Global Scheduling of Weakly-Hard Real-Time Tasks using Job-Level Priority Classes

V. GABRIEL MOYANO, German Aerospace Center (DLR), Germany
ZAIN A. H. HAMMADEH, German Aerospace Center (DLR), Germany
SELMA SAIDI, Technische Universität Braunschweig, Germany
DANIEL LÜDTKE, German Aerospace Center (DLR), Germany

Real-time systems are intrinsic components of many pivotal applications, such as self-driving vehicles, aerospace and defense systems. The trend in these applications is to incorporate multiple tasks onto fewer, more powerful hardware platforms, e.g., multi-core systems, mainly for reducing cost and power consumption. Many real-time tasks, like control tasks, can tolerate occasional deadline misses due to robust algorithms. These tasks can be modeled using the weakly-hard model. Literature shows that leveraging the weakly-hard model can relax the over-provisioning associated with designed real-time systems. However, a wide-range of the research focuses on single-core platforms. Therefore, we strive to extend the state-of-the-art of scheduling weakly-hard real-time tasks to multi-core platforms. We present a global job-level fixed priority scheduling algorithm together with its schedulability analysis. The scheduling algorithm leverages the tolerable continuous deadline misses to assigning priorities to jobs. The proposed analysis extends the Response Time Analysis (RTA) for global scheduling to test the schedulability of tasks. Hence, our analysis scales with the number of tasks and number of cores because, unlike literature, it depends neither on Integer Linear Programming nor reachability trees. Schedulability analyses show that the schedulability ratio is improved by 40% comparing to the global Rate Monotonic (RM) scheduling and up to 60% more than the global EDF scheduling, which are the state-of-the-art schedulers on the RTEMS real-time operating system. Our evaluation on industrial embedded multi-core platform running RTEMS shows that the scheduling overhead of our proposal does not exceed 60 nanosecond.

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

Additional Key Words and Phrases: weakly-hard, multi-core, real-time, global scheduling

ACM Reference Format:
V. Gabriel Moyano, Zain A. H. Hammadeh, Selma Saidi, and Daniel Lüdtke. 2025. Global Scheduling of Weakly-Hard Real-Time Tasks using Job-Level Priority Classes. *ACM Trans. Embedd. Comput. Syst.* 1, 1, Article 1 (January 2025), 28 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Enabling more autonomy in the automotive and the aerospace domains requires involving sophisticated control algorithms with real-time requirements. This is the case of CALLISTO (Cooperative Action Leading to Launcher Innovation for Stage Tossback Operation) [9, 11]; a joint project between the French National Center for Space Studies (CNES), the German Aerospace Center

Authors' Contact Information: V. Gabriel Moyano, gabriel.moyano@dlr.de, German Aerospace Center (DLR), Germany; Zain A. H. Hammadeh, zain.hajhammadeh@dlr.de, German Aerospace Center (DLR), Germany; Selma Saidi, saidi@ida.ing.tu-bs.de, Technische Universität Braunschweig, Germany; Daniel Lüdtke, daniel.luedtke@dlr.de, German Aerospace Center (DLR), Germany.

ACM 1558-3465/2025/1-ART1
https://doi.org/XXXXXXX.XXXXXXX

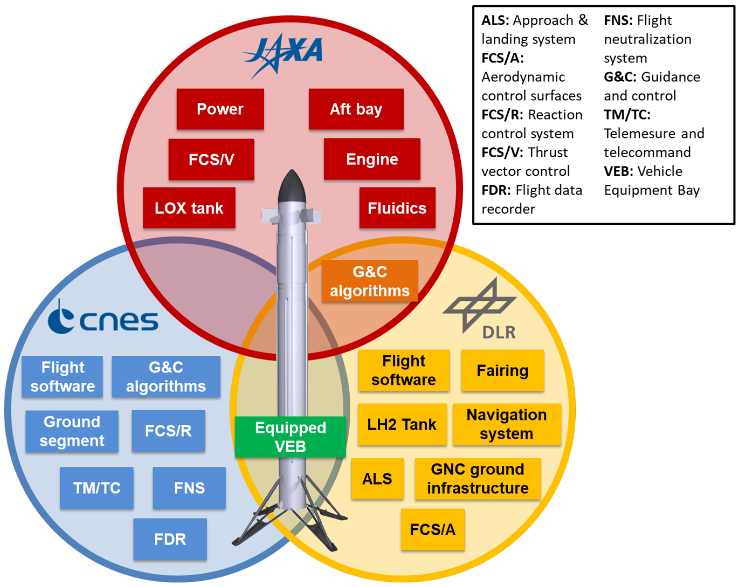
JAXA
Power
Aft bay
FCS/V
Engine
LOX tank
Fluidics
G&C algorithms
cnes
Flight software
G&C algorithms
Ground segment
FCS/R
TM/TC
FNS
FDR
Equipped VEB
DLR
Flight software
Fairing
LH2 Tank
Navigation system
ALS
GNC ground infrastructure
FCS/A
ALS: Approach & landing system
FCS/A: Aerodynamic control surfaces
FCS/R: Reaction control system
FCS/V: Thrust vector control
FDR: Flight data recorder
FNS: Flight neutralization system
G&C: Guidance and control
TM/TC: Telemesure and telecommand
VEB: Vehicle Equipment Bay

Fig. 1. CALLISTO project.

(DLR), and the Japan Aerospace Exploration Agency (JAXA) for developing and building a Vertical-Takeoff/Vertical-Landing rocket that can be reused (Figure 1). The on-board computers in CALLISTO carry-out the state estimation, based on fusing information of several sensors [16], and the execution of the control algorithms.

Furthermore, computing hard real-time guarantees for the developed tasks under worst-case scenarios comes at the cost of exacerbating over-provisioning in such new software-based embedded systems, which implies, for example, higher power consumption. However, literature [8] shows that leveraging the weakly-hard model can relax the over-provisioning associated with designed real-time systems. The weakly-hard real-time model [1] extends the tight region of schedulable tasks, which is defined by the hard real-time model, by exploiting the tolerable deadline misses. Moreover, many papers, e.g. [13, 15, 19], demonstrated that the control systems can tolerate occasional deadline misses with a small amount of performance degradation and they describe the tolerance using the weakly-hard model. Although CALLISTO operates as a hard real-time system, we are exploring potential relaxations by permitting tolerable deadline misses.

Recently, weakly-hard real-time systems received a lot of attention, and several schedulability analyses have been proposed [5, 17, 22]. In general, to compute weakly-hard real-time guarantees, the analysis should not only consider the job(s) in the worst-case scenario, but also 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 to more pessimism. Also, weakly-hard constraints are effective on the job-level, rather than the task-level. Therefore, satisfying such constraints presuppose the job-level schedulability as a more efficient approach than task-level schedulability. Hence, we propose a job-level fixed priority scheduling algorithm.

Furthermore, embedded system developers in the automotive and aerospace industries are turning to multi-core platforms to meet the growing demand for computational performance, a trend that also applies to CALLISTO. However, multi-core systems were out of scope in the majority of papers that address weakly-hard real-time systems. In this work, we fill the gap by proposing a global job-level scheduling algorithm which exploits the tolerable deadline misses together with a schedulability analysis to compute weakly-hard real-time guarantees. Our main contributions are the following:

- We reduce the $\overline{\binom{m_i}{K_i}}$ constraint to a harder one which defines the maximum tolerable number of consecutive deadline misses after a minimum number of required deadline hits
- We propose a global job-level fixed priority scheduling using predefined priority classes for tasks which can tolerate a bounded number of deadline misses
- We prove that satisfying the new constraint for jobs that have the highest priority in the priority classes is sufficient to guarantee that the task τ_i satisfies the constraint $\overline{\binom{m_i}{K_i}}$
- We propose a schedulability analysis for the proposed global scheduling. Our schedulability analysis extends the Response Time Analysis (RTA) [2]

In our proposed analysis, we compromise between pessimism and complexity. Using a harder constraint instead of $\overline{\binom{m_i}{K_i}}$ limits the number of deadline sequences that satisfy the weakly-hard constraint. However, we need neither a reachability tree-based analysis nor an Integer Linear Programming (ILP) based analysis, e.g., [17]. Therefore, our analysis has less complexity than [4, 5] and scales with the number of tasks because we use RTA [2]. The proposed scheduling can be seen as a fault-tolerance mechanism in which the deadline-miss is an error and assigning a higher priority to the next job as a mitigation mechanism. However, the problem will be studied from real-time schedulability perspective.

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 elaborate our problem statement. Section 4 recalls the Response Time Analysis (RTA). Our contribution starts in Section 5 by showing the global scheduling algorithm for weakly-hard real-time tasks. Then, the analysis for the presented scheduling algorithm is shown in Section 6. Also, we experimentally evaluate our proposed scheduling and present the results in Section 7. Finally, Section 8 concludes our paper.

2 Related Work

Weakly-hard real-time constraints define the maximum number of deadline misses that a task can tolerate before going into a faulty state. The term weakly-hard was coined by Bernat et al. in [1] to describe systems in which 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 $\overline{\binom{m}{K}}$, though, is a bit older and was defined in [7] by Hamdaoui et al as (m, K)-firm. Since 2014, the number of papers addressing the weakly-hard real-time systems has increased significantly.

Pazzaglia et al. [15] researched the performance cost of deadline misses in control systems. They have shown the performance impact of the distribution of deadline misses within the sequence of K jobs. Liang et al. [12] presented a fault tolerance mechanism for weakly-hard. Recently, Maggio et al. proposed in [13, 19–21] an approach to analyze the stability of control systems under different patterns of deadline misses. The proposed approach by Maggio et al. can help in extracting the weakly-hard constraints, i.e., bounding m and K. The authors considered a system model of single-core platforms and periodic control tasks.

Sun et al. presented in [17] a weakly-hard schedulability analysis that uses Mixed Integer Linear Programming (MILP) for computing the maximum bound on m within a time window of K consecutive jobs. The MILP checks all possible scenarios within a time window of K consecutive jobs where tasks are periodically activated. The analysis in [17] can, therefore, provide tight bounds on m with reasonable complexity for small $K \leq 10$ [14].

A Linear Programming (LP) based weakly-hard schedulability analysis has been presented in [22] for overloaded systems. This approach considers temporarily overloaded systems due to rare

Table 1. Key mathematical notations used in this work.

