \leq T_i$), T_i i the minimum inter-arrival time between consecutive jobs of τ_i and $\overline{\binom{m_i}{K_i}}$ is the weakly-hard constraint of τ_i (m_i is the number of tolerable deadline misses in a K_i window, $m_i < K_i$ and $m_i \geq 1$). Moreover, we use similar weakly-hard constraint notations as in [2], see Table 1. A hard real-time task is characterized by $m_i = 0$ and $K_i = 1$.

Table 1: Weakly-hard constraint notations

	deadline hits	deadline misses
any order	$\binom{m_i}{K_i}$	$\overline{\binom{m_i}{K_i}}$
consecutive	$\langle \begin{smallmatrix} m_i \\ K_i \end{smallmatrix} \rangle$	$\overline{\langle \begin{smallmatrix} m_i \\ K_i \end{smallmatrix} \rangle}$

Additionally, tasks are classified based on the deadline misses tolerable in a K_i window:

Definition 3.1. Low-tolerance tasks: weakly-hard real-time tasks which require more deadline hits than tolerable misses in the K_i window, i.e. tasks with a ratio $m_i/K_i < 0.5$ and $m_i > 0$.

Definition 3.2. High-tolerance tasks: weakly-hard real-time tasks which tolerate a bigger or equal quantity of deadline misses than quantity of deadline hits in the K_i window, i.e. tasks with a ratio $m_i/K_i \geq 0.5$.

Schedulability of weakly-hard tasks.

Definition 3.3. A deadline sequence is a binary sequence of length K_i, in which 1 represents a deadline hit and 0 represents a deadline miss.

Definition 3.4. A weakly-hard task τ_i with constraint $\overline{\binom{m_i}{K_i}}$ is schedulable if, in any window of K_i consecutive invocations of the task, no more than m_i deadlines are missed.

Utilization. The utilization of a task τ_i is defined as the fraction of processor time required by its execution: $U_i = \frac{C_i}{T_i}$

The total utilization is defined as the sum of all task utilization: $U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$, where n is the number of tasks in the task set.

System-level action for missed deadlines. The proposed scheduling algorithm and schedulability analysis considers the *Job-Kill* in case of a deadline miss. In this system-level action, the job that does not meet its deadline is killed to remove load from the processor.

Problem statement. In this work, we aim to exploit the weakly-hard constraints for increasing the number of schedulable tasks on a single-core platform. Given a task set of independent weakly-hard tasks and a single-core platform, our goal is to provide a scheduling algorithm for the weakly-hard tasks and a scheduling analysis.

4 Scheduling Algorithm for Weakly-hard Real-time Tasks

In this section, we present a new job-class-level algorithm for scheduling weakly-hard tasks. We start by defining a deadline sequence that satisfies the weakly-hard constraint. Our algorithm works on enforcing the defined deadline sequence to guarantee the schedulability by assigning various priorities to released jobs. Then, we show how priorities are assigned to tasks and to released jobs.

In the next section, we show how the enforced deadline sequence facilitates the schedulability analysis.

4.1 Defining the Critical Sequence

The $\overline{\binom{m_i}{K_i}}$ constraint does not specify the distribution of the m_i deadline misses, e.g. if they could happen consecutively or not. Hence, there are different deadline sequences that satisfy the weakly-hard constraint. We are interested in one sequence that we can enforce in our scheduling algorithm such that we guarantee the satisfiability of $\overline{\binom{m_i}{K_i}}$. For that end, we define w_i and h_i.

Definition 4.1 (w_i). The maximum number of consecutive deadline misses resulting from uniformly distributing m_i in a window of K_i and it is calculated as follows:

$$w_i = \max\left(\left\lfloor \frac{m_i}{K_i - m_i} \right\rfloor, 1\right) \tag{1}$$

Definition 4.2 (h_i). The number of deadline hits required per deadline miss and it is calculated as follows:

$$h_i = \left\lceil \frac{K_i - m_i}{m_i} \right\rceil \tag{2}$$

w_i, h_i take particular values when we consider low-tolerance or high-tolerance tasks.

Lemma 4.3. *If τ_i is low-tolerance task, then $w_i = 1$. If τ_i is high-tolerance task, then $h_i = 1$.*

Proof. $m_i/K_i < 0.5 \Rightarrow \lfloor \frac{m_i}{K_i - m_i} \rfloor = 0$, hence, $w_i = 1$. Similarly, $m_i/K_i \geq 0.5 \Rightarrow \lceil \frac{K_i - m_i}{m_i} \rceil = 1$, hence, $h_i = 1$. □

Definition 4.4 (Critical sequence). It is the sequence made up of h_i consecutive deadline hits followed by w_i consecutive deadline misses.

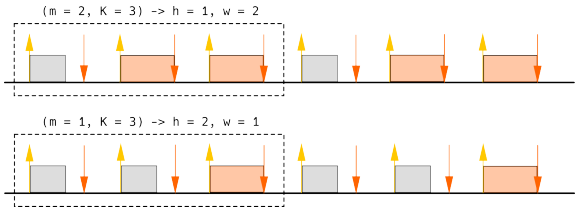


Figure 1: Critical sequence examples for a high-tolerance task (above) and a low-tolerance task (below). Yellow arrows represent task activation, while orange ones represent deadlines. Gray boxes refer to deadline hit. Boxes in orange refer to deadline miss.

Figure 1 shows two critical sequence examples, one for high-tolerance tasks and the other for low-tolerance tasks.

Our scheduling algorithm assigns a higher priority to the h_i consecutive jobs, i.e. to the jobs which require to meet their deadlines according to the critical sequence. Therefore, it is vital to prove that the critical sequence satisfies $\overline{\binom{m_i}{K_i}}$. However, the critical sequence satisfies the $\overline{\left\langle \binom{w_i}{w_i+h_i} \right\rangle}$ constraint by definition. We show now that the critical sequence also satisfies the constraint $\overline{\binom{w_i}{w_i+h_i}}$.

