Guaranteeing weakly-hard timing constraints of real-time server-based systems

Nasim Samimi, Mitra Nasri, Twan Basten, Marc Geilen
Eindhoven University of Technology, Eindhoven, Netherlands

***Abstract*—Centralised servers provide on-demand resources to process offloaded workloads from computing nodes. While server-based computing has been successful for applications with soft timing constraints, it falls short for safety-critical real-time systems with hard timing requirements. To bridge this gap, we develop a job-level admission test to satisfy the requirements for real-time applications deployed on a server by extending the "(M, K)-firm weakly hard" model to server systems, ensuring timely processing of server requests. We introduce an admission policy to regulate the workload and prevent deadline misses while attempting to admit more requests than the minimum required by the initial (M, K) constraints. The admission policy is designed to allow an optimal resource allocation to applications deployed on the server.** [1]

***Index Terms*—Server-based systems, admission policy, worst-case response time analysis, arrival curve, real-time systems**

I. Introduction

Server-based platforms are centralised resources, commonly used for handling tasks like data processing, allowing end nodes to access computing resources easily. Applications like object recognition in self-driving cars [2] and unmanned aerial vehicles [3] utilise server-based computing where end nodes can offload their computation to a server (e.g., the cloud) [4].

Server-based architectures can be employed in real-time tasks (applications), where end nodes send requests to the corresponding tasks hosted on the server, exploiting a more powerful computational platform than the platform of the end nodes (often a micro-controller), hence allowing tasks to finish faster. Arrival curves [5] are commonly used to provide a generic model for the arrival of these requests in server-based systems to model a wide class of arrival patterns including sporadic, bursty, and periodic arrivals with release jitter, as the incoming requests do not necessarily arrive periodically (due to communication overheads between the end node and the server). Hence, server-based systems may not be able to guarantee hard deadlines, as they can be overloaded by a bursty arrival of requests from multiple nodes. To avoid such a situation, there is a need for a *runtime admission policy* to decide whether a new request should be admitted to the server or sent back to the end node. If admitted to the server, the request must meet its deadline under any future arrival scenario. Rejected requests initiate a fallback scenario on the end node, e.g., execute with reduced quality.

To ensure that end nodes do not suffer from frequent rejections and that the server fairly supports all applications, we adopt the (M, K) weakly-hard model in the admission policy. The server guarantees to process at least M out of any K consecutive requests and meet their deadlines. Once a request to an application (task) is admitted it becomes a job of the corresponding task.

Our policy ensures four conditions: (i) the weakly-hard constraints must be met, (ii) the selection of M out of K requests should minimise the required resources on the server considering that requests of different tasks arrive to the server with their own arrival curves, (iii) if the server cannot guarantee a request's timing constraint, it has to reject the request at its arrival time so that it can execute on the end node (e.g., using a fallback scenario), and (iv) all admitted requests on the server must meet their deadlines.

These conditions have been separately addressed in many works. For instance, various methods have been proposed to classify jobs of (M, K) tasks as mandatory and optional [6]–[11]. These, however, consider periodic or sporadic tasks and not generic arrival patterns. Some works followed the same idea using strategies like Markov-chain modelling [12] and the Chinese remainder theorem [13] to classify the jobs at design time. These not only have a long runtime, rendering them impractical to be used in an online admission policy, but also consider only periodic/sporadic tasks. Another method is the distance-based priority assignment (DBP) technique [14]–[18], where more generic arrival patterns such as Poisson distributions are considered. However, these methods cannot guarantee the deadlines of admitted jobs.

Some studies address the problem of bounding deadline misses for tasks with arrival curves [19]–[26]. Typical Worst-Case Analysis (TWCA) [19]–[24] is an approach to limit the number of jobs missing deadlines under weakly-hard constraints. However, these studies lack a runtime policy and thus do not predict if a job will fail to meet its deadline upon arrival. In short, none of the above-mentioned works satisfy all the conditions necessary for our problem.

This paper. We propose a job-level admission policy that ensures (M, K)-firmness and deadline constraints of tasks described by arrival curves, while minimizing the required number of cores. Our admission policy decides about the jobs upon their arrival. Rejected jobs are executed on end nodes (using a fall-back solution).

We employ periodic reservation servers, which are easy to analyse and have static behaviour. This allows isolation among the tasks to reduce interference, protecting task executions against potential overruns of other tasks, and therefore, achieve

[1] This work is an extension of our work-in-progress paper at RTAS'24 [1].

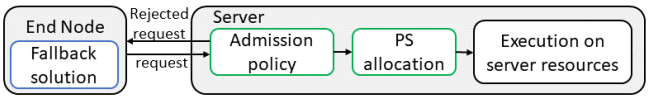

Figure 1: Overview of our framework at runtime

a more predictable timing behaviour. We use partitioned Earliest-Deadline-First (EDF) scheduling for a high per-core resource utilisation, to prevent migration overheads, and to improve schedulability [27]. Our contributions are:

- A method to determine the minimum number of cores for a server to uphold (M, K)-firmness constraints where requests for each task follow their own arrival curves.
- A low-overhead online admission policy (whose runtime is within a few microseconds), minimising the required number of cores while ensuring that admitted requests meet their deadlines.

Our evaluations show that our admission policy can admit more than M out of K jobs and its runtime is very small ($70\mu s$ on average) even for task sets with 25 tasks.

II. System model and problem description

1) Overall system architecture: We assume a server-based architecture with (i) a multicore server that can execute a set of predefined tasks (specified at design time) upon request and (ii) end nodes with limited resources. At runtime, end nodes send requests to the server trying to execute their tasks on the server's resources instead of their local resources (see Fig. 1). We call each request a *job*.

This work assumes that each task is deployed independently. Moreover, jobs are considered independent, single-threaded, and capable of concurrent execution under a preemptive scheduling policy. The jobs are transmitted through a reliable network. Upon receiving a job, the server decides on admission or rejection. Admitted jobs must complete before their deadlines. Admission must respect weakly-hard constraints. Jobs that are rejected (by the server) have to be executed locally on their end nodes, potentially with degraded quality. The fallback scenarios on the end nodes are not considered in this work.