Notation	Description
n_c	number of cores in the system
w_i	maximum consecutive number of deadline misses to uniformly distribute m_i in a window of K_i
h_i	deadline hits required per deadline miss
$\mathcal{JC}_i^q$	job-class q of task τ_i
jl_i	job-level variable of task τ_i
s_i	slack of a task τ_i
$N_i(L)$	number of jobs interfering in the time interval L
$O_i(L)$	number of jobs not interfering in the time interval L
a_i	variable for counting or not the interference of the carry-out job

sporadic jobs. It bounds the impact of sporadic overload jobs on tasks –assumed to be schedulable in the non-overloaded intervals– in terms of deadline misses. This approach has two features: 1) It scales with K and number of tasks because it depends on an LP relaxation. 2) It is extendable for more scheduling policies. However, this approach reports a high pessimism for small K [6].

Wu and Jin proposed in [18] a global scheduling algorithm for multimedia streams. They applied the Distance Based Priority (DBP) algorithm [7] to a global scheduler. In their approach, a task that is close to violate its $\overline{\binom{m_i}{K_i}}$ constraint is assigned dynamically the highest priority. In [10], Kong and Cho computed bounds on the probability of not satisfying the $\overline{\binom{m_i}{K_i}}$ constraint and proposed a dynamic hierarchical scheduling algorithm to improve the quality of service. The goal of our paper is different from [18] and [10] because we aim to exploit the weakly-hard constraints to increase the load that can be scheduled on a multi-core system under job-level fixed priority scheduling.

The job-class-level scheduling presented in [4] and [5] recalled the original concept proposed by Hamdaoui et al. [7], in which each task is assigned a different priority upon meeting/missing their deadlines. The proposed scheduling in [4, 5] is dedicated to single-core systems. The authors showed how to extend the job-class-level scheduling to semi-partitioned multi-core scheduling. Our work extends the job-class-level scheduling to global multi-core scheduling. However, our schedulability analysis does not depend on a reachability tree as the one in [4, 5].

3 System Model

This paper considers independent sporadic tasks with constrained deadlines and preemptive scheduling. A task set is executed on a Symmetric Multi-Processing (SMP) multi-core platform. Table 1 shows the notations used in this work.

Task model. A task τ_i is described using 5 parameters:

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

- C_i: The worst-case execution time of τ_i.
- D_i: The relative deadline of each job of τ_i. Since tasks have a constrained deadline $D_i \leq T_i$.
- T_i: The minimum inter-arrival time between consecutive jobs of τ_i.
- $\overline{\binom{m_i}{K_i}}$: The weakly-hard constraint of τ_i, where m_i is the number of tolerable deadline misses in a K_i window, where $m_i < K_i$ and $m_i \geq 1$. A hard real-time task is characterized by $m_i = 0$ and $K_i = 1$.

In this paper, we use similar weakly-hard constraint notations as in [1]. We use parentheses to denote deadline misses/hits in any order, angle brackets for consecutive deadline misses/hits, and a bar to differentiate deadline misses, see Table 2. Please note that according to [1, Theorem 4], there is no need to specify a window size in case of the consecutive deadline misses notation.

Table 2. Weakly-hard constraint notations.

	deadline hits	deadline misses
any order	$\binom{m_i}{K_i}$	$\overline{\binom{m_i}{K_i}}$
consecutive	$\left\langle {m_i \atop K_i} \right\rangle$	$\overline{\left\langle {m_i \atop K_i} \right\rangle} \equiv \overline{\langle m_i \rangle}$

We classify the tasks based on the deadline misses tolerable in a K_i window:

Definition 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 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. *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 4. *A weakly-hard task τ_i with constraint $\binom{m_i}{K_i}$ is schedulable if, in any window of K_i consecutive invocations of the task, at least m_i deadlines are hits.*

Definition 5. *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.*

Definition 6. *A weakly-hard task τ_i with constraint $\left\langle {m_i \atop K_i} \right\rangle$ is schedulable if, in any window of K_i consecutive invocations of the task, τ_i meets m_i consecutive deadlines.*

Definition 7. *A weakly-hard task τ_i with constraint $\overline{\langle m_i \rangle}$ is schedulable if, the maximum number of consecutive deadlines misses is m_i.*

Utilization. The utilization of a task τ_i is defined as the fraction of processor time required by its execution:

$$U_i = \frac{C_i}{T_i}$$

Then, the utilization of the task set (also known as total utilization) is defined as the sum of all task utilizations:

$$U = \sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

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 missing its deadline is killed to remove load from the processor.

3.1 Problem Statement

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

4 Original Response Time Analysis

Our paper extends the well-established multi-core Response Time Analysis (RTA) [2] in Section 6. Therefore, we recall it in this section. The response time R_k of a task τ_k is defined as:

$$R_k \doteq C_k + I_k$$

Where I_k is the interference from other tasks and it is computed as follows:

$$I_k = \frac{1}{n_c} \sum_{i \neq k} I_{i,k}$$

Where n_c is the number of cores in the system and $I_{i,k}$ is the interference of a task τ_i over the task τ_k. For Task-Level Fixed Priority (TLFP), only the tasks with higher priority than τ_k interfere, therefore, I_k is reduced to:

$$I_k = \frac{1}{n_c} \sum_{i \in hp(k)} I_{i,k}$$

Where $hp(k)$ is the set of tasks indices with higher priority than τ_k.

An upper bound on the response time R_k^{ub} of a task τ_k can be computed by bounding the interference I_k. For computing $I_{i,k}$, we bound the workload $\hat{W}_i$ imposed from τ_i [2, Equation 4]:

$$\hat{W}_i(L) = N_i(L)C_i + \min(C_i, (L + D_i - C_i) \mod T_i)$$

Where $N_i(L)$ is the maximum number of jobs of τ_i that may execute within the time window of size L [2, Equation 3]:

$$N_i(L) = \left\lfloor \frac{L + D_i - C_i}{T_i} \right\rfloor$$

Hence, RTA iterates over the tasks in priority order to compute upper bounds on their response times.

$\hat{W}_i(L)$ calculates the workload imposed by τ_i considering the carry-in job, the body jobs and the carry-out job defined as follow (see Figure 2):

Definition 8. *A carry-in job is a job with a deadline within the interval of interest, but its release time is outside of it.*

Definition 9. *Body jobs are jobs with both their release time and deadline within the interval of interest.*

Definition 10. *A carry-out job is a job with a release time within the interval of interest but a deadline outside of it.*

To conservatively bound the workload of τ_i within the interval of interest L, we have to consider the execution of the carry-in job. Therefore, the first term of $\hat{W}_i(L)$ represents the workload due to the carry-in job and the body jobs while the second term bounds the workload due to the carry-out job.

The bound on $\hat{W}_i$ and $N_i(L)$ can be tightened by introducing the *slack* of τ_i. The slack of τ_i is calculated based on its response time as follows: $s_i = max(D_i - R_i^{ub}, 0)$, where R_i^{ub} is the upper bound on the response time of τ_i.

Considering s_i, $\hat{W}_i$ and $N_i(L)$ are computed as follow [2, Equation 8]:

$$\hat{W}_i = N_i(L)C_i + \min(C_i, (L + D_i - C_i - s_i) \mod T_i) \tag{1}$$

$$N_i(L) = \left\lfloor \frac{L + D_i - C_i - s_i}{T_i} \right\rfloor \tag{2}$$

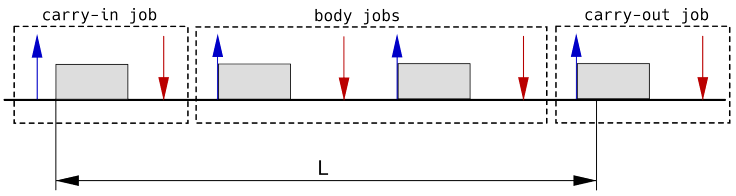

Fig. 2. Types of jobs within a time interval L as proposed in RTA [2]. Blue arrows represent task activation, while red ones represent deadlines. Gray boxes refer to job executions.

For TLFP, R_k^{ub} of the task τ_k can be found by the following fixed point equation, starting with $R_k^{ub} = C_k$, [2, Equation 7]:

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{n_c} \sum_{i \in hp(k)} \min(\hat{W}_i(R_k^{ub}), R_k^{ub} - C_k + 1) \right\rfloor \tag{3}$$

5 Global Scheduling for Weakly-hard Tasks

In this section, we present a new job-class-level algorithm for global scheduling of 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.

5.1 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 11 (w_i). *It represents the maximum number of consecutive deadline misses that may occur per deadline hit while guaranteeing that the total number of misses within any window of length K_i does not exceed m_i and it is calculated as follows:*

$$w_i = \max\left(\left\lfloor \frac{m_i}{K_i - m_i} \right\rfloor, 1\right) \tag{4}$$

Definition 12 (h_i). *It represents the number of deadline hits required per deadline miss and it is calculated as follows:*

$$h_i = \left\lceil \frac{K_i - m_i}{m_i} \right\rceil \tag{5}$$

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

Lemma 1. *If $m_i/K_i < 0.5$, then $w_i = 1$. If $m_i/K_i \geq 0.5$, 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, $h_i = 1$. □

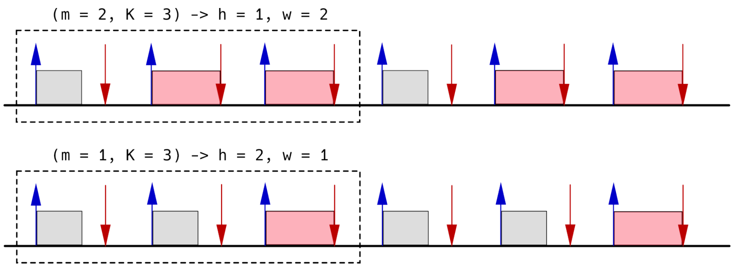
(m = 2, K = 3) -> h = 1, w = 2
(m = 1, K = 3) -> h = 2, w = 1

Fig. 3. Critical-sequence examples for a high-tolerance task (above) and a low-tolerance task (below). Gray boxes refer to execution finishing before the deadline, i.e., deadline hit. Boxes in pink refer to deadline miss.

Definition 13 (Critical sequence). *It is the sequence made up of h_i consecutive deadline hits followed by w_i consecutive deadline misses.*

The critical-sequence allows us to uniformly distribute the deadline misses in a window of K_i. Figure 3 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 deadline according to the critical-sequence. Therefore, it is vital to prove that a repeating cycle of instances of the critical-sequence do not violate the deadline-miss constraint given by $\overline{\binom{m_i}{K_i}}$. However, we know that the critical-sequence satisfies the $\overline{\left\langle {w_i \atop w_i+h_i} \right\rangle}$ constraint by definition. Next, we show that the critical-sequence also satisfies the constraint $\overline{\binom{w_i}{w_i+h_i}}$.