Lemma 4.5. *The critical sequence satisfies the weakly-hard constraint $\overline{\binom{w_i}{w_i+h_i}}$.*

Proof. For low-tolerance tasks $w_i = 1$, hence, the following holds: $\overline{\left\langle \binom{1}{1+h_i} \right\rangle} \equiv \overline{\binom{1}{1+h_i}}$. From [2], we have $\overline{\binom{w_i}{w_i+h_i}} \equiv \binom{h_i}{w_i+h_i}$. Therefore, we can change our focus for high-tolerance tasks to the deadline hits. As $h_i = 1$, hence, the following holds: $\left\langle \binom{1}{1+w_i} \right\rangle \equiv \binom{1}{1+w_i}$. □

Our goal is to prove that the critical sequence satisfies $\overline{\binom{m_i}{K_i}}$ and not only $\overline{\binom{w_i}{w_i+h_i}}$. Therefore, we have to prove that $\overline{\binom{w_i}{w_i+h_i}}$ is harder than $\overline{\binom{m_i}{K_i}}$.

Definition 4.6 ([2]). Given two constraints, λ and γ, we say that λ is harder than γ, denoted by $\lambda \preccurlyeq \gamma$, if the deadline sequences that satisfy λ also satisfy γ.

Lemma 4.7. $\overline{\binom{w_i}{w_i+h_i}} \preccurlyeq \overline{\binom{m_i}{K_i}}$.

Proof. Theorem 5 of [2] shows that if every sequence of deadline hits and misses that satisfies the constraint λ satisfies the constraint γ then $\lambda \preccurlyeq \gamma$. In our case, $\overline{\binom{w_i}{w_i+h_i}}$ is derived from $\overline{\binom{m_i}{K_i}}$ such that m_i is uniformly distributed over K_i. Hence, every sequence that satisfies $\overline{\binom{w_i}{w_i+h_i}}$, also satisfies $\overline{\binom{m_i}{K_i}}$, i.e., $\overline{\binom{w_i}{w_i+h_i}} \preccurlyeq \overline{\binom{m_i}{K_i}}$. □

Theorem 4.8. *If τ_i fulfills the constraint $\overline{\binom{w_i}{w_i+h_i}}$, it also fulfills the constraint $\overline{\binom{m_i}{K_i}}$.*

Proof. This is proven by Lemma 4.7, as $\overline{\binom{w_i}{w_i+h_i}} \preccurlyeq \overline{\binom{m_i}{K_i}}$. □

4.2 Job Classes Priorities

We assign a group of priorities to every weakly-hard task. Each job of a task is assigned only one priority from this group, i.e., job-level fixed priority. Jobs which require to meet their deadlines based on the critical sequence will receive the highest priority of the task. The other jobs will receive a lower priority. In this way, a task can reduce its priority after achieving the minimum number of deadline hits (h_i) relaxing its interference to other tasks.

The group of priorities of a task is represented by the concept of job classes coming from [5]. Each task has job classes and every job class has a different designated priority.

Definition 4.9. A task τ_i comprises $\mathcal{JC}_i = K_i - m_i + 1$ job classes, each represented by $\mathcal{JC}_i^q$, where q can take values from the range $[0, K_i - m_i]$. Job classes with lower values of q are assigned with higher priorities, i.e. $\mathcal{JC}_i^{q=0}$ and $\mathcal{JC}_i^{q=K_i-m_i}$ have the highest and lowest priority of the task, respectively.

Table 2 shows an example of three different tasks and their corresponding q range.

Table 2: Example of three tasks and their q ranges according to Def. 4.9. The last column shows the assigned priorities according to Algorithm 1.

Tasks $(C_i, D_i, T_i, \overline{(\frac{m_i}{K_i})})$	q range	Priorities
$\tau_1 = (2, 6, 6, \overline{(\frac{2}{5})})$	$[0, 3]$	$[9, 6, 3, 1]$
$\tau_2 = (3, 7, 7, \overline{(\frac{1}{3})})$	$[0, 2]$	$[8, 5, 2]$
$\tau_3 = (2, 8, 8, \overline{(\frac{2}{3})})$	$[0, 1]$	$[7, 4]$

Every job class has a different priority, i.e. the same priority is not shared between job classes of different tasks. Algorithm 1 shows how priorities are assigned to each job class. First, tasks are sorted in ascending order of deadline (Line 2). If two or more tasks share the same deadline, the task with the lower m_i is ordered first. If tasks have also the same m_i, the order between them is selected randomly. Then, the total number of priorities is calculated by counting number of job classes between all tasks (Line 5). Finally, the priority is assigned to each job class level by iterating over them (from Line 8 until Line 12).

4.3 Scheduling Algorithm

Every time a job is released, the scheduler assigns it to a job class based on the previous deadline misses/hits. We define a variable for every task named job-level jl_i used for selecting to which particular job class a job should be assigned.

Algorithm 2 shows how jl_i is calculated for a given task based on the last deadline. The inputs for the algorithm are a task and a boolean variable (named ld) that indicates if the last deadline was a hit. In the algorithm, it is observed that every task is characterized by the values of h_i, w_i, K_i and m_i. Additionally, every task also saves the number of deadline misses (*deadline_misses*) and the number of consecutive deadline hits (*deadline_hits*). *deadline_misses* is used to count misses which is required to reset jl_i to its default value $-(h_i - 1)$. *deadline_hits* counts up to h_i deadlines hits and is used to reset *deadline_misses*, allowing a task to miss w_i deadlines after h_i hits.