2) Resource model: We model the resources in the server as m homogeneous CPU cores.

3) Workload model: We assume a set $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n weakly-hard real-time tasks that might need to be executed on the server. A task τ_i is a tuple $(\alpha_i^+, C_i^{max}, D_i, (M_i, K_i))$, where α_i^+ is the upper arrival curve, i.e., the maximum workload that may be sent to the server (see Sec. II-4), C_i^{max} is the worst-case execution time (WCET) of the task (including memory contention), and D_i is the relative deadline of the jobs. M_i and K_i define the weakly-hard constraints of the task, namely, at least M_i jobs out of any K_i consecutive jobs that arrive at the server must be executed on the server and meet their deadlines. We assume that the (M, K) constraints are given, and there are sufficient resources available in the end nodes to process $K - M$ jobs.

4) Workload arrival model: We assume that the maximum workload that task τ_i can send to the server follows an arrival curve, a flexible way to represent a wide class of known arrival models such as periodic, sporadic, bursty, and irregular arrivals. An arrival curve is denoted by a function $\alpha_i^+ : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$, with $\alpha_i^+(\Delta)$ representing the maximum workload of task τ_i (maximum number of jobs times WCET) that can arrive in the server in any time interval of length Δ.

5) Representing arrival curves: According to [28], an infinite arrival curve can be described by a (possibly empty) irregular finite curve, denoted by α_s, defining the beginning of the arrival curve up to interval Δ_s, followed by a regularly repeated curve (denoted by α_p). In the regular part, in each repetition, $N_\alpha \times C^{max}$ amount of workload (N_α jobs) can be emitted in an interval of Δ_p, the period. The first occurrence of the regular part starts at $(\Delta_s, N_s \times C^{max})$ (a point on the curve corresponding to the workload of N_s jobs).

Arrival curves are generally assumed to be step functions. That is, for $k \in \mathbb{N}$, both α_s and α_p can be expressed as

$$\alpha(\Delta) = \begin{cases} \omega_0 & 0 \leq \Delta < \Delta_1, \\ \ldots & \\ \omega_k & \Delta_{k-1} \leq \Delta < \Delta_k. \end{cases}$$

where the ω_k are constant values.

6) Job model: A request to a task τ_i is called a job. It has an absolute deadline d derived from the time of the request and relative deadline D_i of the task and it has a WCET C_i^{max}.

7) Scheduling and execution models: A periodic server (PS) is assigned, specifically, to each task. Each server represented by S_i is characterised by a pair (P_i, Q_i), where P_i is the period and Q_i is the budget of the server. The servers do not overrun their budget. We assume the worst-case context-switching overhead for preempting a server by higher-priority servers, represented by O_c. We consider this constant overhead each time a server is preempted, according to the model presented in [29], [30].

8) Problem description: Our goal is to propose an admission policy that strategically selects jobs to admit while respecting the (M, K)-firmness constraints of the tasks, guaranteeing that all admitted jobs are completed before their deadlines, and using a minimum number of cores.

9) Overview of the solution: The primary challenges in this work are (i) the strategic selection of jobs to be admitted for each task, aiming to minimise resource consumption while ensuring the admission of at least M out of every K consecutive jobs (see Sec. III); (ii) the allocation of a PS with minimum utilisation to each task, ensuring meeting the deadlines of admitted jobs (see Sec. IV), and (iii) scheduling the resulting PSs, according to the partitioned EDF scheduling policy with the least number of cores possible (see Sec. IV-B).

III. Admission policy of (M, K)-firm tasks

We introduce an online admission policy that admits jobs according to a static periodic *admission sequence*, represented by a finite binary sequence. This sequence classifies jobs

into mandatory and optional ('1' and '0' in the sequence, respectively). This classification is performed once, at design time. Mandatory jobs must be admitted, whereas optional jobs can be rejected. At run-time, we keep track of *free processing time* (FPT) which results from the slack between the actual completion time of jobs and their worst-case completion time to opportunistically admit some of the optional jobs if their deadlines can be guaranteed from the current FPT.

Definition 1 (Admission Policy)**.** *Given a binary admission sequence ζ, upon the arrival of a job, admit the job if it corresponds to a '1' in sequence ζ. Otherwise, admit it if the current FPT is at least equal to the WCET of the job; otherwise, reject it.*

In this section, first, we derive an optimal sequence for task τ such that at least M jobs out of any K consecutive jobs of a task τ are admitted to the server while minimising the amount of resources required by task τ. Subsequently, we need to determine the least upper bound on the admitted workload (called *admitted workload curve*) to facilitate resource allocation for task τ, at design time. Finally, we introduce FPT and a method to track and use FPT to admit optional jobs.

A. Deriving an optimal (M, K) sequence of jobs

The goal of this section is to select the set of M jobs of task τ, such that the amount of resources allocated to task τ is minimised. We achieve this by minimising the workload that must be admitted to the server, within any given interval.

Selecting the M jobs evenly across the K jobs will reduce the maximum amount of admitted workload in any interval during a burst (of jobs of the task), regardless of when the burst occurs (see Lemma 2 for the proof of the effectiveness of this method). We show this idea with an example.

Example 1. *Consider a periodic arrival curve in which the task releases two jobs at the same time with a period of $5ms$ and a deadline of $5ms$. The task has $C^{max} = 1ms$ and an (M, K) constraint with $M = 2$ and $K = 4$. This constraint allows us to reject at most two jobs per any four incoming jobs of this task. The following are two possible scenarios. First, we repeatedly admit and reject two jobs. Second, we repeatedly admit and reject one job, as shown in Fig. 2(a).*

In the first scenario, the PS is required to provide a minimum service of $2ms$ per $5ms$ to meet the deadline of the jobs. However, in the second scenario, the minimum required service to meet the deadline is reduced to $1ms$ per $5ms$.

We characterise the rejection and admission of K consecutive jobs by a static binary sequence of length K, where '0' means possible rejection and '1' means mandatory admission. This sequence is denoted by $\zeta = (\zeta_1, \zeta_2, \ldots, \zeta_K) \in \{0, 1\}^K$. The sequence is repeated for the next K consecutive jobs, and so forth, as shown in Example 2.