Lemma 2. *For low-tolerance and high-tolerance tasks, for which w_i and h_i are defined as in Definition 11 and Definition 12 respectively, the critical-sequence satisfies the constraint $\overline{\binom{w_i}{w_i+h_i}}$.*

Proof. For low-tolerance tasks $w_i = 1$, hence, the following holds: $\overline{\left\langle {1 \atop 1+h_i} \right\rangle} \equiv \overline{\binom{1}{1+h_i}}$. For high-tolerance tasks $h_i = 1$, therefore, we focus on the deadline hits instead of misses. From [1, Theorem 3], we have $\overline{\binom{w_i}{w_i+h_i}} \equiv \binom{h_i}{w_i+h_i}$. Hence, the following holds: $\left\langle {1 \atop 1+w_i} \right\rangle \equiv \binom{1}{1+w_i}$. □

The next step 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 $\overline{\binom{m_i}{K_i}}$, i.e. that the repeating cycles of the critical-sequence do not contains more deadline misses than $\overline{\binom{m_i}{K_i}}$.

Definition 14. *[1, Definition 10] 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 3. *The weakly-hard constraint $\overline{\binom{w_i}{w_i+h_i}}$ is harder than the constraint $\overline{\binom{m_i}{K_i}}$, formally, $\overline{\binom{w_i}{w_i+h_i}} \preccurlyeq \overline{\binom{m_i}{K_i}}$.*

Proof. Theorem 5 of [1] provides the condition that must be satisfied for one weakly-hard constraint to be harder than another. In this proof, we show that $\overline{\binom{w_i}{w_i+h_i}}$ satisfies the condition to be harder than $\overline{\binom{m_i}{K_i}}$. The detailed proof is in the appendix. □

In the examples of Figure 3, we can count the deadline misses within a $K_i = 3$ window. Shifting the window to the right, we observe that there is no more deadline misses than allowed by $\overline{\binom{m_i}{K_i}}$.

Theorem 1. *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 3, as $\overline{\binom{w_i}{w_i+h_i}} \preccurlyeq \overline{\binom{m_i}{K_i}}$. □

5.2 Priority Assignment to Job-classes

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 15. *A task τ_i has $\mathcal{JC}_i = K_i - m_i + 1$ job-classes. Here, every job-class is denoted 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.*

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). In case of two or more tasks have the same deadline, the one with 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). Table 3 shows an example of three different tasks, their corresponding q range and their job-class priorities after running the priority assignment algorithm.

Table 3. Example of three tasks, their q ranges and their priorities. Note that jobs of τ_1 may get the highest priority among all possible priorities (9) and the lowest possible priority (1).

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

5.3 Scheduling Routine: Assigning Priorities to Released Jobs

Every time a job is released, the scheduler assigns it to a job-class based on the previous deadline misses/hits. For selecting to which particular job-class a job should be assigned, we define job-level:

Definition 16. *The job-level jl_i is a variable of a task τ_i which is used to select the job-class $\mathcal{JC}_i^q$ of the released job. The value of q is selected according to $q = max(0, jl_i)$. The initial value of jl_i is $-(h_i - 1)$ and every time a job meets its deadline, jl_i is increased by one until $K_i - m_i$. When w_i deadline misses happened, the value of jl_i is restored to $-(h_i - 1)$.*

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 1).

Figure 4 shows the transitions between job-classes for low-tolerance and high-tolerance tasks.

Algorithm 1: Priority assignment to job-classes.

```
1  Input: taskset $\mathcal{T}$
2  $sort\_tasks\_ascending\_deadline(\mathcal{T})$
3  for $\tau_i \in \mathcal{T}$ do
4      $\mathcal{JC}_i \leftarrow K_i - m_i + 1$
5  $\mathcal{JC} \leftarrow \sum_{\forall \tau_i \in \mathcal{T}} \mathcal{JC}_i$
6  $prio \leftarrow \mathcal{JC}$
7  $\mathcal{JC}^{max} \leftarrow max\{\mathcal{JC}_i | \forall \tau_i \in \mathcal{T}\}$
8  for $q \leftarrow 0; q < \mathcal{JC}^{max}; q \leftarrow q + 1$ do
9      for $\tau_i \in \mathcal{T}$ do
10         if $q < \mathcal{JC}_i$ then
11             $\mathcal{JC}_i^q \leftarrow prio$
12             $prio \leftarrow prio - 1$
```

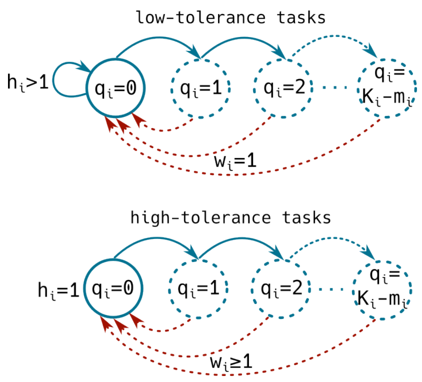

Fig. 4. Job-classes transitions for low-tolerance and high-tolerance tasks. Solid circles represent the highest priority. Hence, jobs assigned to the priority represented by the solid circle are guaranteed to meet their deadlines. Solid transition lines indicate transitions after deadline hits. Jobs of low-tolerance tasks remain h_i times in $q_i = 0$ before changing to the next level. Jobs of high-tolerance tasks change of level after the first deadline hit. Red dashed transition lines indicate possible transition after w_i misses.

6 Response Time Analysis Extension for Weakly-hard Real-time Tasks

This section presents a schedulability analysis for the proposed job-class-level scheduling algorithm described in the previous section. The analysis is an extension of RTA for weakly-hard real-time tasks scheduled by our algorithm. As mentioned in Section 4, RTA provides a sufficient condition for schedulability by proving that the response time of a task is not longer than its deadline. The response time of a task is prolonged as much as tasks with higher priorities interfere with the execution of the analyzed task. The extension of RTA presented here bounds the interference over lower priority tasks by taking into account only jobs belonging to $\mathcal{JC}_i^{q=0}$. Remember that jobs in $\mathcal{JC}_i^{q=0}$ have the highest assignable priority to their tasks and are selected based on the minimum consecutive deadline hits (h_i) from the critical-sequence. Moreover, considering only jobs in $\mathcal{JC}_i^{q=0}$ reduces the interference to lower priority jobs.

Lemma 4. *For a task τ_i, if the jobs belonging to $\mathcal{JC}_i^{q=0}$ meet their deadlines, then τ_i meets its constraint $\overline{\binom{w_i}{w_i+h_i}}$.*

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 4 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. □

Consequently, this section considers only the jobs in job-classes $\mathcal{JC}_i^{q=0}$. However, note that the other job-classes are used as a part of the relaxation mechanism which gives the opportunity to other tasks be executed with a higher priority temporarily. This relaxation mechanism fairly shares the priorities among the jobs to enable all tasks meeting their $\overline{\binom{w_i}{w_i+h_i}}$ constraints.

This section starts by proving that it is possible to extend RTA for the proposed scheduling algorithm. Finally, the schedulability condition for a task set is defined.

6.1 Proving RTA Extension

We prove the extension of RTA for our algorithm by showing how to bound the interference over jobs in job-classes $\mathcal{JC}_i^{q=0}$. First, we show that only the interference between the highest priority job-classes have to be considered. Then, we evaluate the interference based on the critical-sequence. This is done separately for low-tolerance and high-tolerance tasks because of the differences in the critical-sequences (see Definition 13).

With the following lemma, we show that only the response time of jobs belonging to $\mathcal{JC}_i^{q=0}$ are of interest.

Lemma 5. *A job of a task τ_k in $\mathcal{JC}_k^{q=0}$ suffers interference only from the jobs of a task τ_i in $\mathcal{JC}_i^{q=0}$, if the priority of $\mathcal{JC}_i^{q=0}$ is higher than the priority of $\mathcal{JC}_k^{q=0}$.*

Proof. The 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 belong to job-class $\mathcal{JC}_k^{q=0}$ suffer 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 $\mathcal{JC}_k^{q=0}$. □

To bound the interference induced by τ_i on τ_k, we must bound the maximum number of jobs in $\mathcal{JC}_i^{q=0}$. In the worst-case, jobs of τ_i are assigned to $\mathcal{JC}_i^{q=0}$ h_i times every $w_i + h_i$ jobs. For a high-tolerance task τ_i, the jobs in $\mathcal{JC}_i^{q=0}$ are w_i apart. This allows to consider the workload coming from such a task as the same workload produced by a task with a longer inter-arrival time that is equal to $(w_i + 1)T_i$. Formally writing this:

Lemma 6. *The workload of a high-tolerance task τ_i with constraint $\overline{\binom{m_i}{K_i}}$ is calculated as 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. □

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 the inter-arrival time. However, we need to exclude the jobs that do not belong to $\mathcal{JC}^{q=0}$ from the workload bound. Hence, we update the workload bound defined in Equation (1) as follows. In the first term, the update subtracts the number of jobs with lower priority from N_i over the interval of interest L. For updating the workload coming from the carry-out job, the second term of $\hat{W}_i(L)$ is made equal to zero when a carry-out job has a lower priority than $\mathcal{JC}_i^{q=0}$.

For updating the first term, the number of lower priority jobs is defined by the following lemma:

Lemma 7. *The minimum number of jobs of a low-tolerance task τ_i that have lower priorities than $\mathcal{JC}_i^{q=0}$ during the interval L is given by:*

$$O_i(L) = \left\lfloor \frac{L + D_i - C_i - s_i}{T_i(h_i + 1)} \right\rfloor \tag{6}$$

Proof. The critical-sequence for a low-tolerance task allows one deadline miss after h_i deadline hits. This means that the maximum distance between two jobs of τ_i with a priority less than $\mathcal{JC}_i^{q=0}$ is $T_i(h_i + 1)$. □

Lemma 8. *According to the critical-sequence, only $N_i(L) - O_i(L)$ jobs contribute with an interference of C_i to lower priorities classes $\mathcal{JC}_k^{q=0}$ over the interval L.*

Proof. $N_i(L)$ jobs may contribute with a interference of C_i. However, the $(h_i + 1)T_i$ job does not contribute because, according to the critical-sequence, it is assigned with a lower priority than $\mathcal{JC}_k^{q=0}$. Hence, only $N_i(L) - O_i(L)$ jobs contribute with an interference of C_i. □

For removing the contribution of carry-out jobs, we introduce a variable that enables the second term in $\hat{W}_i(L)$ only if the carry-out jobs are in $\mathcal{JC}_i^{q=0}$.

