Budgeting Under-Specified Tasks for Weakly-Hard Real-Time Systems*

Zain A. H. Hammadeh[1], Sophie Quinton[2], Marco Panunzio[3], Rafik Henia[4], Laurent Rioux[5], and Rolf Ernst[6]

1 **Technische Universität Braunschweig, Braunschweig, Germany**
`hammadeh@ida.ing.tu-bs.de`
2 **Inria Grenoble – Rhône-Alpes, Grenoble, France**
`sophie.quinton@inria.fr`
3 **Thales Alenia Space, France**
`marco.panunzio@thalesaleniaspace.com`
4 **Thales Research & Technology, France**
`rafik.henia@thalesgroup.com`
5 **Thales Research & Technology, France**
`laurent.rioux@thalesgroup.com`
6 **Technische Universität Braunschweig, Braunschweig, Germany**
`ernst@ida.ing.tu-bs.de`

Abstract

In this paper, we present an extension of slack analysis for budgeting in the design of weakly-hard real-time systems. During design, it often happens that some parts of a task set are fully specified while other parameters, e.g. regarding recovery or monitoring tasks, will be available only much later. In such cases, slack analysis can help anticipate how these missing parameters can influence the behavior of the whole system so that a resource budget can be allocated to them. It is, however, sufficient in many application contexts to budget these tasks in order to preserve weakly-hard rather than hard guarantees. We thus present an extension of slack analysis for deriving task budgets for systems with hard and weakly-hard requirements. This work is motivated by and validated on a realistic case study inspired by industrial practice.

1998 ACM Subject Classification B.8.2 Performance Analysis and Design Aids

Keywords and phrases real-time, weakly-hard, slack analysis, execution budget, fixed priority

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2017.17

1 Introduction

In the design of real-time systems, it is not uncommon for some parts of a task set to be fully specified while other parameters, e.g. regarding recovery or monitoring tasks, will be available only much later. In such cases, slack analysis can help anticipate how these missing parameters can influence the behavior of the whole system so that a resource budget can be allocated to them. It is, however, sufficient in many application contexts to budget these tasks so as to preserve *weakly-hard* rather than hard guarantees. Such guarantees allow for a bounded number of consecutive deadline misses (*"at most m out of k deadlines may be*

* This work has been partially funded by the German Research Foundation (DFG) as part of the project "TypicalCPA" under the contract number TWCA ER168/30-1.

missed"). We thus present an extension of slack analysis for budgeting under-specified tasks for systems with hard and weakly-hard requirements.

The four main contributions of this paper are:

- an extension of slack analysis [11] to compute the maximum slack which guarantees that no more than m deadline misses out of k consecutive executions can happen;
- an execution time budget for under-specified tasks based on the multiframe task model [15] where we consider that each under-specified task has two execution times: a long one and short one;
- a methodology that explains why and how this method should be used safely in the design of systems that have hard and weakly-hard requirements;
- a case study dealing with satellite on-board software which shows the practical usefulness of weakly-hard constraints and how to guarantee them.

This paper is organized as follows. Section 2 introduces the application context of this paper, that is a satellite on-board software system. Section 3 then explains the timing verification problem that we are facing and why the proposed analysis is the appropriate solution to address it. Section 4 provides the preliminaries on response-time analysis which are needed to introduce our work. These preliminaries include principles of standard worst-case response-time analysis, typical worst-case analysis and slack analysis. Section 5 shows how to budget the under-specified tasks to satisfy hard real-time constraints. Our major contribution is in Section 6 where we present an extension of slack analysis and our general approach for budgeting based on the multiframe task model. We summarize the methodology that we propose in Section 7, present our experimental results in Section 8 and discuss related work in Section 9. Section 10 concludes.

2 Motivational Example

In this section we introduce the case study which motivates the work presented in this paper.

2.1 State of the practice for the timing analysis of satellite software

A satellite is made of two major parts: the platform and the payload. The payload realizes the main satellite mission, and comprises scientific instruments, telescopes or telecommunication antennas, according to the mission of the satellite. The payload is typically characterized by high computation requirements but in the general case its software is considered at best firm or soft real-time.

The platform is the service module that governs the satellite and ensures the execution of the mission. The platform on-board software (OBSW) implements all major functions of the satellite: e.g., the Attitude and Orbit Control System (AOCS), the Thermal Control System (TCS), mode management, Data Handling System (DHS).

A subset of those OBSW functions are characterized by hard real-time requirements. For example, sending thruster commands at the wrong moment during an attitude modification or an orbital maneuver (e.g., the main orbit insertion of a deep-space orbiter) may lead to mission failure.