Example 2. *Consider task τ with an (M, K) constraint with $M = 3$ and $K = 4$ and admission sequence '1011'. Fig. 2(b) illustrates 8 consecutive jobs of task τ and the assigned bits according to its admission sequence. Jobs J_2 and J_6 are rejected; the remaining jobs are admitted.*

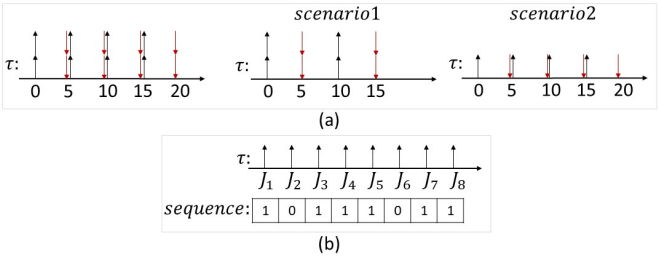

Figure 2: (a) A balanced admission policy reduces the peak task utilisation, (b) Assigning an admission sequence to incoming jobs. (black and red arrows illustrate the arrival and the deadline of a job, respectively)

Our goal is to construct an optimal admission sequence ζ_{opt} such that it minimises the maximum number of ones in any word of length 2 to length K. Let $\zeta : k : l$ be a word of length k of sequence ζ, starting at position l (the starting position of the word can range from 1 to K), and $\sum(\zeta : k : l)$ be the number of ones in such a word. ζ_{opt} is optimal if, for all sequences ζ of length K containing M ones and for all k, it satisfies the following:

$$\max_{1 \le l \le K} \sum(\zeta_{opt} : k : l) \le \max_{1 \le l \le K} \sum(\zeta : k : l). \tag{1}$$

To distribute the M ones evenly, we assign ${\zeta_{opt}}_k = 1$ for any $k \in ind(M, K)$ and ${\zeta_{opt}}_k = 0$ otherwise, where we compute $ind(M, K)$ as

$$ind(M, K) = \left\{ \left\lceil \frac{iK}{M} \right\rceil + 1 \;\middle|\; 0 \le i < M \right\}. \tag{2}$$

Example 3. *Consider task τ with a $(2, 5)$ constraint. We construct an optimal sequence ζ_{opt} for this task, according to Eq. (2). Since $ind(2, 5) = \{1, 4\}$, $\zeta_{opt} = (1, 0, 0, 1, 0)$.*

Lemma 1. *A sequence ζ_{opt} that is assigned ones according to the indices computed in Eq. (2) satisfies Eq. (1).*

Proof. We sketch a proof that any word of length k of any sequence ζ of length K with M ones has at least as many ones as any word of length k in ζ_{opt}.

According to Eq. (2), the shortest word in ζ_{opt}, containing n ones is $L(n) = 1 + \min_i \left\lceil \frac{(i+n-1)K}{M} \right\rceil - \left\lceil \frac{iK}{M} \right\rceil$. Using $i = M - (n-1)$, we find the following upper bound: $L(n) \le 1 + \left\lfloor \frac{(n-1)K}{M} \right\rfloor$. We can further show that this is also a lower bound on $L(n)$. Let $a, b, c, d \in \mathbb{N}$ be such that $iK = aM + b + 1$, $(n-1)K = cM + d$ and $0 \le b, d < M$ (such a, b, c, d exist according to Euclid's division lemma). Then we show that $\left\lceil \frac{(i+n-1)K}{M} \right\rceil - \left\lceil \frac{iK}{M} \right\rceil \ge \left\lfloor \frac{(n-1)K}{M} \right\rfloor$. Thus, $L(n) \ge 1 + \left\lfloor \frac{(n-1)K}{M} \right\rfloor$. Together, it follows that $L(n) = 1 + \left\lfloor \frac{(n-1)K}{M} \right\rfloor$.

We proceed to prove that any sequence ζ such that for some $n \in \mathbb{N}$, $L(\zeta, n) > L(n)$, where $L(\zeta, n)$ is the length of the shortest word in ζ containing n ones, cannot be (M, K), by contradiction. Let ζ be an (M, K) sequence. Then, in any word

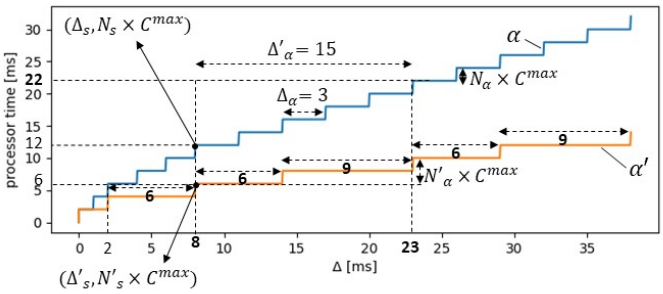

Figure 3: Deriving admitted workload curve in Example 4

of length $L(n)$, ζ has strictly less than n, so at most $n-1$, ones. Consider a word of length $L(n)\cdot K$. So ζ has at most $K\cdot(n-1)$ ones. We show $K\cdot(n-1)$, using $L(n) = 1 + \left\lfloor \frac{(n-1)K}{M} \right\rfloor$, to be strictly smaller than $M \cdot L(n)$, the number of ones needed to be (M, K). Thus, ζ cannot be (M, K). □

B. Deriving the admitted workload curve

Suppose that we admit only the mandatory jobs following the sequence derived above. We derive a *admitted workload curve*, denoted α', which represents the maximum workload of the task τ that must be *admitted* in any interval of time. It is derived at design time. Fig. 3, where $C^{max} = 2$, $K = 5$, and $M = 2$, shows an example. To derive α' for a task with arrival curve α, we must find, for any l, the minimum interval of time Δ_l in which a given amount $l \times C^{max}$ of workload for l jobs must be admitted, i.e., $\Delta_l = \min\{\Delta \in \mathbb{R}^{\geq 0} \mid \alpha'(\Delta) = l \times C^{max}\}$. E.g., in Fig. 3, the minimum interval in which 3 jobs are admitted is 8.