Algorithm 1: Priority assignment to job classes.

```
1  Input: taskset 𝒯
2  sort_tasks_ascending_deadline(𝒯)
3  for τ_i ∈ 𝒯 do
4      𝒥𝒞_i ← K_i − m_i + 1
5  𝒥𝒞 ← Σ_{∀τ_i∈𝒯} 𝒥𝒞_i
6  prio ← 𝒥𝒞
7  𝒥𝒞^max ← max{𝒥𝒞_i | ∀τ_i ∈ 𝒯}
8  for q ← 0; q < 𝒥𝒞^max; q ← q + 1 do
9      for τ_i ∈ 𝒯 do
10         if q < 𝒥𝒞_i then
11             𝒥𝒞_i^q ← prio
12             prio ← prio − 1
```

Algorithm 2: Selection of job class.

```
1  Inputs: task (τ), last deadline (ld)
2  if ld == True then
3      τ.jl ← min(τ.jl + 1, τ.K − τ.m)
4      τ.deadline_hits ← τ.deadline_hits + 1
5      if τ.deadline_hits == τ.h then
6          τ.deadline_misses ← 0
7          τ.deadline_hits ← 0
8  else
9      τ.deadline_misses ← τ.deadline_misses + 1
10     τ.deadline_hits ← 0
11     if τ.deadline_misses == τ.w then
12         τ.deadline_misses ← 0
13         τ.jl ← −(τ.h − 1)
14 τ.q ← max(0, τ.jl)
```

Furthermore, based on how jl_i is updated, the following consequences can be deduced. For high-tolerance tasks, the starting value of jl_i is zero, since for that kind of tasks h_i is one; and for low-tolerance tasks, every time a deadline is missed, jl_i is restored to $-(h_i - 1)$, since for those kind of tasks w_i is one (see Lemma 4.3).

Figure 2 shows the transitions between job classes for low-tolerance and high-tolerance tasks.

5 Scheduling Analysis

This section presents a scheduling analysis for the proposed job-class-level scheduling algorithm described in the previous section. We compute the response time for every task considering the critical instant, i.e. considering that the task is released simultaneously with all higher-priority tasks. The response time of a task is prolonged as much as tasks with higher priorities interfere with the execution of the analyzed task. Our analysis verifies that the response time is shorter than the deadline for every task in the set.

Furthermore, we consider the critical sequence for calculating the interference. We start this section explaining why the critical

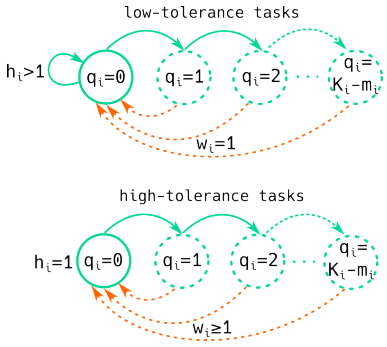


Figure 2: Job classes transitions for low-tolerance and high-tolerance tasks. Solid circles represent the highest priority, hence, jobs which are assigned to the priority represented by the solid circle are guaranteed to meet their deadlines.

sequence is important for our analysis. Next, we calculate the interference produced by the critical sequence for both high-tolerance and low-tolerance tasks. Finally, the schedulability condition for a task set is defined.

5.1 Using the Critical Sequence

First, we show that is sufficient to only take jobs in $\mathcal{JC}_i^{q=0}$ into account.

Lemma 5.1. *For a task τ_i, to meet its constraint $\overline{\binom{w_i}{w_i+h_i}}$, it is sufficient that the jobs belonging to $\mathcal{JC}_i^{q=0}$ meet their deadlines.*

Proof. Our proposed scheduling assigns h_i jobs to the $\mathcal{JC}_i^{q=0}$ every time τ_i misses w_i deadlines as Figure 2 illustrates. Hence, if all jobs belonging to $\mathcal{JC}_i^{q=0}$ meet their deadlines, τ_i meets its constraint $\overline{\binom{w_i}{w_i+h_i}}$ regardless whether the other jobs, which belong to other job classes, meet their deadlines or not. □

Lemma 5.2. *If the priority of $\mathcal{JC}_i^{q=0}$ is higher than the priority of $\mathcal{JC}_k^{q=0}$, then the jobs of τ_k in $\mathcal{JC}_k^{q=0}$ suffer interference from the jobs of τ_i in $\mathcal{JC}_i^{q=0}$.*

Proof. The algorithm in Algorithm 1 assigns a priority value to job classes starting by $q = 0$ and every time a priority is assigned, the next priority value is reduced by one. In this way, priority values of job classes $\mathcal{JC}^{q\geq 1}$ are always lower than the ones assigned to job classes $\mathcal{JC}^{q=0}$. From which it follows that jobs of a task τ_k which belongs to job class $\mathcal{JC}_k^{q=0}$ suffers interference of other jobs in $\mathcal{JC}_i^{q=0}$, only if the priority of $\mathcal{JC}_i^{q=0}$ is higher than the priority of $\mathcal{JC}_k^{q=0}$. □

Consequently, our analysis considers only the jobs in job classes $\mathcal{JC}_i^{q=0}$.

Furthermore, considering the critical sequence for analysing the interference corresponds with the highest induced interference over lower-priority jobs. Hence, the following Lemma:

Lemma 5.3. *The maximum interference induced by a task occurs when its jobs follow the critical sequence.*

Proof. The critical sequence contains the maximum number of consecutive deadline misses allowed. Only after this amount of deadline misses occurs, the task is executed with its highest priority. Therefore, the maximum number of consecutive highest priority jobs are contained in the critical sequence. Hence, the interference produced by the critical sequence is bigger than the produced by other sequences which satisfy the $\overline{\binom{w_i}{w_i+h_i}}$. □

5.2 Interference Produced by the Critical Sequence

Let us start by recalling how to calculate the interference to task τ_k over the interval of time t:

$$I_{\tau_k}(t) = \sum_{i \neq k} I_{i,k}(t) \tag{3}$$

Where $I_{i,k}(t)$ is the interference of the task τ_i over the task τ_k for the interval t:

$$I_{i,k}(t) = \left\lceil \frac{t}{T_i} \right\rceil C_i \tag{4}$$

For simplification, we use $I_{i,k}(t) = I_i(t)$ sometimes in this paper.