Definition 17. *The variable a_i equals one by default and zero when carry-out jobs are not in $\mathcal{JC}_i^{q=0}$.*

$$a_i = 1 - \left\lfloor \frac{N_i(L) \mod (h_i + 1)}{h_i} \right\rfloor \tag{7}$$

The value of $N_i(L) \mod (h_i + 1)$ gives the number of jobs of the current critical-sequence. When this value is h_i, the next carry-out job belongs to a lower priority job-class. Furthermore, the result of the floor is one when: $N_i(L) \mod (h_i + 1) = h_i$. It follows that the value of a_i is zero when the carry-out job belongs to a different job-class than $\mathcal{JC}_i^{q=0}$.

Then, we update the workload bound equation as follows:

Lemma 9. *The workload bound function of a low-tolerance task τ_i is updated for removing non-interfering jobs as follow:*

$$\hat{W}_i(L) = (N_i(L) - O_i(L))C_i + a_i.\min(C_i, (L + D_i - C_i - s_i) \mod T_i) \tag{8}$$

Proof. The proof is given by Lemma 8 and Definition 17. □

Algorithm 2 shows the function for calculating $\hat{W}_i$ which is used in the fixed point iteration of Equation (3).

Lemma 10. *R_k^{ub} as given in (3) is a safe upper bound on the response time of jobs in $\mathcal{JC}_k^{q=0}$, where $\hat{W}_i$ is bounded as in (8).*

Proof. From Algorithm 2, the workload contribution of high-tolerance tasks uses the approach proven in Lemma 6 and for low-tolerance tasks, it uses the approach proven in Lemma 9. □

Algorithm 2: Workload computation.

```
1  Input: Length L, task $\tau_i$
2  Output: Workload $\hat{W}_i$
3  $s_i \leftarrow max(D_i - R_i^{ub}, 0)$ // Slack
4  if $m_i/K_i >= 0.5$ then
5      // High-tolerance task
6      $T_i \leftarrow (w_i + 1)T_i$
7      $O_i \leftarrow 0$
8      $a_i \leftarrow 1$
9  else
10     // Low-tolerance task
11     $O_i \leftarrow \left\lfloor \frac{L+D_i-C_i-s_i}{T_i(h_i+1)} \right\rfloor$ // Equation (6)
12     $a_i \leftarrow 1 - \left\lfloor \frac{N_i \mod (h_i+1)}{h_i} \right\rfloor$ // Equation (7)
13 $N_i \leftarrow \left\lfloor \frac{L+D_i-C_i-s_i}{T_i} \right\rfloor$ // Equation (2)
14 $\hat{W}_i \leftarrow (N_i - O_i)C_i + a_i.\min(C_i, (L + D_i - C_i - s_i) \mod T_i)$ // Equation (8)
15 return $\hat{W}_i$
```

6.2 Schedulability Analysis

We introduce the following theorem for schedulability of a weakly-hard real-time task τ_i scheduled by our algorithm:

THEOREM 2. *For a task τ_i with the constraint $\overline{\binom{m_i}{K_i}}$, meeting the deadlines of the jobs belonging to $\mathcal{JC}_k^{q=0}$ is a sufficient condition for the schedulability of τ_i .*

PROOF. It follows directly from Theorem 1 and Lemma 4. □

Then, the schedulability of a task set is defined as follows:

COROLLARY 1. *A task set is schedulable by our scheduling algorithm if for every task, the sufficient condition given in Theorem 2 is satisfied.*

7 Evaluation

Our global scheduling depends on the transformation of the weakly-hard constraint $\overline{\binom{m_i}{K_i}}$ to the $\overline{\binom{w_i}{w_i+h_i}}$ constraint, which has a smaller window. In this section, we study first the limitations introduced by adopting a harder constraint than $\overline{\binom{m_i}{K_i}}$.

We also evaluate our global scheduling for weakly-hard real-time tasks in this section. To the best of our knowledge, there is no other global scheduling analysis for weakly-hard real-time tasks. Furthermore, since we are interested in RTEMS and the available global schedulers are EDF and RM, we compare the proposed scheduling algorithm against the results of RTA (Section 4) for RM and EDF. 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. Additionally, we ran our experiments for different values of K_i and measured the computation times for different number of tasks.

Furthermore, in order to analyse the scalability of our approach, we also compare our analysis (here labeled as RTA WH) against the Integer Linear Programming (ILP) approach proposed in [17] and the Job-Class Level (JCL) proposed in [5]. However, note that the comparison of a multi-core scheduling analysis against another for single-core is not fair. In single-core scheduling analysis, the condition for schedulabilty (known as critical instant) considers that all higher priority tasks are released simultaneously with the analyzed task. However, the critical instant cannot be used in multi-core. Moreover, the sufficient schedulability condition used in multi-core is pessimistic for single-core. Therefore, the schedulability ratio results of comparing against the single-core approaches are only for illustration.

Finally, at the end of this section, we show the execution time distribution for assigning priorities to released jobs. This last experiment runs on the same hardware platform used in CALLISTO.

7.1 Transformation Cost

For analyzing the limitation introduced by transforming $\overline{\binom{m_i}{K_i}}$ to $\overline{\binom{w_i}{w_i+h_i}}$, where $\overline{\binom{w_i}{w_i+h_i}}$ is harder than $\overline{\binom{m_i}{K_i}}$, we count the possible deadline sequences which satisfy both constraints. Algorithm 3 counts these possible solutions based on a given $\overline{\binom{m_i}{K_i}}$. It starts by calculating h_i and w_i for a given $\overline{\binom{m_i}{K_i}}$ (Lines 3 and 4). Then, it creates a table with K_i columns (Line 5). Every row of the table represents a sequence of deadlines, in which 1 is a deadline hit and 0 is a deadline miss, see Table 4. In Lines 7 and 8, the solutions for $\overline{\binom{m_i}{K_i}}$ and $\overline{\binom{w_i}{w_i+h_i}}$ are counted from the rows.

Algorithm 3: Algorithm for counting possible solutions.

```
1 Input: constraint $\overline{\binom{m_i}{K_i}}$
2 Output: relation between solutions for the harder and the original constraints
3 $w_i \leftarrow \max\left(\left\lfloor \frac{m_i}{K_i - m_i} \right\rfloor, 1\right)$ // Definition 11
4 $h_i \leftarrow \left\lceil \frac{K_i - m_i}{m_i} \right\rceil$ // Definition 12
5 $table \leftarrow create\_table(K_i)$
6 $rows \leftarrow extract\_rows(table)$
7 $solutions \leftarrow count\_solutions(\overline{\binom{m_i}{K_i}}, rows)$
8 $harder\_solutions \leftarrow count\_solutions(\overline{\binom{w_i}{w_i+h_i}}, rows)$
9 return $harder\_solutions/solutions$
```

Results for different values of $\overline{\binom{m_i}{K_i}}$ are shown in Table 5. For low-tolerance tasks, the omitted deadline sequences can be very high when m_i/K_i is near to 0.5. On the other hand, there are less omitted sequences for high-tolerance. For both kind of tasks, the number of omitted sequences increases when considering a constraint which is multiple of another, e.g. $\overline{\binom{m_i}{K_i}} = \overline{\binom{8}{20}}$ and $\overline{\binom{m_i}{K_i}} =$

Table 4. Table created for counting deadline sequences. Every row represents a possible deadline sequence.

d_0	d_1	$\cdots$	d_{K_i-2}	d_{K_i-1}
0	0	$\cdots$	0	0
0	0	$\cdots$	0	1
0	0	$\cdots$	1	0
0	0	$\cdots$	1	1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
1	1	$\cdots$	1	0
1	1	$\cdots$	1	1

Table 5. 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

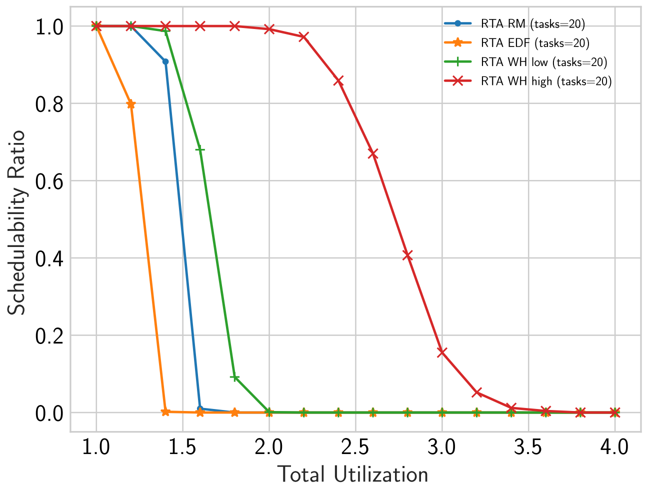

Fig. 5. Schedulability ratio for 2 cores.

$\overline{\binom{2}{5}}$. This happens because specific deadline sequences with consecutive deadline misses are not accounted when the harder constraint is used. Despite this limitation, our experiments show better results than traditional hard real-time scheduling analysis. Furthermore, the results obtained here are theoretical since not all the uncounted deadline sequences will also mean schedulable solutions.

7.2 Scheduling Analysis Setup

Task sets are generated for a list of desired total utilization values using UUnifast. For every total utilization value, 1000 sets are generated. The utilization of the generated tasks is not higher than one. The inter-arrival times are randomly generated within the interval $[10, 100)$, with a uniform probability distribution. The generated tasks have implicit deadlines, i.e. their deadlines are equal to their inter-arrival time ($D_i = T_i$). Furthermore, all scheduling analysis were developed in Python 3 and executed on parallel for each total utilization value.

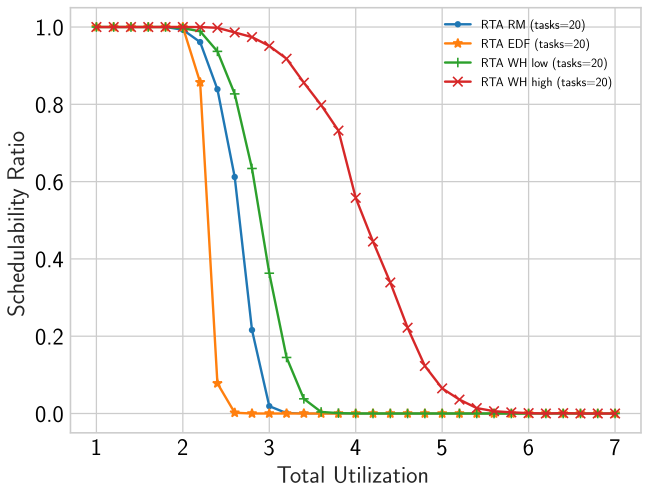

Fig. 6. Schedulability ratio for 4 cores.