For any l admitted jobs, we must compute the shortest word of ζ_{opt} including l ones. Then, the minimum interval during which such a sequence arrives can be derived from the arrival curve. Therefore, we compute the shortest word containing l ones which for ζ_{opt}, see the proof of Lemma 1, is as follows.

$$WL(l) = 1 + \left\lfloor (l-1)\frac{K}{M} \right\rfloor \tag{3}$$

Using (3), we obtain the minimum interval of admitting l jobs as $\Delta_l = \min\{\Delta \in \mathbb{R}^{\geq 0} \mid \alpha(\Delta) = WL(l) \times C^{max}\}$.

Alg. 1 computes an admitted workload curve. We describe the admitted workload curve, similar to the arrival curve (see Sec. II-4), by an initial irregular part and a regular part starting at some point $\Delta'_s, N'_s \cdot C^{max}$ with a period of $N'_\alpha \cdot C^{max}$. Alg. 1 iteratively calculates an interval for N admitted jobs. It continues until we complete the irregular part and one repetition of the regular part of the admitted workload curve (α'). We construct the rest of α' by repeating the regular part.

Therefore, the search for new Δ' values in Alg. 1 continues until $WL(N) \leq H$ with $H = N_s + LCM(N_\alpha, K)$, where N_α is the period of the original arrival curve, N_s is the maximum number of jobs that can arrive following a non-repetitive pattern, K is the length of the admission sequence, and $LCM(N_\alpha, K)$ is the least common multiple of K and N_α, that is, the hyper period of the arrival curve and the sequence length. For example, in Fig. 3, the irregular part

Algorithm 1: Deriving a admitted workload curve

Input :
α: the arrival curve of task τ,
(M, K): firmness constraint,
C^{max}: WCET of task τ.

Output: α' : the admitted workload curve

1 N_α, N_s: derived from the arrival curve
2 $N \leftarrow 1$
3 $H \leftarrow N_s + LCM(N_\alpha, K)$
4 obtain $WL(N)$ from Eq. (3)
5 **while** $WL(N) \leq H$ **do**
6 $\quad \Delta' \leftarrow \min\{\Delta \mid \alpha(\Delta) = WL(N) \times C^{max}\}$
7 $\quad \alpha'(\Delta') \leftarrow N \times C^{max}$
8 $\quad N \leftarrow N + 1$
9 $\quad$ obtain $WL(N)$ from Eq. (3)

ends at $\Delta = 8$ and the period of the regular part is 15. Let $LCM(N_\alpha, K) = \eta_\alpha \times N_\alpha = \eta_K \times K$ be the hyper period of the arrival curve in which we can reject $\eta_K \times (K - M)$ jobs. Hence, we can compute the period in the admitted workload curve as $N'_\alpha = \eta_\alpha \times N_\alpha - \eta_K \times (K - M)$.

Suppose that at the end of the algorithm, N_{tot} is the value of N, that is to say, the admitted workload curve is derived for workload up to $N_{tot} \times C^{max}$. The derived admitted workload curve contains the irregular part and one repetition of the regular part. Thus, the starting point of the regular part in α' is computed as $N'_s = N_{tot} - N'_\alpha$ and $\Delta'_s = \Delta_s$.

Example 4. *Let us consider the arrival curve given in Fig. 3 for task τ in Example 3. We derive the admitted workload curve following the procedure in Alg. 1. From the arrival curve, $N_\alpha = 1$ and $N_s = 6$. Hence, $H = 6 + LCM(1, 5) = 11$ jobs (a workload of 22 on the y-axis). Considering that $LCM(N_\alpha, K) = 5$, and 3 jobs per hyper period are rejected, we compute the $N'_\alpha = 5 - 3 = 2$.*

We initially set the value of $N = 2$. The shortest word containing two ones, has a length of 3 (Eq. (3)). Hence, $\Delta_2 = \min\{\Delta \mid \alpha(\Delta) = 3 \times C^{max}\} = 2$. In the next step, we increase the value of N to 3. The shortest word containing three ones has a length of 6. Hence, $\Delta_3 = 8$. Again, we increase the value of N to 4. The shortest word containing four ones has a length of 8, i.e., $\Delta_4 = 14$. In the last step, the value of N is set to 5 and the shortest word containing five ones has a length of 11. That is, $\Delta_5 = 23$. We cannot increase the value of N because the WL has reached the maximum value, i.e., $N_{tot} = 5$. At this point, we compute $N'_s = N_{tot} - N'_\alpha = 5 - 2 = 3$.

Lemma 2. *The admitted workload curve derived from Alg. 1 is the least upper bound on the workload that must be admitted.*

Proof. (i) According to Lemma 1, the generated sequence in Sec. III-A ensures the property in Eq. (1). According to this property, the number of jobs that must be admitted in any sequence of the incoming jobs is minimised. (ii) Moreover, line 6 of Alg. 1 guarantees that for any word length only the

smallest Δ' (the minimum interval) is added to the admitted workload curve, because we search for the shortest sequence of incoming jobs. (i) and (ii) result in the least upper bound on the workload that must be admitted. □

C. Opportunistic admission of optional jobs

The occurrence of the WCET is not always a certainty. Hence, we employ a runtime strategy to benefit from potential opportunities, where jobs have shorter execution time than C^{max}. That is to say, the jobs may finish their execution earlier than the WCET, which frees some processor time between the actual completion time (CT) and the worst-case completion time (CT^{max}). When the processor becomes free, the generated free processor time is $CT^{max} - CT$. This free time gradually decreases over time with a slope of -1 and finishes at CT^{max}. We use this free processor time (denoted as FPT) to schedule optional jobs. If the generated FPT exceeds the WCET of an optional job J, we allocate that time to job J and admit the job. Part of the FPT is then consumed (FPT will be updated as $FPT - C^{max}$).

Note that the optional jobs cannot endanger the deadlines of any mandatory jobs, because the free processor time that they occupy cannot be bigger than the anticipated interference from mandatory jobs extending to their worst-case completion time CT^{max}.