For Task-Level Fixed Priority (TLFP), only the tasks with higher priority than τ_k interfere, therefore, $I_k(t)$ is reduced to:

$$I_{\tau_k}(t) = \sum_{i \in hp(k)} I_{i,k}(t) \tag{5}$$

Here, $hp(k)$ is the set of tasks that have higher priority than τ_k.

Now, we update Equation (4) for high-tolerance tasks considering the critical sequence. Remember that the critical sequence for a high-tolerance task τ_i is w_i deadline misses after a single deadline hit. This allows to consider the interference generated by such a task as the same produced by a task with a longer inter-arrival time that is equal to $(w_i + 1)T_i$. Formally writing this:

Lemma 5.4. *The interference generated by a high-tolerance task τ_i with constraint $\overline{\binom{m_i}{K_i}}$ can be calculated as if it were coming from an equivalent hard-real time task $\tau_i^{eq} = (C_i, D_i, (w_i + 1)T_i)$.*

Proof. The jobs of the hard real-time task $\tau_i^{eq} = (C_i, D_i, (w_i + 1)T_i)$ have the same worst-case execution time C_i as the jobs of τ_i in $\mathcal{JC}_i^{q=0}$, and occur at the same inter-arrival time $(w_i + 1)T_i$. Therefore, they induce the same interference. □

Next, we calculate the interference.

Lemma 5.5. *The interference coming from a high-tolerance task τ_i within the window of size t is: $I_i(t) = \left\lceil \frac{t}{(w_i+1)T_i} \right\rceil C_i$*

Proof. Lemma 5.4 allows to consider the interference coming from τ_i as it were coming from a task with longer inter-arrival time which is equal to $(w_i + 1)T_i$. □

We show next the interference from low-tolerance tasks considering the critical sequence. For low-tolerance tasks, the minimum inter-arrival time between two jobs belonging to $\mathcal{JC}_i^{q=0}$ is T_i. Therefore, bounding the interference of low-tolerance tasks requires considering T_i as inter-arrival time. However, we also need to exclude the jobs that do not belong to $\mathcal{JC}^{q=0}$, i.e. the lower-priority jobs. The following Lemma shows the number of jobs excluded to calculate the interference.

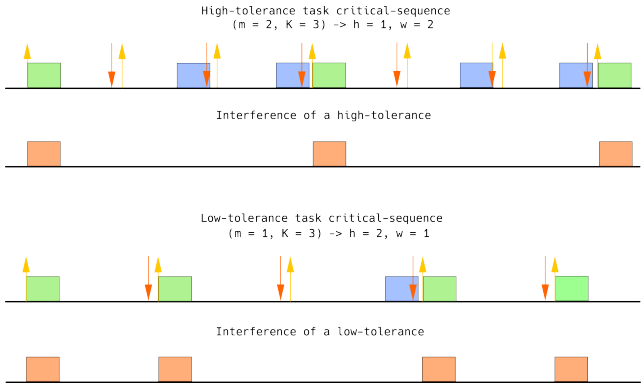


Figure 3: Interference due to the critical sequence coming from a high-tolerance task (above) and a low-tolerance task (below). Green boxes refer to jobs in $\mathcal{JC}_i^{q=0}$. Boxes in blue refer to lower priority jobs. Orange boxes represents the interference seeing by a lower priority task.

Lemma 5.6. *Following the critical sequence, there are*

$$O_i(t) = \left\lfloor \frac{t}{(h_i+1)T_i} \right\rfloor \tag{6}$$

jobs that belong to a priority class different from $\mathcal{JC}_i^{q=0}$ over the interval of time t.

Proof. From the critical sequence for low-tolerance tasks, there are h_i jobs of τ_i should meet their deadlines and only after a deadline miss is tolerated. Therefore, the time interval in which a job misses its deadline is $(h_i+1)T_i$. □

Finally, we calculate the interference generated from a low-tolerance task as follows:

Lemma 5.7. *The interference coming from a low-tolerance task τ_i within a time interval t is:*

$$I_i(t) = (N_i(t) - O_i(t))C_i \tag{7}$$

where $N_i(t) = \left\lceil \frac{t}{T_i} \right\rceil$, i.e. the total number of jobs in the interval of time t and $O_i(t)$ as in (6).

Proof. Within the interval t there are no more than $N_i(t)$ jobs, out of which there are $O_i(t)$ jobs that do not belong to $\mathcal{JC}_i^{q=0}$. Hence, it is safe to consider only $N_i(t) - O_i(t)$ jobs to bound the interference. □

Figure 3 shows the interference coming from a high-tolerance and a low-tolerance task.

5.3 Schedulability Condition

Our schedulability analysis is based on the Deadline Monotonic (DM) guarantee [1] which verifies that the response time of every tasks in the set is shorter than its deadline. For analysing the response time of a task, the DM guarantee considers the critical instant which happens when the task is released simultaneously with all higher-priority tasks.

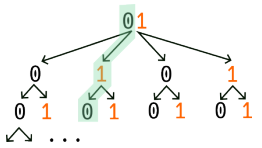


Figure 4: Binary trees created for counting deadline sequences (example of sequence highlighted).

Furthermore, we modified the algorithm in [1] by introducing the interference coming from high-tolerance tasks (Lemma 5.5) and from low-tolerance tasks (Lemma 5.7).

Finally, we show the schedulability condition for one task and then we extend it to the set.