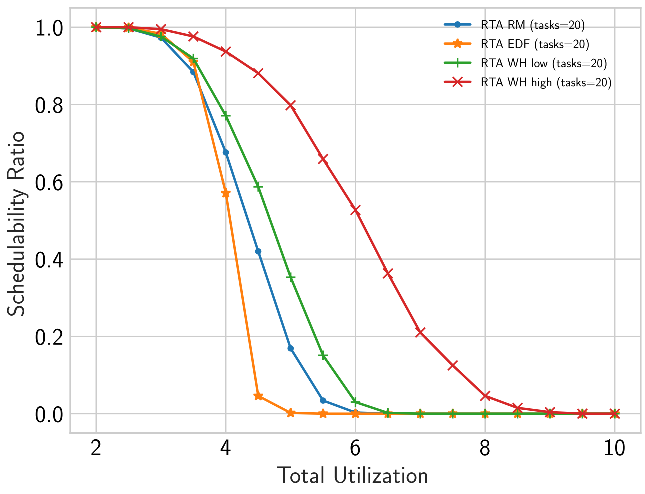

Fig. 7. Schedulability ratio for 8 cores.

For every generated task set, the schedulability ratio is calculated and the computation time is measured. In case of the global scheduling comparison, RTA verifies its schedulability for RM, EDF and the extension of RTA for weakly-hard real-time tasks considering two scenarios. In the first scenario, all tasks are low-tolerance tasks and in the second, all tasks are high-tolerance tasks. 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.

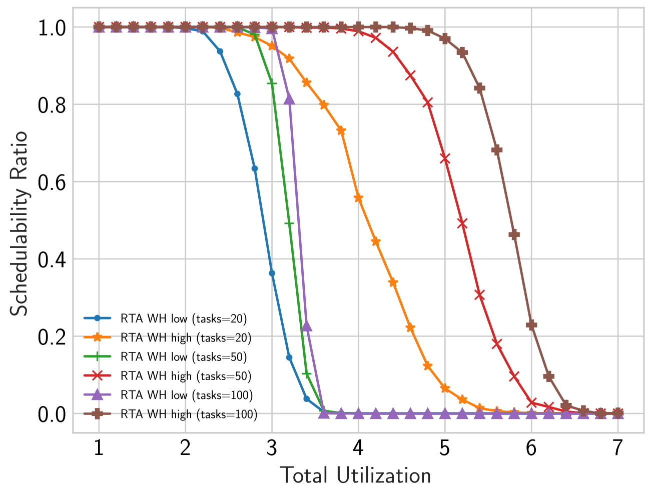

Fig. 8. Schedulability ratio for 20, 50 and 100 tasks (4 cores).

For the comparison against the other weakly-hard scheduling approaches, the value of m_i is either 1 or $K_i - 1$ because the ILP analysis [17] only permits the same weakly-hard constraint for the tasks in the set. Additionally, only $K_i = 5$ was used for ILP due to the scalability issues [14]. Nevertheless, further experiments compare against JCL using $K_i = 10$ and $K_i = 20$.

Finally, 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 to run the experiments.

7.3 Scheduling Analysis Experiments

Figure 5, Figure 6, and Figure 7 show the schedulability ratio using a set of 20 tasks for 2, 4 and 8 cores, respectively. The plots for EDF show a worst schedulability ratio in comparison with RM. This is due to RTA calculates a higher interference for EDF than for RM (remember that in RM only tasks with higher priority are considered). Furthermore, it is observed that the schedulability ratio for the weakly-hard tasks is much better than for RM. Additionally, high-tolerance tasks seem to be schedulable even after the practical limit (U = number of cores). This is explained by Lemma 6, which allows us to consider the workload coming from the high-tolerance tasks as it were coming from a task with longer inter-arrival time, reducing the utilization of the task. This also explains why this behavior is not seen for low-tolerance tasks.

Figure 8 shows the schedulability ratio for 20, 50 and 100 tasks in the set when scheduled on 4 cores. Here, it is observed that sets with more tasks have a better schedulability. The reason is that having more tasks, while keeping the same total utilization, makes the tasks more lightweight which reduces the interference. On the other hand, the computation time is increased with the number of tasks. This is observed in Figure 9 which shows the computation time distribution using box plots (the blue curve represents the average duration for RM). Also, note that computation time decreases with the schedulability ratio because the scheduling analysis ends when the first no schedulable task is found.

Figure 10 shows the schedulability ratio of tasks scheduled on 4 cores when K_i is 5, 50 and 500. There are no significant changes in the schedulability ratio when the value of K_i is changed. Also,

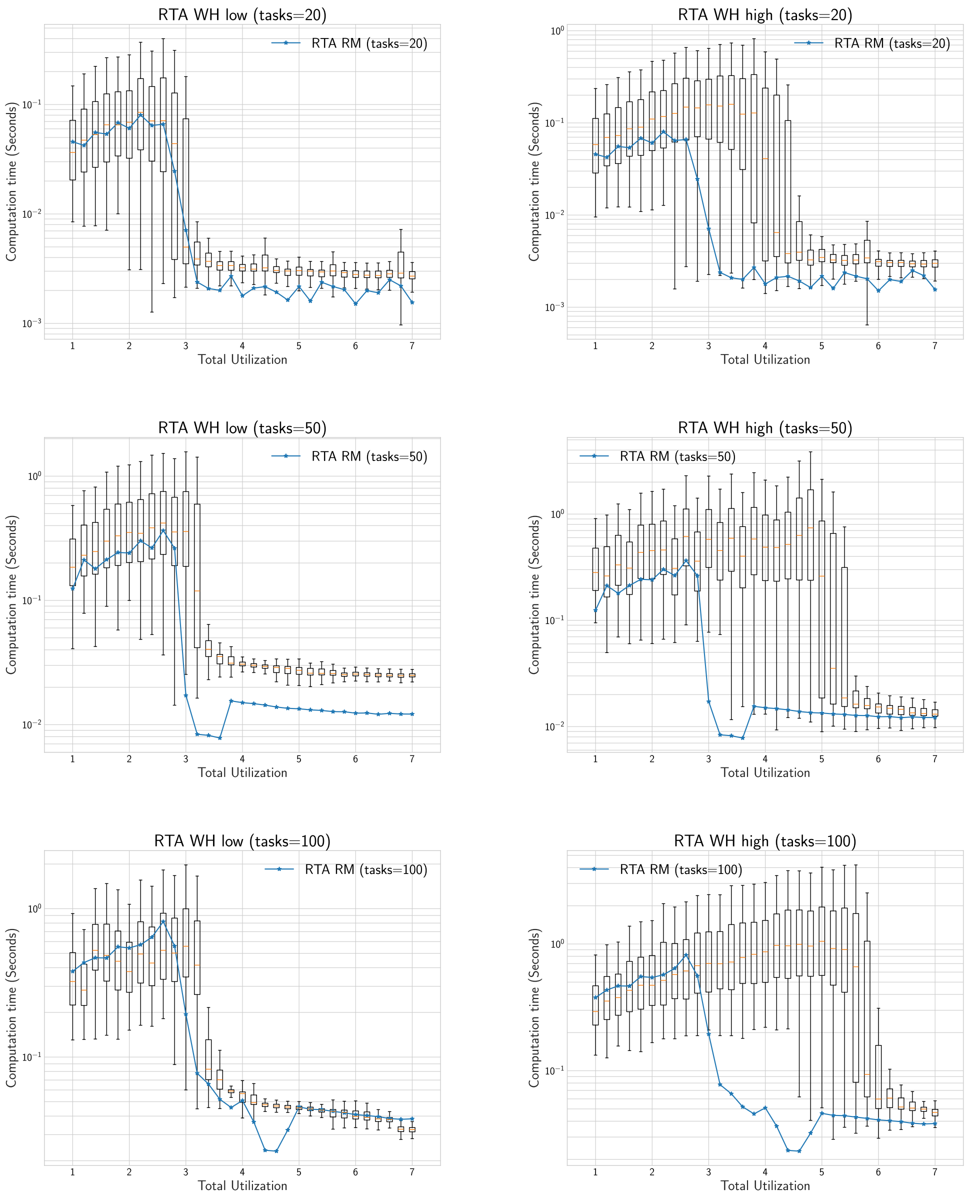

Fig. 9. Computation time for 20, 50 and 100 tasks (4 cores). Box plots show the distribution of computation times and the blue curve indicates the average for RM. The first column shows computations times for low-tolerance tasks and the second column for high-tolerance tasks. Every row shows results for different amount of tasks in the set.

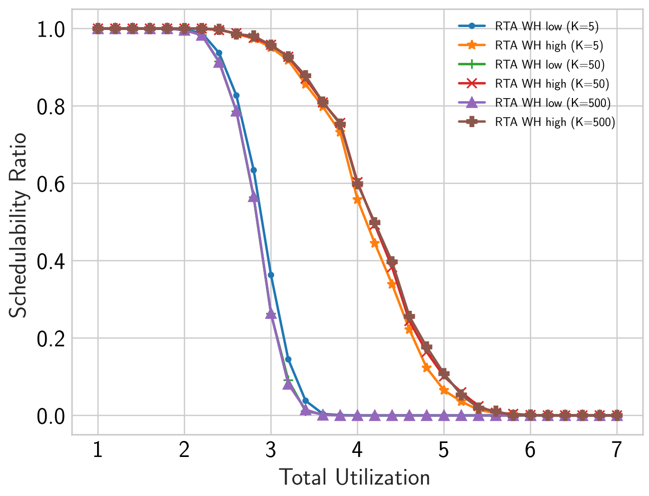

Fig. 10. Schedulability ratio for K_i equals to 5, 50 and 500 (4 cores).

we highlight that the theoretical results shown in Table 5 are pessimistic in comparison with the results of this experiment. In terms of the computation time, no significant changes were observed, see Figure 11.

7.4 Comparison Against Single-Core Scheduling Approaches

Figure 12 shows the schedulability ratio comparison between ILP, JCL and RTA WH using task sets with 30 tasks and the same $K = 5$ for all tasks (due to the scalability of the ILP approach for larger K). For experiments with $m_i = 1$, JCL is the best of the three while ILP is the worst of them. The bad results of the ILP approach are explained by remembering that the ILP approach works at task-level while the other approaches work at job-level. In case of RTA WH, there is pessimism introduced by using a multi-core scheduling analysis for single-core (as mentioned before).