Example 5. *Consider a full core allocated to a task τ with $C^{max} = 5$. The admission sequence of this task is derived as '1110'. Assume four jobs of this task $\{J_1, J_2, J_3, J_4\}$ arrive in the system, where, according to the sequence, J_4 can be rejected and the rest must be admitted. J_1, J_2, J_3 arrive at times $0, 1, 2$ and have the actual execution times of $C_1 = 3, C_2 = 2, C_3 = 2.5$, respectively. First, we show the changes of FPT over time, considering that no future jobs arrive. Then, we show the same diagram for the FPT of different scenarios, where J_4 with an actual execution time of 4 can arrive at times $2.5, 4, 6, 8, 11$.*

(i) When J_1 arrives in the system, it is scheduled on the core. The processor is free and $FPT = 0$ at this point. The worst-case completion time (which is computed considering the job starts as late as possible and has the WCET) of this job is 5. Then, J_2 arrives at time 1 and J_3 arrives at 2. Job J_1 is finished at 3 and J_2 is scheduled immediately. This generates a FPT of 2. Since the core is not free, FPT does not change. Then, at 5, J_2 is completed. The FPT of the processor is now changed to 5, and J_3 is immediately scheduled. Again, FPT does not change since the core is not free. At time 7.5, J_3 is completed, which increases the FPT to 7.5. The processor is free afterwards; hence, the FPT decreases and finally becomes zero at time 15 (the original worst-case completion time of the three jobs) (see Fig. 4(a)).

(ii) If J_4 arrives at 2.5, the FPT is still 0, i.e., there is not enough processor time to be granted to J_4; hence, J_4 is rejected. If J_4 arrives at time 4, $FPT = 2$. The processor time is still not sufficient; hence, J_4 is rejected. If J_4 arrives at time 6, $FPT = 5$, which is equal to the WCET of J_4; hence, J_4 consumes the FPT, is admitted and will be scheduled after J_3. The FPT becomes 0 at this point (see Fig. 4(b)). If J_4 arrives at 8, $FPT = 7$, which is greater than the C^{max}; thus, J_4 is admitted and consumes FPT (decreasing it to 2) (see Fig. 4(c)). If J_4 arrives at 11, $FPT = 4$; thus, J_4 is rejected.

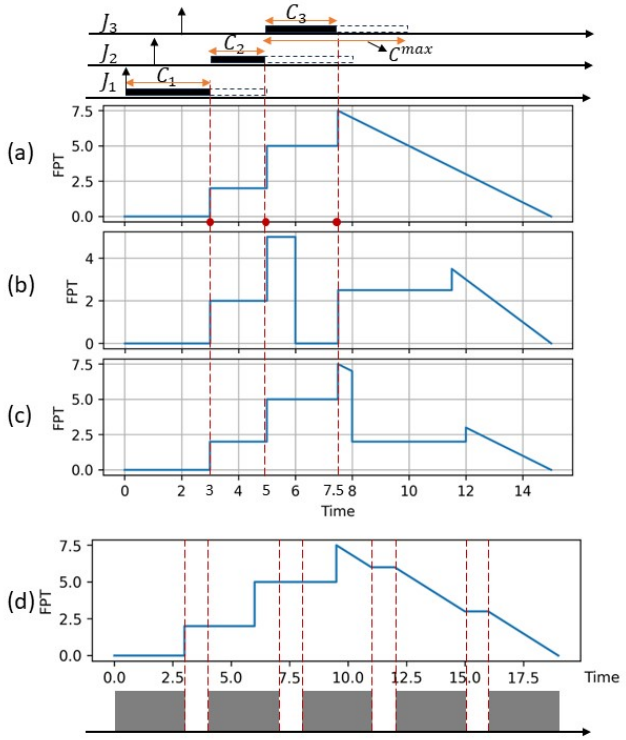

Figure 4: Tracking FPT on a processor in Example 5, (a) FPT of jobs $\{J_1, J_2, J_3\}$ (b) when J_4 arrives at 6, (c) when J_4 arrives at 8. (d) Tracking FPT from Example 6 on a PS with $(P, Q) = (4, 3)$

Example 6. *Now, consider a system where jobs of a task are executed on a PS (with utilisation smaller than 1, i.e., it does not use a full core). Fig. 4(d) illustrates the FPT of $\{J_1, J_2, J_3\}$ in Example 5 on a PS with period $P = 4$ and budget $Q = 3$. Note that the FPT is depleted only when the budget is available and no job is executed.*

Proposition 1. *If an optional job is admitted according to our proposed admission policy (Definition 1), it meets its deadline.*

Proof. We prove that for an optional job J from task τ with an arrival time of σ_J, WCET of C_J^{max} and relative deadline of D (absolute deadline of $d_J = \sigma_J + D$), its completion time, denoted by CT_J is less than or equal to d_J.

We prove this by contradiction, assuming that $CT_J > d_J$. According to our admission policy, an optional job is admitted if the $C_J^{max} \leq FPT_\tau$, where FPT_τ is the current FPT of task τ. FPT_τ reduces to 0 at $CT_{J_0}^{max}$, the worst-case completion time of job J_0, the last admitted job before J. Thus, $CT_J \leq CT_{J_0}^{max}$. This results in $CT_{J_0}^{max} > d_J$.

On the other hand, we know that J_0 meets the deadline (denoted by d_{J_0}), i.e., $CT_{J_0}^{max} \leq d_{J_0}$. Hence, $d_{J_0} > d_J$.

According to our definition of absolute deadline, $\sigma_{J_0} + D > \sigma_J + D$, which leads to $\sigma_{J_0} > \sigma_J$. This means that J_0 arrived after J. This contradicts the assumption that J_0 arrived before J, proving the claim that $CT_J \leq d_J$. □

IV. Scheduling (M, K) tasks with PS

In our solution, admitted jobs of a task τ execute within a PS dedicated to that task. We characterise the PS such that task τ meets the deadline. Then we find the minimum required resources to schedule a set of tasks with (M, K) constraints.