Theorem 5.8. *A weakly-hard real-time task τ_k is schedulable by our algorithm if the response time of its jobs in $\mathcal{JC}_k^{q=0}$ is shorter than D_k.*

Proof. Lemma 5.2 shows that jobs in $\mathcal{JC}_k^{q=0}$ are interfered only by other jobs in $\mathcal{JC}_i^{q=0}$ when these last have a higher priority. Meaning that, the response time of jobs in $\mathcal{JC}_k^{q=0}$ is prolonged as much as other jobs in $\mathcal{JC}_i^{q=0}$ with higher priority interfere. Moreover, jobs in $\mathcal{JC}_k^{q=0}$ meet their deadlines if the interference over them allows them to finish before its deadline (D_k). Finally, Lemma 5.1 shows that a task τ_k meets its constraint when its jobs in $\mathcal{JC}_k^{q=0}$ meet their deadlines. □

Corollary 5.9. *A task set of weakly-hard real-time tasks is schedulable by Algorithm 2 if Theorem 5.8 is satisfied for every task in the set.*

6 Evaluation

Our scheduling algorithm is based on transforming the weakly-hard constraint $\overline{\binom{m_i}{K_i}}$ into the constraint $\overline{\binom{w_i}{w_i+h_i}}$, which operates over a smaller window. In this section, we begin by highlighting the limitations that arise from adopting a more stringent constraint than $\overline{\binom{m_i}{K_i}}$.

Next, our scheduling analysis is compared against the Integer Linear Programming (ILP) analysis in [12] and the Job-Class-Level (JCL) in [5]. The experiments are based on the analysis of task sets randomly generated using the UUnifast algorithm [3]. For a given total utilization, we calculate the percentage of schedulable task sets, known as schedulability ratio, and measure the required computation time.

6.1 Transformation Cost

To analyze the limitations introduced by transforming the constraint $\overline{\binom{m_i}{K_i}}$ into the harder constraint $\overline{\binom{w_i}{w_i+h_i}}$, we examine the possible deadline sequences that satisfy both constraints. For counting the deadline sequences, we create a binary tree with depth of K. Later, the solutions for $\overline{\binom{m_i}{K_i}}$ and $\overline{\binom{w_i}{w_i+h_i}}$ are counted from the branches, see Fig. 4.

Results for various values of $\overline{\binom{m_i}{K_i}}$ are presented in Table 3. For low-tolerance tasks, the number of omitted deadline sequences

Table 3: Limitation for harder constraints

$\overline{\binom{m_i}{K_i}}$	$\overline{\binom{w_i}{w_i+h_i}}$	$\overline{\binom{w_i}{w_i+h_i}}$ solutions / $\overline{\binom{m_i}{K_i}}$ solutions
$\overline{\binom{1}{5}}$	$\overline{\binom{1}{5}}$	1.0
$\overline{\binom{2}{5}}$	$\overline{\binom{1}{3}}$	0.5625
$\overline{\binom{3}{5}}$	$\overline{\binom{1}{2}}$	0.5
$\overline{\binom{4}{5}}$	$\overline{\binom{4}{5}}$	1.0
$\overline{\binom{4}{10}}$	$\overline{\binom{1}{3}}$	0.1554
$\overline{\binom{8}{10}}$	$\overline{\binom{4}{5}}$	0.9003
$\overline{\binom{8}{20}}$	$\overline{\binom{1}{3}}$	0.01040
$\overline{\binom{16}{20}}$	$\overline{\binom{4}{5}}$	0.7511

can be significantly high when m_i/K_i approaches 0.5. In contrast, high-tolerance tasks exhibit fewer omitted sequences. Additionally, for both types of tasks, the number of omitted sequences increases when considering constraints that are multiples of one another, such as $\overline{\binom{m_i}{K_i}} = \overline{\binom{8}{20}}$ and $\overline{\binom{m_i}{K_i}} = \overline{\binom{2}{5}}$. This phenomenon occurs because certain deadline sequences with consecutive deadline misses are not accounted. Despite this limitation, our experiments show better results than ILP [12] and JCL [5]. Furthermore, the results regarding the transformation cost presented here are theoretical, as not all uncounted deadline sequences necessarily correspond to schedulable solutions.

6.2 Scheduling Analysis Setup

Task sets are generated for a specified range of total utilization values using the UUnifast algorithm [3]. The utilization of the generated tasks does not exceed one. For each total utilization value, 1000 task sets are created. The schedulability ratio is calculated, and the computation time is recorded for each generated task set. Additionally, the generated tasks are assigned implicit deadlines, meaning their deadlines are equal to their inter-arrival times (i.e., $D_i = T_i$).

We compare the results of our scheduling analysis (here labeled as RTA WH) against the Integer Linear Programming (ILP) analysis [12] and the Job-Class-Level (JCL) proposed in [5]. The first experiment compares the result of three analysis. For this experiment, the range of total utilization used is from 0.1 to 1 due to the longer computation times of the ILP analysis for utilization bigger than 1. Also, the value used for K_i is 5 due to the scalability issues of the ILP analysis [10]. We run the three analysis on sets of 30 tasks, which are all either low-tolerance or high-tolerance. Moreover, we use $m_i = 1$ for low-tolerance tasks and $m_i = 4$ for high-tolerance tasks.

In the rest of the experiment, we compare our analysis (RTA WH) only against JCL. The values for m_i are chosen randomly between the values that fulfill the desired m_i/K_i. For example, given $K_i = 5$, m_i can be 1 or 2 for low-tolerance tasks; and 3 or 4 for high-tolerance tasks. In the legends, "high" denotes sets of high-tolerance tasks while "low" refers sets of low-tolerance tasks. The range of total utilization starts from 0.1 and ends at 2.5. In order to compare scalability, we use different amount of tasks per set (30 and 100) and different values of K_i (5 and 10).

We used a desktop computer with a Intel(R) Core(TM) i7-8700 processor (6 cores, 2 threads per core) and 32GB of RAM running Ubuntu 22.04 to run the experiments.