When $K_i = 10$ and $K_i = 20$, Figure 13 and Figure 14 show that the schedulability ratio is also better for JCL than for RTA WH. Again, this is explained similar as before, i.e. using RTA for single-core introduces pessimism. However, the schedulability ratio curves for $m_i = K_i - 1$ are similar for JCL and RTA WH until a total utilization equals to 1.

Figure 15 shows the computation times for ILP, JCL and RTA WHA. For ILP, the computation time increases with the utilization. However, for JCL and RTA WH, the computation time decreases together with the schedulability ratio because unschedulable sets are found faster. In addition, the JCL approach is faster than the others when $K_i = 5$. Moreover, JCL is still faster when $K_i = 10$ and $K_i = 20$ but only for high-tolerance tasks. The reason is that the sufficient condition used in RTA WH takes longer than the analysis used in JCL which considers the critical instant. Nevertheless, for low-tolerance tasks, while increasing K_i to 10, the computation times of JCL are in the same range than those for RTA WH (see Figure 16). Furthermore, when $K_i = 20$, RTA WH becomes faster than JCL for low-tolerance tasks due to the iteration of JCL over a recheabilty tree, see Figure 17. In fact, RTA WH does not variate too much while changing the value of K_i. This was also observed in Figure 11.

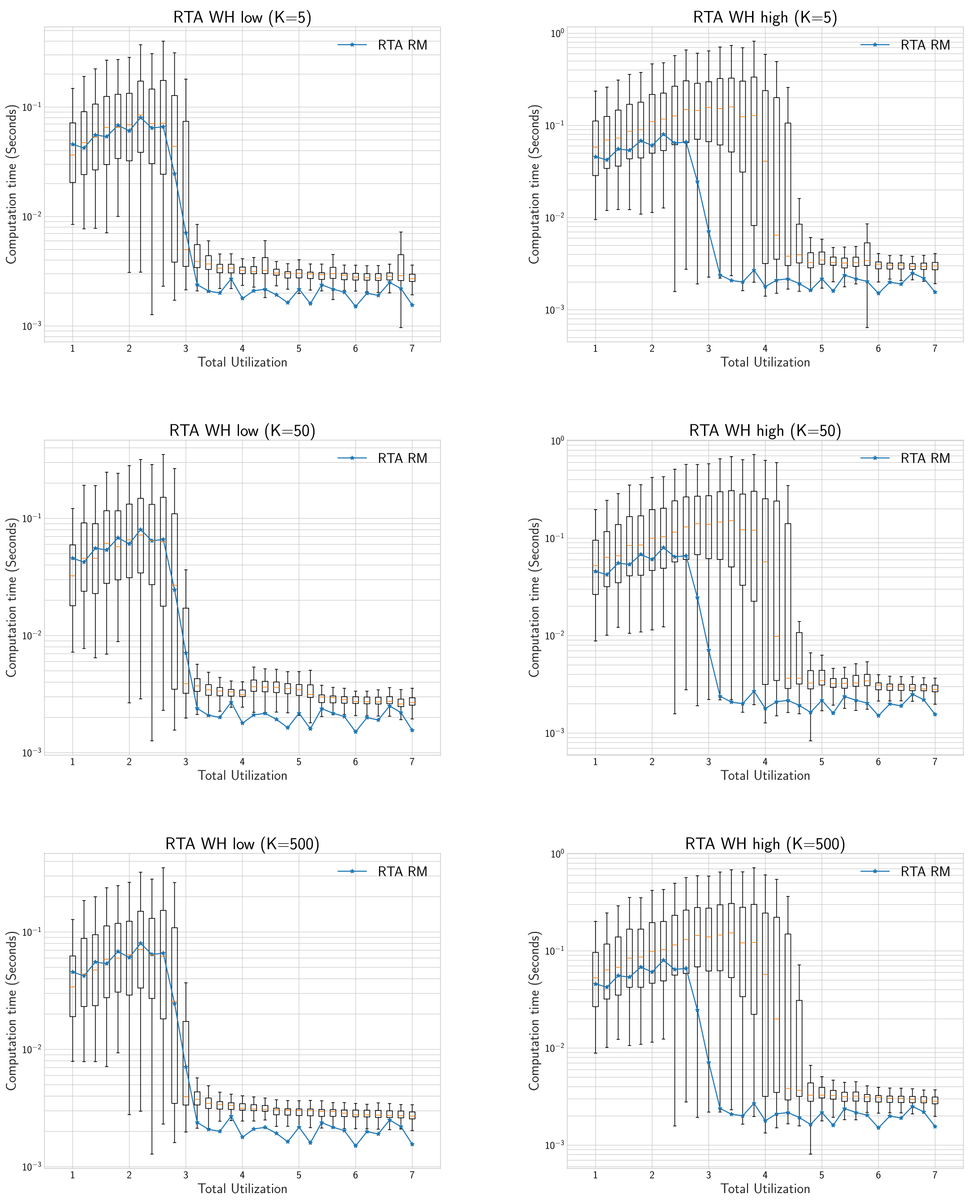

Fig. 11. Computation time for K_i equals to 5, 50 and 500 (4 cores, 20 tasks in the set). Box plots show the distribution of computation times and the blue curve indicates the average for RM. The first column shows computations times for low-tolerance tasks and the second column for high-tolerance tasks. Every row shows results for different values of K_i.

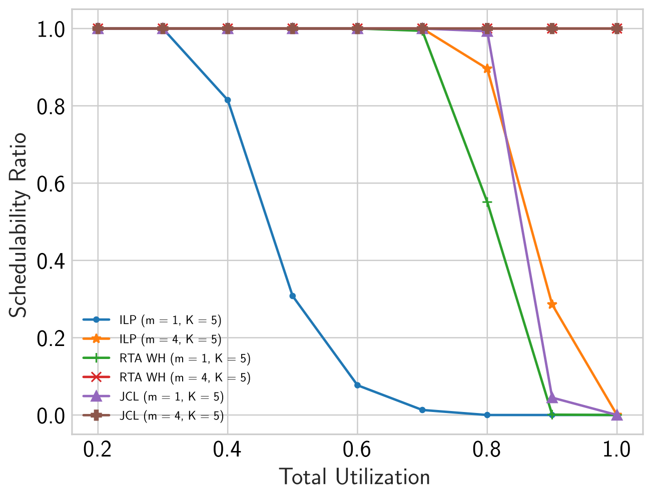

Fig. 12. Schedulability ratio for ILP, JCL and RTA WH (K = 5).

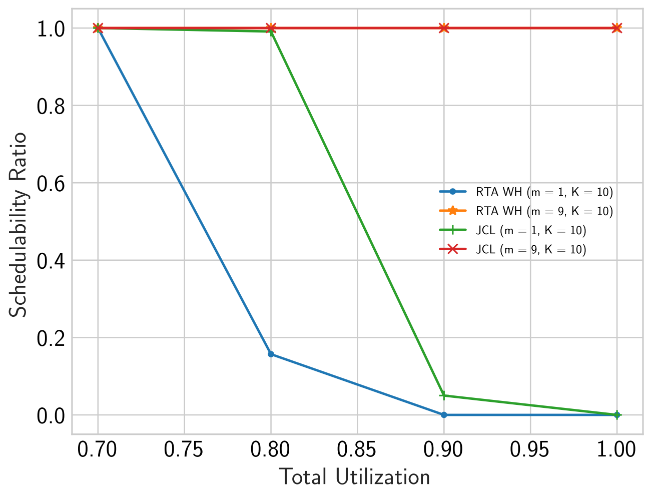

Fig. 13. Schedulability ratio for JCL and RTA WH (K = 10).

7.5 Priority Assignment Overhead

In order to know the scheduling overhead due to the priority assignment algorithm defined in Definition 16, we have conducted an experiment for measuring its execution time considering only the transition among job classes in isolation, i.e. no OS system overhead is considered. This experiment was executed on the same processor architecture used in the on-board computers of CALLISTO, i.e. Intel Atom Quad-Core with core frequency of 1.91GHz. The real-time operating system used in this experiment is RTEMS 5.1 and the code was compiled with optimization level 2.

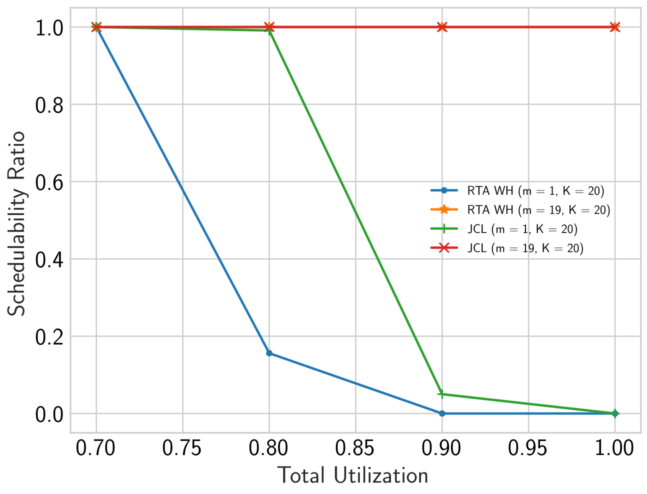

Fig. 14. Schedulability ratio for JCL and RTA WH (K = 20).

Figure 18 shows the execution time distribution for executing the priority assignment in case of both low and high-tolerance tasks. The algorithm execution was repeated 1000 times for every type of task. Results show an overhead below 60 nanoseconds which is negligible overhead comparing to the execution time of the tasks. Low-tolerance tasks show a higher median than high-tolerance tasks. This behavior is related with the fact that high-tolerance tasks tolerate more deadline misses before going back to the $\mathcal{JC}_i^0$, making some part of the algorithm be executed more often.

8 Conclusion

In centralized embedded system architectures, multi-core processors are utilized to consolidate multiple tasks leveraging the high computational power and the low power-consumption of multi-core platforms. In real-time systems, if few deadline misses are tolerable, leveraging the weakly-hard model can reduce the over-provisioning, hence, consolidating more real-time tasks on to the multi-core platform. This paper proposed a job-class based global scheduling for weakly-hard real-time tasks, appended with a schedulability test to compute the needed guarantees. 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 proposed analysis is schedulable where through all the synthetic test cases, the computation time for the proposed analysis does not exceed 1.6 seconds for 100 tasks and 4 cores. Also, results illustrates that the improvement on the schedulability ratio is up to 40% over the global Rate Monotonic (RM) scheduling and up to 60% over the global EDF scheduling. Furthermore, our future work will contemplate a reduction in the limitations for low-tolerance tasks, the exploration of the priority assignment in the context of job-class-level scheduling and the interference between tasks due to shared resources.