A. Deriving server parameters

A PS with period P and budget Q may not have access to the CPU during a time interval of length $2(P-Q)$ in the worst case (namely, when two consecutive instances of the PS are scheduled as far as possible from each other). Therefore, the PS does not guarantee to provide any service in $0 \leq \Delta < 2(P-Q)$. Furthermore, a PS (regardless of its priority) ensures a service of $\Delta-(P-Q)$ in any time interval of length Δ with $2(P-Q) \leq \Delta < P + (P-Q)$ (see [31] for more details). The guaranteed service curve of the PS in any interval of Δ is expressed by a periodic function with jitter of $P-Q$, as:

$$\beta^{(P,Q)}(\Delta) = \max\left(0, \left\lfloor \frac{\Delta-(P-Q)}{P} \right\rfloor Q, \Delta-(P-Q)-\left\lceil \frac{\Delta-(P-Q)}{P} \right\rceil (P-Q)\right), \tag{4}$$

where the second term expresses the case where the PS does not provide any service when $\Delta \leq 2(P-Q)$, and the third term is for when $\Delta > 2(P-Q)$.

According to the *real-time calculus* [5], the WCRT of a task τ with an arrival curve $\alpha(\Delta)$ that executes on a PS with a service curve of $\beta^{(P,Q)}(\Delta)$ is upper bounded by:

$$Del(\alpha, \beta^{(P,Q)}) \triangleq \sup_{\lambda \geq 0}\{\inf\{\delta \geq 0 : \alpha(\lambda) \leq \beta^{(P,Q)}(\lambda+\delta)\}\}, \tag{5}$$

where $Del(\alpha, \beta)$ is the maximum horizontal distance, called *delay*, between the arrival and the service curves. Thiele et al. [5] showed that the task τ is schedulable if and only if $Del(\alpha, \beta) \leq D$. Using (5), the minimum demand curve ($\underline{\beta}$) by the task to satisfy the deadline is computed as $\underline{\beta}(\Delta) \triangleq \alpha(\Delta - D) \quad \forall \Delta \in \mathbb{R}^{\geq 0}$, where $\alpha(\Delta-D)$ is the arrival curve delayed by the relative deadline. According to [32] and [33], the task τ is schedulable on a PS S with service curve $\beta^{(P,Q)}(\Delta)$ if and only if:

$$\beta^{(P,Q)}(\Delta) \geq \underline{\beta}(\Delta) \quad \forall \Delta \in \mathbb{R}^{\geq 0}, \tag{6}$$

i.e., the service curve of the PS satisfies the demand curve of the task. For details, we refer to [33], [34].

1) Optimal budget: With respect to the schedulability of the task τ, for a given period P, the optimal budget is the minimum budget Q that satisfies the inequality in Eq. (6).

2) Optimal period: For a given task τ, the optimal period is the period that results in the minimum utilisation of PS, while task τ is schedulable. Considering the budget and the context-switching overhead of the PS, namely O_c, the utilisation of the PS is $U = \frac{Q+O_c}{P}$, where $0 \leq U \leq 1$, or $P \geq Q + O_c$, which indicates a lower bound on the PS period. To find the optimal period of the PS, we search the values of $P > O_c$ and we increase it by 1 unit of time (depending on the system, the time scale can be μs, ms, etc.). For each value of P, we compute the minimum possible budget, according to Sec. IV-A1, and then we derive U. The optimal period is where U possesses the minimum possible value.

We do not need to search for the optimal period among infinitely many values because PS has a jitter of $2(P-Q)$. Hence, when we increase the period, if we increase the budget at the same or lower rate, i.e., maintaining the same level of utilisation or reducing it, the jitter also increases, i.e., the service curve of PS falls below the demand curve. Hence, by increasing the period, the required budget that maintains the schedulability of the task increases at a higher rate, i.e., the utilisation increases.

B. Scheduling a task set with (M, K) constraints

The output of the presented techniques in Sec. III and Sec. IV-A is a set of PSs, one for each task. We must respect the (M, K)-firmness constraint and the deadline of any admitted job. Therefore, the goal is to find the minimum number of cores such that all the PSs are schedulable.

We derive the minimum number of cores (PS sets) needed for the task set to be schedulable. By scheduling the tasks with PS, the scheduling problem is interpreted as scheduling of strictly periodic tasks, where the period and the budget of a PS are considered as the period and the *WCET* of the task. We employ the Worst-Fit decreasing algorithm [35] to schedule the PSs according to the partitioned EDF scheduling policy. The Worst-Fit algorithm aims to decrease fragmentation on cores by attempting to best utilise available resources.

V. Case Study

In this section, we evaluate (i) the performance of the proposed admission policy when the jobs do not necessarily have the worst-case arrival or WCET, and (ii) the runtime of the admission policy. The optimality of the admission sequence, PS parameters, functionality of the admission policy, deadlines guarantees and weakly hard constraints guarantees are theoretically proved throughout the paper. Our experiments are based on a medical imaging use case described in [36].

This use case considers a scenario in which a hospital uses a cloud connection to its medical imaging equipment. This enables surgeons to request the latest updates, such as 3D image processing analysis, during treatment from the cloud to gain additional insight into the medical procedure at hand. Ensuring the safety and performance of complex cloud-based image processing is crucial for live image-guided treatments. Cloud processing must be quick and efficient to avoid delays for the surgeon and patient.

In this particular case study, the medical imaging device is the end node. The application is crucial during image-guided therapy, so if there is a temporary disruption or failure in the cloud service, the surgeon must be able to switch back to the device and perform the image processing with lower accuracy. Hence, an (M, K) constraint is required to maintain a certain quality of service by limiting the number of image processing jobs that must be performed on the device itself.

A. Input data

We obtained our arrival patterns and cloud workload processing times from our industrial partner. To evaluate the performance of our approach against various systems, we created variations of those numbers within reasonable ranges and generated a set of tasks and jobs. Each treatment may take between 20 to 40 minutes, during which the surgeon sends requests every 5 to 15 minutes. The WCET may vary according to the type of image processing.