6.3 Scheduling Analysis Experiments

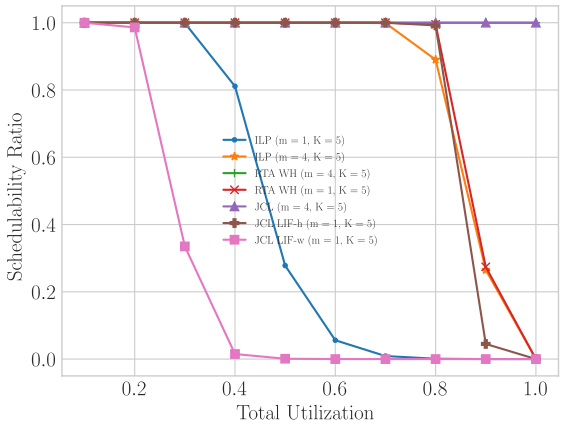


Figure 5: Schedulability ratio for ILP, JCL and RTA WH (K = 5) with maximal utilization used is U = 1

Figure 5 shows the schedulability ratio for ILP, JCL and our analysis (RTA WH) using sets with 30 tasks. In case of JCL, both priority assignment algorithms ($LIF-w$ and $LIF-h$) are used. $LIF-w$ assigns priorities to job classes $\mathcal{JC}_i^{q=0}$ based on the deadline and then based on w_i for the remaining job classes. $LIF-h$ improves $LIF-w$ for low-tolerance tasks by repeating the assigned priorities h_i times between the job classes. Since $LIF-h$ does not change the assigned priorities for high-tolerance tasks, there is no difference between $LIF-w$ and $LIF-h$ for high-tolerance tasks. Therefore, we do not label which priority assignment is used for high-tolerance tasks. For low-tolerance tasks, the results for ILP are better than for JCL $LIF-w$ but using the priority assignment $LIF-h$ improves the schedulability. Then, our analysis shows better schedulability ratio than JCL $LIF-h$. For high-tolerance tasks, JCL and our approach show similar results while ILP does not show any schedulable set after utilization higher than 1.

Figure 6, Figure 8 and Figure 9 show the computation times for ILP, JCL and our approach respectively. Overall, ILP computation times are longer than the rest. In particular, ILP starts taking longer from $U = 0.4$ because no complicated calculation is required for lower utilizations. JCL $LIF-h$ is faster than JCL $LIF-w$ to reject unschedulable sets (this is observed for values of $U = 0.9$ and $U = 1$). Also, JCL $LIF-h$ takes up to 40 milliseconds. Our scheduling analysis is the fastest one, always taking less than 2 milliseconds and generally around tens of microseconds.

Figure 7 shows a schedulability ratio analysis extended until $U = 2.5$ for JCL and our analysis. In this experiment, we observe the same behavior as before, i.e. our approach is better than JCL $LIF-h$ for low-tolerance tasks and equivalent than JCL for high-tolerance tasks. The reason our approach is better for low-tolerance

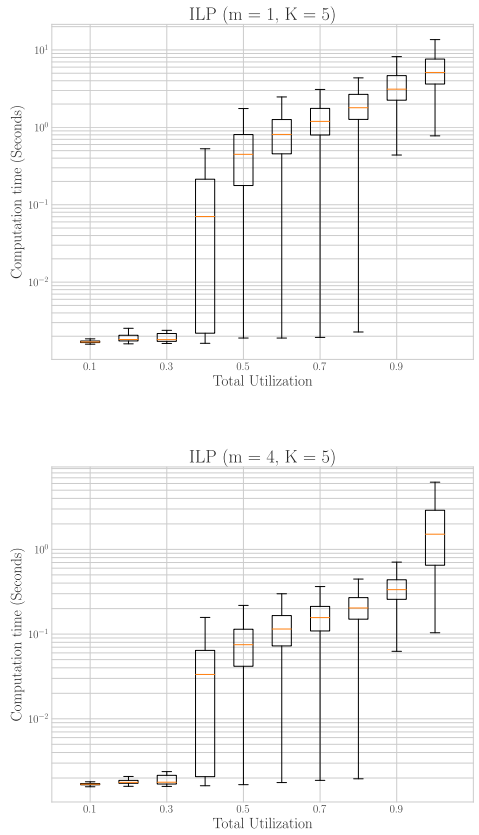


Figure 6: Computation time for ILP (K = 5) with maximal utilization used is U = 1

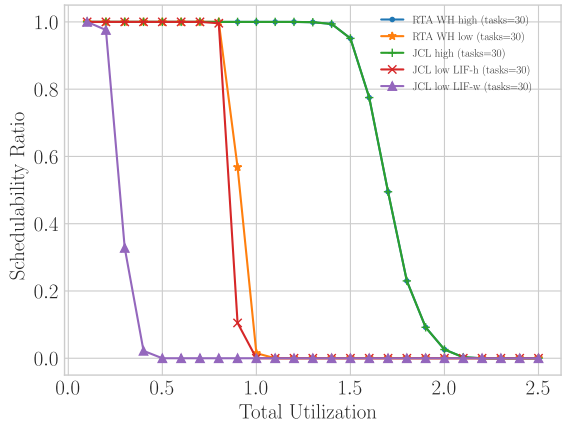


Figure 7: Schedulability ratio for JCL and RTA WH (K = 5). Note that the green curve is above the blue one.