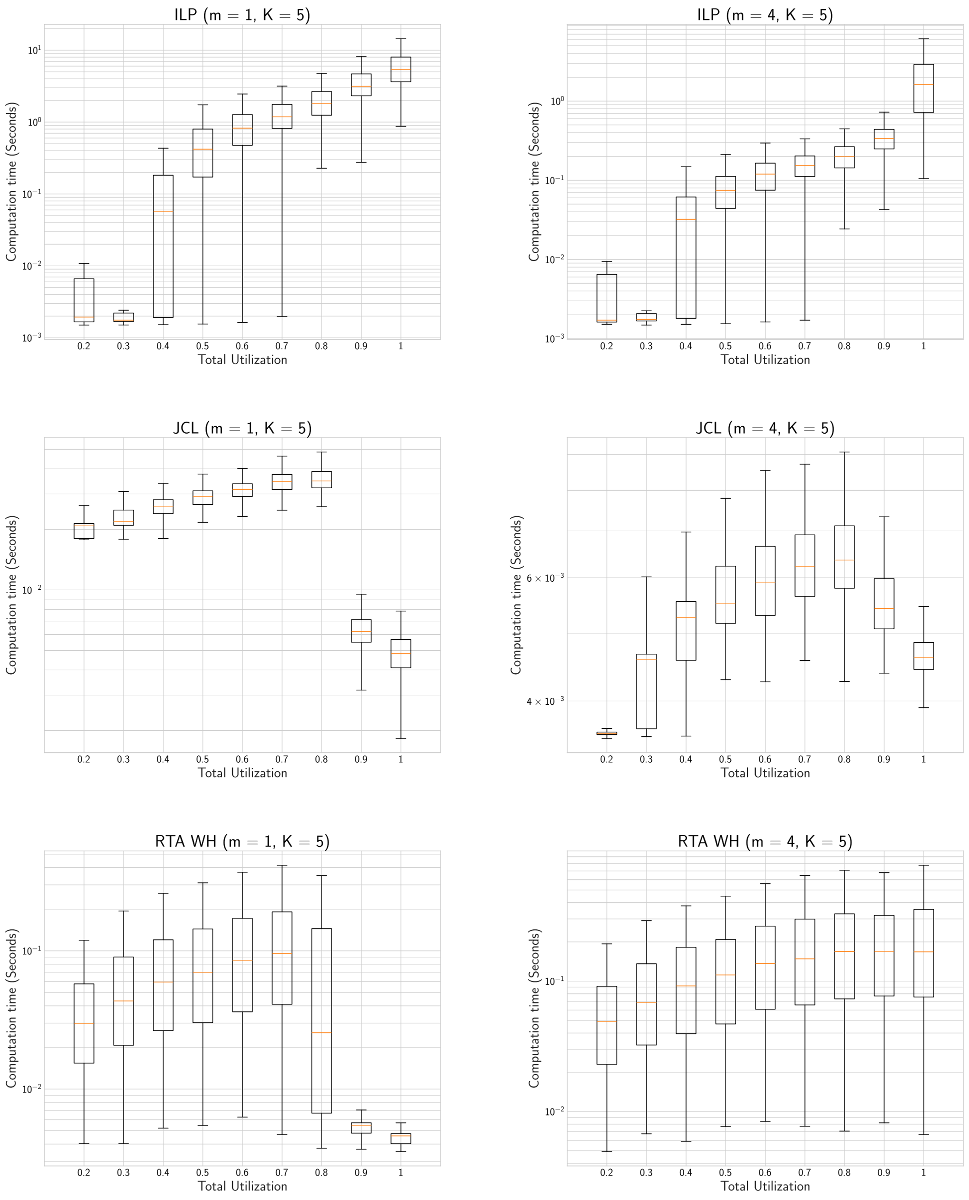

Fig. 15. Computation time for ILP, JCL and RTA WH (K = 5). Box plots show the distribution of computation times. The first column shows computations times for ($m_i = 1, K_i = 5$) tasks and the second column for ($m_i = 4, K_i = 5$) tasks. The first row shows results for ILP, the second for JCL and the third one for RTA WH.

References

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

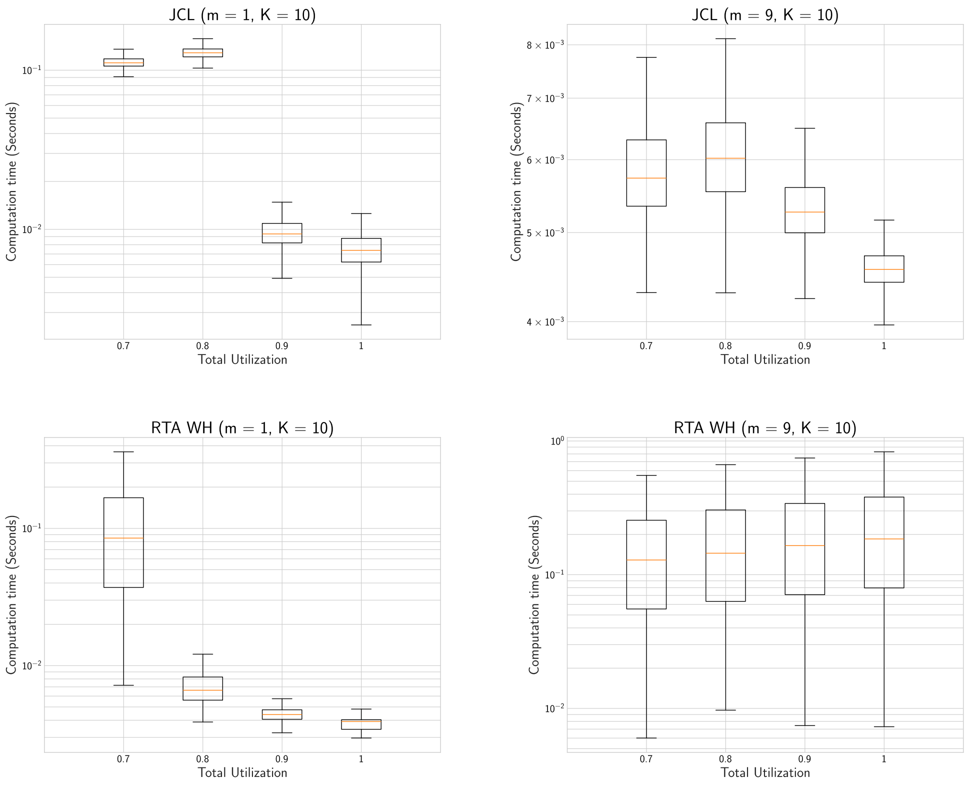

Fig. 16. Computation time for JCL and RTA WH (K = 10). Box plots show the distribution of computation times. The first column shows computations times for $(m_i = 1, K_i = 10)$ tasks and the second column for $(m_i = 9, K_i = 10)$ tasks. The first row shows results for JCL and the second for RTA WH.

[2] Marko Bertogna and Michele Cirinei. 2007. Response-Time Analysis for Globally Scheduled Symmetric Multiprocessor Platforms. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE Computer Society, USA, 149–160. doi:10.1109/RTSS.2007.31

[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)*. IEEE Computer Society, USA, 241–253. doi: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. doi:10.1145/3356865

[7] 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.

[8] 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) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 76)*, Marko Bertogna (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 17:1–17:22. doi:10.4230/LIPIcs.ECRTS.2017.17

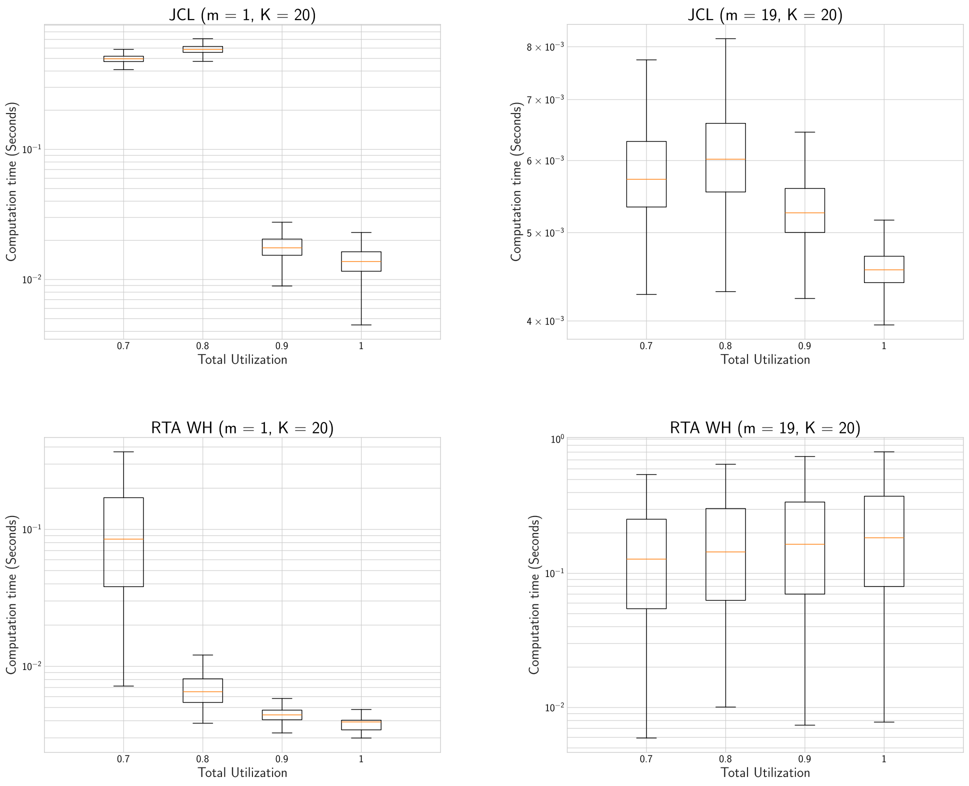

Fig. 17. Computation time for JCL and RTA WH (K = 20). Box plots show the distribution of computation times. The first column shows computations times for $(m_i = 1, K_i = 20)$ tasks and the second column for $(m_i = 19, K_i = 20)$ tasks. The first row shows results for JCL and the second for RTA WH.

[9] Michel Illig, Shinji Ishimoto, and Etienne Dumont. 2022. CALLISTO, a demonstrator for reusable launchers. In *9th European Conference for Aeronautics and Space Sciences (EUCASS)*.