1) Taskset generation: We considered task sets with $n = 5$, 10, 15, 20, and 25 tasks and examined nine configuration points, where each point represents C^{max}/D with the deadline D of a task selected. We generated 50 sets of tasks for each configuration point. The minimum inter-arrival times were in the range $[1, 5]$ minutes, the maximum inter-arrival times were 5 to 10 minutes larger than the corresponding minimum inter-arrival times, and the WCET (C^{max}) was in the range $[1, 5]$ minutes. To ensure that a task is schedulable on a single core, the C^{max} of the task must be less than or equal to its minimum inter-arrival time. We set a K of 10 for all the tasks and M is randomly chosen from 5 to 10, considering that $M = 10$ is the highest quality of service (highest accuracy of 3D images) and $M = 5$ is the lowest quality of service (lowest acceptable accuracy of 3D images) that the user (hospital) expects.

2) Job set generation: To perform the experiments during runtime, we generated a set of jobs including at least 200 jobs per task following the arrival curve of its corresponding task. We draw a random arrival time for a job, between the minimum and maximum inter-arrival time of the corresponding task. The arrival of the first job is the generated number and for the next jobs, the generated number is the distance between the newly generated job and the previous job. The execution time of the job is randomly selected in the interval of $[0, C^{max}]$ of the corresponding task.

B. Runtime Evaluation

1) Experimental setup: We simulated a model of our real-time system, PSs, and admission policy using Python 3.11.0 on an $x64$-based Linux machine, Intel $i7$-9750H CPU operating on $2.60GHz$, and $16GB$ RAM.

2) Evaluation criteria: As we mentioned in Sec. III-C, the arrival of the workload does not necessarily follow the upper arrival curve. Hence, the amount of admitted workload at runtime can be higher than M out of K consecutive jobs. In our experiment, we measure the *acceptance ratio*, which is the average number of admitted optional jobs (jobs that were originally supposed to be rejected) per total number of optional jobs for each task. Also, we measure the runtime of the admission policy for each job.

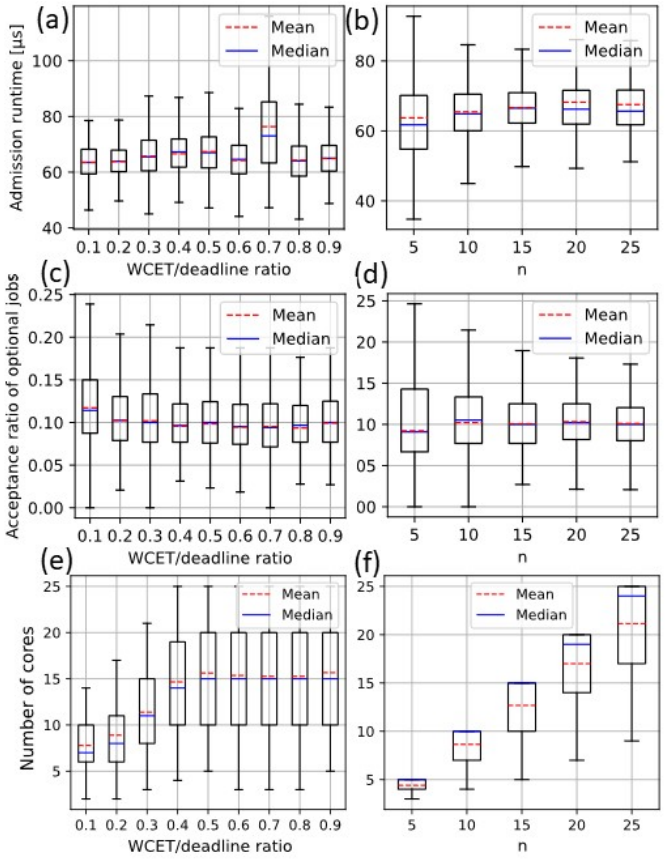

Figure 5: Runtime of the admission policy (in microseconds) per (a) WCET/deadline ratio, (b) number of tasks, acceptance ratio of optional jobs per (c) WCET/deadline ratio, (d) number of tasks, and number of allocated cores to each task set per (e) WCET/deadline ratio, (f) number of tasks

3) Evaluation results: Fig. 5(a) and (b) illustrate the average runtime of the admission policy to assess one job as a function of the WCET/deadline ratio and number of tasks in a task set, respectively. As shown in Fig. 5(a), the average runtime of the admission policy is $70\mu s$ and does not change considerably as the WCET/deadline ratio increases. Fig. 5(b) shows that increasing the number of tasks in a task set barely affects the admission policy runtime (about $70\mu s$ on average), since each task admission policy is conducted independently.

Fig. 5(c) and (d) illustrate the acceptance ratio of the task sets for each configuration point for $n = 5$, 10, 15, 20, and 25. Increasing WCET/deadline decreases the maximum acceptance ratio (from 25% to 20%); however, it does not impact the average acceptance ratio (10%). If the FPT of one PS ends before the arrival of the next optional job, the acceptance ratio of the optional jobs decreases. This usually happens in tasks with sparse arrivals. In the jobs with lower WCET/deadline, the generated FPT is larger and lasts longer. Additionally, Fig. 5(d) demonstrates that increasing the number of tasks in a set does not affect the average acceptance ratio (10% on

average with a maximum of 25% to 20%), since each task's admission policy is conducted independently of others using the admission sequence and FPT of the corresponding task. This shows the scalability of our method through employing PSs and partitioned scheduling.

Fig. 5(e) and (f) illustrate the number of cores allocated to each task set. The number of cores increases by increasing the WCET/deadline point (from an average of 7 to 15 cores). By increasing the WCET/deadline, the required utilisation of the PS increases. Since we used partitioned scheduling, by increasing the required utilisation of a task, at some point, only one task can be allocated to a core. Hence, the number of the cores becomes equal to the number of tasks in the task set. Additionally, the number of cores increases by increasing the number of tasks in a task set (4 to 22 on average).