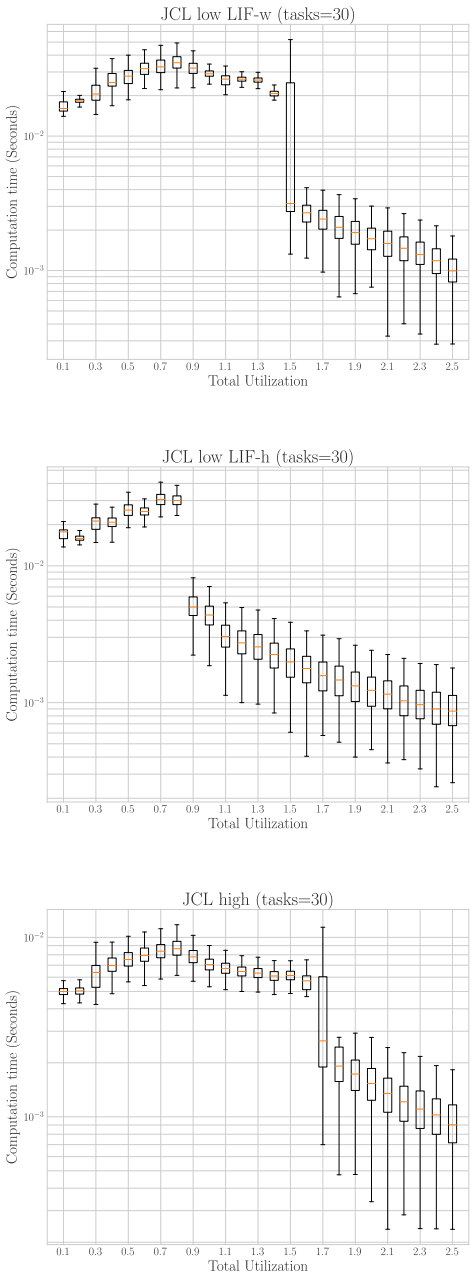


Figure 8: Computation time for JCL (K = 5).

tasks is because JCL $LIF - h$ considers the interference of all job-classes with the repeated priority (there is no repeated priority between job classes in our approach). Additionally, high-tolerance tasks seem to be schedulable even after the practical limit ($U = 1$). This is explained by Lemma 5.4, which allows us to consider the interference coming from the high-tolerance tasks as it were coming from a task with longer inter-arrival time, reducing the utilization

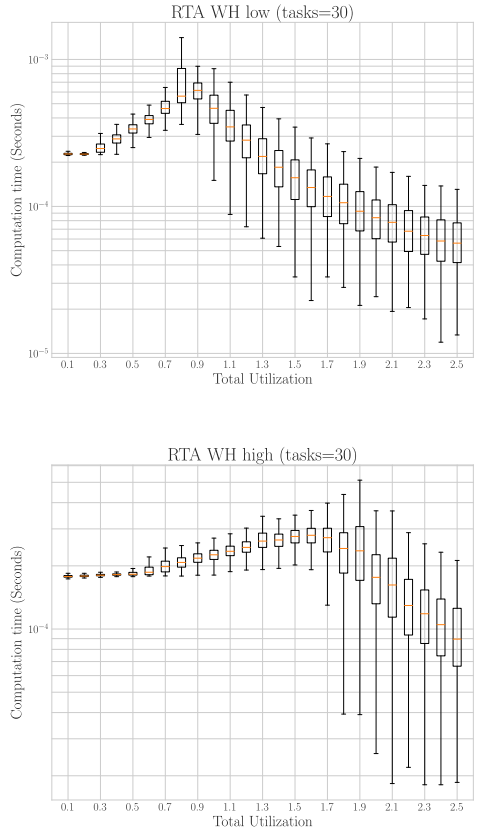


Figure 9: Computation time for RTA WH (K = 5)

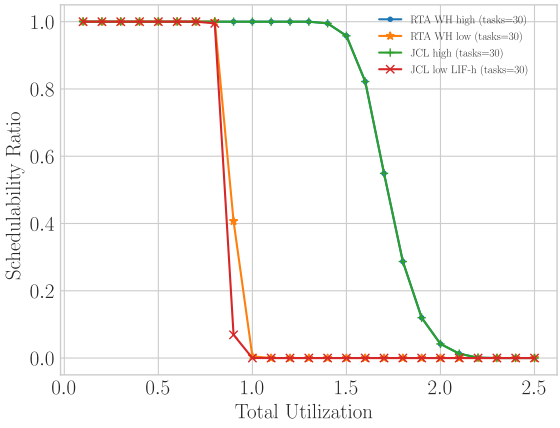


Figure 10: Schedulability ratio for JCL and RTA WH (K = 10). Similar than before, the blue and green curves are overlapped

of the task. This also explains why this behavior is not seen for low-tolerance tasks.

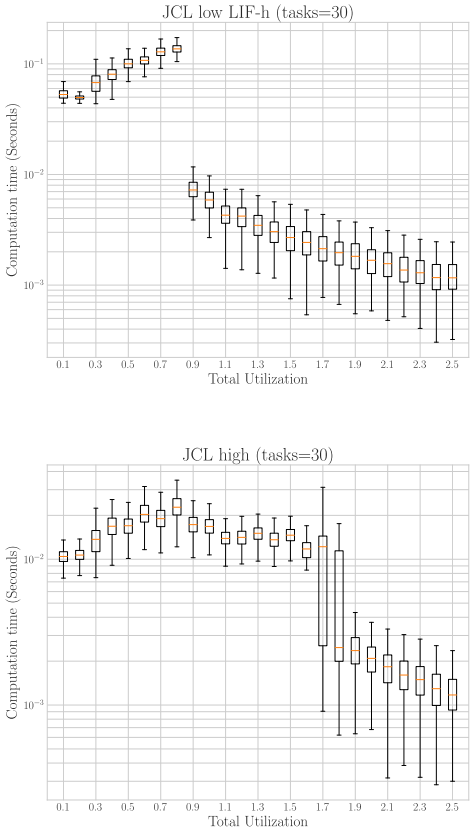


Figure 11: Computation time for JCL (K = 10).

We executed experiments for sets of 100 tasks with $K_i = 5$. JCL and our approach for high-tolerance tasks increase the schedulability ratio for values higher than $U = 1.5$ in comparison with sets of 30 tasks. The reason is that having more tasks, while keeping the same total utilization, makes the tasks more lightweight which reduces the interference between them. However, it was observed that the computation time increases by no more than 10 times.

Figure 10 shows the schedulability ratio for sets with 30 tasks and $K_i = 10$. There is no observable difference with the schedulability ratio for sets with 30 tasks and $K_i = 5$. However, as it is shown in Figure 11, the computation times for JCL $LIF-h$ increases significantly (up to around 100 milliseconds). The longer computation times reflect the increase of the complexity due to the size of the reachability trees. In case of JCL for high-tolerance tasks, the computation times also increase but remaining below 40 milliseconds. Finally, the computation times of our scheduling analysis are not significantly affected by increasing the value of K_i remaining bellow 1 millisecond.