[10] Y. Kong and H. Cho. 2011. Guaranteed Scheduling for (m,k)-firm Deadline-Constrained Real-Time Tasks on Multiprocessors. In *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*. IEEE Computer Society, USA, 18–23.

[11] Sven Krummen, Jean Desmariaux, Yasuhiro Saito, Marcelo Boldt, Lale Evrim Briese, Nathalie Cesco, Christophe Chavagnac, Elisa CLIQUET-MORENO, Etienne Dumont, Tobias Ecker, et al. 2021. Towards a reusable first stage demonstrator: Callisto-technical progresses & challenges. In *Proceedings of the International Astronautical Congress, IAC*.

[12] Hengyi Liang, Zhilu Wang, Ruochen Jiao, and Qi Zhu. 2020. Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees. In *Proceedings of the 39th International Conference on Computer-Aided Design* (Virtual Event, USA) *(ICCAD '20)*. Association for Computing Machinery, New York, NY, USA, Article 101, 9 pages. doi:10.1145/3400302.3415717

[13] 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) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 165)*, Marcus Völp (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:24. doi:10.4230/LIPIcs.ECRTS.2020.21

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

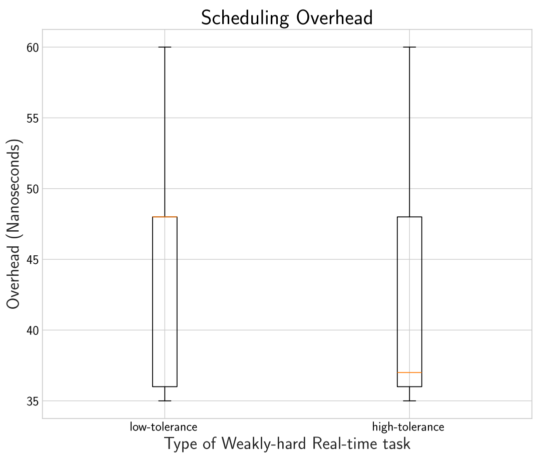

Fig. 18. Scheduling overhead distribution due to job classes transitioning (1000 samples).

[15] 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) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 106)*, Sebastian Altmeyer (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 10:1–10:22. doi:10.4230/LIPIcs.ECRTS.2018.10

[16] René Schwarz, Marco Solari, Bronislovas Razgus, Michael Dumke, Markus Markgraf, Matias Bestard Körner, Dennis Pfau, Martin Reigenborn, Benjamin Braun, and Jan Sommer. 2019. Preliminary Design of the Hybrid Navigation System (HNS) for the CALLISTO RLV Demonstrator. In *8th European Conference for Aeronautics and Space Sciences (EUCASS)*. doi:10.13009/EUCASS2019-735

[17] 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. doi:10.1145/3126497

[18] Tong Wu and Shiyao Jin. 2008. Weakly hard real-time scheduling algorithm for multimedia embedded system on multiprocessor platform. In *2008 First IEEE International Conference on Ubi-Media Computing*. IEEE Computer Society, USA, 320–325. doi:10.1109/UMEDIA.2008.4570910

[19] 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. doi:10.4230/LIPIcs.ECRTS.2021.15

[20] 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)*. IEEE Computer Society, USA, 228–240. doi:10.1109/RTAS54340.2022.00026

[21] 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. doi:10.1109/LCSYS.2022.3179960

[22] 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 *Proceedings of the 2015 27th Euromicro Conference on Real-Time Systems (ECRTS '15)*. IEEE Computer Society, USA, 247–256. doi:10.1109/ECRTS.2015.29

A Lemma 3 Proof

Theorem 5 of [1] states that a weakly-hard constraint $\binom{a}{b}$ is harder than (denoted as $\preccurlyeq$) other constraint $\binom{p}{q}$ if:

$$\begin{pmatrix} a \\ b \end{pmatrix} \preccurlyeq \begin{pmatrix} p \\ q \end{pmatrix} \Leftrightarrow p \leq max\left\{ \left\lfloor \frac{q}{b} \right\rfloor a, q + \left\lceil \frac{q}{b} \right\rceil (a-b) \right\} \tag{9}$$

According to Theorem 3 in [1] we have $\left(\frac{h_i}{w_i+h_i} \right) \equiv \overline{\left(\frac{w_i}{w_i+h_i} \right)}$. Hence, replacing the variables for our case, we get:

$$\begin{pmatrix} h_i \\ w_i + h_i \end{pmatrix} \preccurlyeq \begin{pmatrix} K_i - m_i \\ K_i \end{pmatrix} \Leftrightarrow K_i - m_i \leq max\left\{ \left\lfloor \frac{K_i}{w_i + h_i} \right\rfloor h_i, K_i - \left\lceil \frac{K_i}{w_i + h_i} \right\rceil w_i \right\} \tag{10}$$

For proving the theorem, we need to show that $K_i - m_i$ is always less or equal than the following terms:

$$\left\lfloor \frac{K_i}{w_i + h_i} \right\rfloor h_i \tag{11}$$

$$K_i - \left\lceil \frac{K_i}{w_i + h_i} \right\rceil w_i \tag{12}$$

We prove this theorem separately for low-tolerance and high-tolerance tasks. Summarizing, the steps are the following:

(1) We show Equation (11) is greater or equal than Equation (12) for high-tolerance tasks.
(2) We verify that Equation (11) is greater or equal than $K_i - m_i$ for high-tolerance tasks.
(3) We show Equation (11) is also greater or equal than Equation (12) for low-tolerance tasks.
(4) Then, for simplicity, we start by verifying that Equation (12) is greater or equal than $K_i - m_i$ for low-tolerance tasks. Showing this last is valid, we conclude that for low-tolerance tasks Equation (11) is also greater or equal than $K_i - m_i$.

Step 1. From Lemma 1, we know that $h_i = 1$ for high-tolerance tasks. Replacing $h_i = 1$ and assuming that Equation (11) ≥ Equation (12), we get:

$$\left\lfloor \frac{K_i}{w_i + 1} \right\rfloor \geq K_i - \left\lceil \frac{K_i}{w_i + 1} \right\rceil w_i \tag{13}$$

We consider the following relation between a real number x and a integer number n for removing the floor.

$$\lfloor x \rfloor \geq n \Leftrightarrow x \geq n \tag{14}$$

By removing the floor and then clearing K_i in Inequation (13), we get the assumption Equation (11) ≥ Equation (12) holds.

$$\frac{K_i}{w_i+1} \geq K_i - \left\lceil \frac{K_i}{w_i + 1} \right\rceil w_i \Rightarrow K_i \geq \left(K_i - \left\lceil \frac{K_i}{w_i + 1} \right\rceil w_i \right) (w_i + 1) \Rightarrow \left\lceil \frac{K_i}{w_i + 1} \right\rceil (w_i + 1) \geq K_i$$

Step 2. We need to prove:

$$\left\lfloor \frac{K_i}{w_i + 1} \right\rfloor \geq K_i - m_i \tag{15}$$

Using Relation 14 for removing the floor:

$$\frac{K_i}{w_i + 1} \geq K_i - m_i$$

Next, clearing w_i and replacing it for its value for high-tolerance tasks, i.e. $w_i = \left\lfloor \frac{m_i}{K_i - m_i} \right\rfloor$:

$$w_i \leq \frac{m_i}{K_i - m_i} \implies \left\lfloor \frac{m_i}{K_i - m_i} \right\rfloor \leq \frac{m_i}{K_i - m_i}$$

Inequation (15) holds and we prove Equation (10) for high-tolerance tasks.

Step 3. From Lemma 1, we know that $w_i = 1$ for low-tolerance tasks. Replacing $w_i = 1$ and assuming that Equation(11) is greater or equal than Equation(12), we get:

$$\left\lfloor \frac{K_i}{1 + h_i} \right\rfloor h_i \geq K_i - \left\lceil \frac{K_i}{1 + h_i} \right\rceil \tag{16}$$

For converting from ceiling to floor, we use the following identity:

$$\left\lceil \frac{a}{b} \right\rceil = \left\lfloor \frac{a + b - 1}{b} \right\rfloor \tag{17}$$

Hence:

$$\left\lfloor \frac{K_i}{1 + h_i} \right\rfloor h_i \geq K_i - \left\lfloor \frac{K_i + h_i}{1 + h_i} \right\rfloor \implies \left\lfloor \frac{K_i + h_i}{1 + h_i} \right\rfloor \geq K_i - \left\lfloor \frac{K_i}{1 + h_i} \right\rfloor h_i$$

Using Relation 14 for removing the floor in $\left\lfloor \frac{K_i + h_i}{1 + h_i} \right\rfloor$ and then clearing K_i:

$$\frac{K_i + h_i}{1 + h_i} \geq K_i - \left\lfloor \frac{K_i}{1 + h_i} \right\rfloor h_i \Rightarrow K_i + h_i \geq \left(K_i - \left\lfloor \frac{K_i}{1 + h_i} \right\rfloor h_i \right) (1 + h_i) \Rightarrow (1 + h_i) \left\lfloor \frac{K_i}{1 + h_i} \right\rfloor + 1 \geq K_i$$

We see that assumption Equation (11) $\geq$ Equation (12) holds.

Step 4. As we mentioned before, here we start by showing that Equation (12) is greater or equal than $K_i - m_i$:

$$K_i - \left\lceil \frac{K_i}{1 + h_i} \right\rceil \geq K_i - m_i \implies m_i \geq \left\lceil \frac{K_i}{1 + h_i} \right\rceil \tag{18}$$

We now use the following relation between a real number x and a integer number n to remove the ceiling from Inequation (18).

$$\lceil x \rceil \leq n \Leftrightarrow x \leq n$$

$$m_i \geq \left\lceil \frac{K_i}{1 + h_i} \right\rceil \implies m_i \geq \frac{K_i}{1 + h_i}$$

Next step is clearing h_i and replacing it for its value, i.e. $h_i = \left\lceil \frac{K_i - m_i}{m_i} \right\rceil$:

$$h_i \geq \frac{K_i - m_i}{m_i} \implies \left\lceil \frac{K_i - m_i}{m_i} \right\rceil \geq \frac{K_i - m_i}{m_i}$$

We see the assumption Equation (12) $\geq K_i - m_i$ holds and since also Equation (11) $\geq$ Equation (12), we get also Equation (11) $\geq K_i - m_i$. This proves Equation (10) for low-tolerance tasks which finalizes the proof of Theorem 5 of [1] for $\binom{h_i}{w_i + h_i} \preccurlyeq \binom{K_i - m_i}{K_i}$.