VI. Conclusion

We have introduced a framework for managing resources that offers practical solutions to meet strict deadline requirements of real-time applications in server-based architectures. Our approach ensures efficient resource utilization and prevents requests from missing deadlines. These contributions can lead to the development of reliable and efficient real-time systems on servers. To validate our approach, we evaluated synthetic task sets inspired by a real-world industrial case study. The results demonstrated the effectiveness of our admission policy in meeting weakly hard constraints. In the future, we aim to develop a real-time Kubernetes-based [37] framework to integrate our proposed method into cloud platforms. This framework will manage resources while considering virtualisation, multi-server architectures, and the fluctuating resource demands of applications. Additionally, we plan to consider tasks that are not schedulable on a single core, network reliability and its overhead, scheduling with hard constant-bandwidth servers, and statistical distribution of execution time of the tasks.

VII. Acknowledgement

This work was supported by the EU ECSEL project TRANSACT (grant no. 101007260).

References

[1] N. Samimi, M. Nasri, T. Basten, and M. Geilen, "Work in progress: Guaranteeing weakly-hard timing constraints in server-based real-time systems," *RTAS*, 2024.

[2] K. Goldberg and B. Kehoe, "Cloud robotics and automation: A survey of related work," Tech. Rep. UCB/EECS-2013-5, EECS Department, University of California, Berkeley, 2013.

[3] J. Lee, J. Wang, D. Crandall, S. Sabanovic, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," *IEEE International Conference on Robotic Computing, IRC*, pp. 36–43, 2017.

[4] P. Mell and T. Grance, "The nist definition of cloud computing," 2011.

[5] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," *ISCAS*, vol. 4, pp. 101–104, 2000.

[6] G. Koren and D. Shasha, "Skip-over: algorithms and complexity for overloaded systems that allow skips," *RTSS*, pp. 110–117, 1995.

[7] A. Marchand and M. Silly-Chetto, "Rlp: Enhanced qos support for real-time applications," *RTCSA*, pp. 241–246, 2005.

[8] P. Ramanathan, "Overload management in real-time control applications using (m, k)-firm guarantee," *IEEE TPDS*, vol. 10, pp. 549–559, 1999.

[9] L. Niu and G. Quan, "Energy minimization for real-time systems with (m, k)-guarantee," *IEEE TVLSI*, vol. 14, pp. 717–729, 2006.

[10] H. Choi, H. Kim, and Q. Zhu, "Toward practical weakly hard real-time systems: A job-class-level scheduling approach," *IEEE IoTJ*, vol. 8, pp. 6692–6708, 2021.

[11] G. Bernat and A. Burns, "Combining (m,n)-hard deadlines and dual priority scheduling," pp. 46–57, 2002.

[12] D. Liu, X. S. Hu., M. D. Lemmon, and Q. Ling, "Firm real-time system scheduling based on a novel qos constraint," *IEEE TC*, vol. 55, pp. 320–333, 2006.

[13] G. Quan and X. Hu, "Enhanced fixed-priority scheduling with (m,k)-firm guarantee," *RTSS*, pp. 79–88, 2000.

[14] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE TC*, vol. 44, pp. 1443–1451, 1995.

[15] F. Wang and P. Mohapatra, "Using differentiated services to support internet telephony," *Comput. Commun.*, vol. 24, pp. 1846–1854, 2001.

[16] Z. Wang, J. M. Chen, and Y. X. Sun, "Integrated dbp for streams with (m, k)-firm real-time guarantee," *J. Zhejiang Univ. Sci.*, vol. 5, pp. 816–826, 2004.

[17] E. Poggi, Y.-Q. Song, A. Koubaa, and Z. Wang, "Matrix-dbp for (m, k)-firm real time guarantee," p. 27, 2003.

[18] J. Li, S. YeQiong, and S. L. Françoise, "Providing real-time applications with graceful degradation of qos and fault tolerance according to (m, k)-firm model," *IEEE TII*, vol. 2, pp. 112–119, 2006.

[19] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," *EMSOFT*, 2014.

[20] Z. A. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," *DATE*, pp. 584–589, 2017.

[21] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks," *ACM TECS*, vol. 18, 2019.

[22] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," *DATE*, pp. 515–520, 2012.

[23] W. Xu, Z. A. Hammadeh, A. Kroller, R. Ernst, and S. Quinton, "Improved deadline miss models for real-time systems using typical worst-case analysis," *ECRTS*, pp. 247–256, 2015.

[24] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein, "System-level timing feasibility test for cyber-physical automotive systems," *SIES*, 2016.

[25] L. Ahrendts, S. Quinton, and R. Ernst, "Finite ready queues as a mean for overload reduction in weakly-hard real-time systems," *RTNS*, pp. 88–97, 2017.

[26] L. Ahrendts, R. Ernst, and S. Quinton, "Exploiting execution dynamics in timing analysis using job sequences," *IEEE Des. Test*, vol. 35, pp. 16–22, 2018.

[27] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," *RTSS*, vol. 0, pp. 99–110, 2016.

[28] E. Wandeler, *Modular performance analysis and interface-based design for embedded real-time systems*. 2006.

[29] A. Burns, A. J. Wellings, and A. Hutcheon, "The impact of an ada run-time system's performance characteristics on scheduling models," *Ada-Europe*, pp. 240–248, 1993.

[30] M. H. Klein and T. Ralya, *An Analysis of Input-output Paradigms for Real-time Systems*. Citeseer, 1990.

[31] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," *RTSS*, pp. 2–13, 2003.

[32] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *RTSJ*, vol. 17, pp. 5–22, 1999.

[33] E. Wandeler and L. Thiele, "Interface-based design of real-time systems with hierarchical scheduling," *RTAS*, pp. 243–252, 2006.

[34] E. Wandeler and L. Thiele, "Optimal tdma time slot and cycle length allocation for hard real-time systems," *ASP-DAC*, pp. 479–484, 2006.

[35] S. Baruah, "Partitioned edf scheduling: A closer look," *RTSJ*, vol. 49, pp. 715–729, 2013.

[36] T. Hendriks, B. Akesson, J. Voeten, M. Hendriks, J. C. Parada, M. García-Gordillo, S. Sáez, and J. J. Valls, "Thirteen concepts to play it safe with the cloud," *SysCon*, 2023.

[37] B. Burns, J. Beda, K. Hightower, and L. Evenson, *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.