7 Conclusion

In real-time systems, if few deadline misses are tolerable, leveraging the weakly-hard model can reduce the over-provisioning. This paper proposed a new job class scheduling for weakly-hard

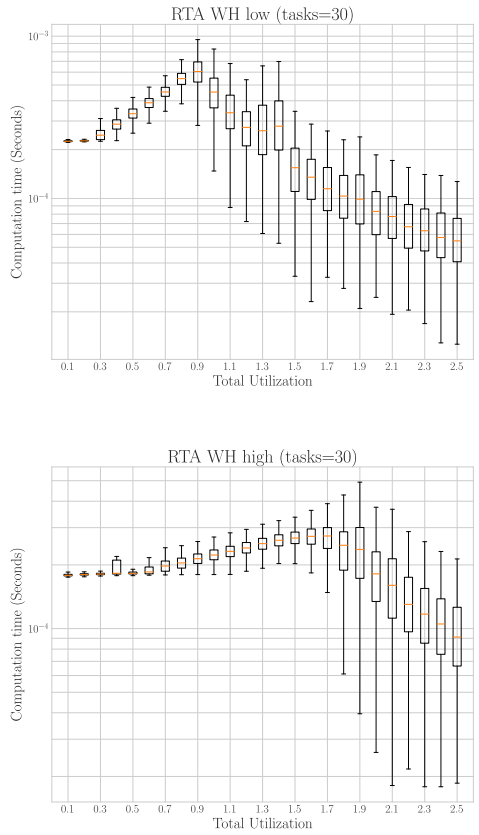


Figure 12: Computation time for RTA WH (K = 10).

real-time tasks, appended with a schedulability test. The scheduling algorithm exploits the tolerable deadline misses by assigning different priorities to jobs upon urgency of meeting their deadline. Such job-level priority assignment reduces the interference with low-priority tasks and helps them to satisfy their weakly-hard constraints. The proposed schedulability analysis utilizes neither ILP nor reachability tree-based analysis, as similar approaches in the literature. Rather, it focuses on verifying the schedulability of the maximum tolerable consecutive deadline misses.

Our experiments show that the computation time of the proposed analysis scales with K_i being always faster than the state-of-the-art approaches, which complexity increases with K_i due to the size of the recheability tree.

Furthermore, our future work will contemplate a reduction in the limitations for low-tolerance tasks, different system-level actions for reacting to deadline misses and the interference between tasks due to shared resources.

References

[1] Neil C Audsley, Alan Burns, Mike F Richardson, and Andy J Wellings. 1991. Hard real-time scheduling: The deadline-monotonic approach. *IFAC Proceedings Volumes* 24, 2 (1991), 127–132.

[2] Guillem Bernat, Alan Burns, and Albert Liamosi. 2001. Weakly hard real-time systems. *IEEE transactions on Computers* 50, 4 (2001), 308–321.

[3] Enrico Bini and Giorgio C Buttazzo. 2005. Measuring the performance of schedulability tests. *Real-Time Systems* 30, 1 (2005), 129–154.

[4] Hyunjong Choi, Hyoseung Kim, and Qi Zhu. 2019. Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 241–253. https://doi.org/10.1109/RTAS.2019.00028

[5] Hyunjong Choi, Hyoseung Kim, and Qi Zhu. 2021. Toward practical weakly hard real-time systems: A job-class-level scheduling approach. *IEEE Internet of Things Journal* 8, 8 (2021), 6692–6708.

[6] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. 2019. Weakly-Hard Real-Time Guarantees for Earliest Deadline First Scheduling of Independent Tasks. *ACM Trans. Embed. Comput. Syst.* 18, 6, Article 121 (dec 2019), 25 pages. https://doi.org/10.1145/3356865

[7] Zain A. H. Hammadeh, Sophie Quinton, Marco Panunzio, Rafik Henia, Laurent Rioux, and Rolf Ernst. 2017. Budgeting under-specified tasks for weakly-hard real-time systems. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

[8] Moncef Hamdaoui and Parameswaran Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE transactions on Computers* 44, 12 (1995), 1443–1451.

[9] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein. 2020. Control-system stability under consecutive deadline misses constraints. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[10] Marco Di Natale. 2017. Beyond the m-k model: restoring performance considerations in the time abstraction. ESWEEK - Tutorial Slides.

[11] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. 2018. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, Vol. 106. 10:1–10:22. https://doi.org/10.4230/LIPIcs.ECRTS.2018.10

[12] Youcheng Sun and Marco Di Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 171 (sep 2017), 19 pages. https://doi.org/10.1145/3126497

[13] Nils Vreman, Anton Cervin, and Martina Maggio. 2021. Stability and Performance Analysis of Control Systems Subject to Bursts of Deadline Misses. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 196)*, Björn B. Brandenburg (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 15:1–15:23. https://doi.org/10.4230/LIPIcs.ECRTS.2021.15

[14] Nils Vreman, Richard Pates, and Martina Maggio. 2022. WeaklyHard.jl: Scalable Analysis of Weakly-Hard Constraints. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 228–240. https://doi.org/10.1109/RTAS54340.2022.00026

[15] Nils Vreman, Paolo Pazzaglia, Victor Magron, Jie Wang, and Martina Maggio. 2022. Stability of Linear Systems Under Extended Weakly-Hard Constraints. *IEEE Control Systems Letters* 6 (2022), 2900–2905. https://doi.org/10.1109/LCSYS.2022.3179960

[16] Wenbo Xu, Zain A. H. Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. 2015. Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis. In *2015 27th Euromicro Conference on Real-Time Systems*. 247–256. https://doi.org/10.1109/ECRTS.2015.29