In contrast, some tasks executing some less critical functions, may occasionally miss deadlines without dreadful consequences on the mission, and at most some performance degradation. One example is the AOCS functions itself, where sensor acquisition and processing are somehow robust to occasional deadline misses because of the intrinsic robustness of the implemented control laws.

The OBSW is however traditionally designed, analyzed and implemented with techniques typical of safety-critical, hard real-time systems. This implies that all tasks defined for the OBSW are considered as hard real-time and treated as such in the schedulability analysis used to confirm the system feasibility. The analysis is performed using representative worst-case operational scenarios. The reason for this choice is twofold:

1. It is much easier to prove to clients that the system is schedulable and fulfills the mission goals by treating all tasks as hard real-time, with a design process and analysis equations consolidated along several years, and without admitting exceptions on the treatment of task deadlines.
2. The OBSW development team does not know completely the possible consequences of deadline misses from the point of view of performance degradation or function losses, as such knowledge requires deep analysis at system / avionics level. It is therefore not obvious to understand if deadline misses are admissible in the overall mission context.

2.2 System model and use case

Current satellite OBSW is typically executed on a single-core processor, and using a Fixed-Priority Preemptive scheduling policy (FPP).

Table 1 shows a representative task set and the real-time attributes of each task. The attributes are representative of a high-load scenario for the OBSW in a mission operational mode. Each task τ_i in the system is characterized by its:

- priority index π_i; for simplicity of notation, we assume that tasks are given in order of their static priority, i.e., τ_j has higher priority than τ_i for every $j < i$;
- type of task release pattern: periodic (P), possibly with static offset, software sporadic (S), hardware sporadic (HWS), i.e., triggered by an interrupt, background task;
- worst-case execution time C_i; this value is not based on static analysis but rather on the observed execution times;
- period or interarrival time T_i;
- offset φ_i if applicable;
- relative deadline D_i – all deadlines are constrained;
- maximum blocking time b_i; execution in mutual exclusion is enforced with semaphores or protected objects (monitors) for which the maximum blocking can be bounded.

Note that some tasks specified as sporadic have in fact a pseudo-periodic behavior.

Table 1 includes two different kinds of tasks:

(i) *nominal tasks*: tasks that are active and executed in the represented operational scenario;
(ii) *recovery tasks*: tasks that are involved in asynchronous fault handling or recovery activities and are triggered only on given fault / error occurrences. They are marked as gray in the table.

Among the nominal tasks, some have real-time constraints that we will consider as hard real-time; others can be considered as weakly-hard real-time, as they can withstand occasional deadline misses without significant system-level consequences.

3 Problem Statement

The specification of recovery tasks typically occurs in the latest development phases, and therefore their characteristics are not known until late in the development cycle. The execution of such recovery tasks may however perturb the execution of nominal tasks, leading to deadline misses which would potentially induce a degradation of the system performance.

Table 1 A task set representative of on-board software. $\boldsymbol{\pi}$, $\boldsymbol{C}$, $\boldsymbol{T}$, $\boldsymbol{\varphi}$, $\boldsymbol{D}$ and $\boldsymbol{b}$ denote respectively: priority, worst-case execution time, period/minimum distance, offset, deadline, blocking time. The time unit is ms.

Name	π	**Type**	$\boldsymbol{C}$	$\boldsymbol{T}$	$\boldsymbol{\varphi}$	$\boldsymbol{D}$	$\boldsymbol{b}$
τ_1	1	HWS	0.56	15.625		15.625	0.1
τ_2	2	P	0.76	15.625		15.625	0.1
τ_3	3	P	15	125		31.25	0.1
τ_4	4	P	25.03	125	78.125	46.875	0.1
τ_5	5	P	7.5	62.5		62.5	0.1
τ_6	6	P	6.15	125		125	0.1
τ_7	7	P	1.2	125	93.750	125	0.1
τ_8	8	P	0.9	1000	500	500	0.1
τ_9	9	P	1.95	250		250	0.1
τ_{10}	10	S	▨	10000		125	0.1
τ_{11}	11	S	▨	▨		125	0.1
τ_{12}	12	P	1.2	125		125	0.1
τ_{13}	13	P	5.15	250	46.875	203.125	0.1
τ_{14}	14	P	1.2	1000		500	0.1
τ_{15}	15	P	22.5	500		500	0.1
τ_{16}	16	P	3.5	250		250	0.1
τ_{17}	17	P	27	500		500	0.1
τ_{18}	18	HWS	1.5	1000		1000	0.1
τ_{19}	19	P	16	1000		1000	0.1
τ_{20}	20	P	19.1	1000		1000	0.1
τ_{21}	21	S	▨	▨		1000	0.1
τ_{22}	22	P	88.8	2000		2000	0.1
τ_{23}	23	P	2	32000		32000	0.1
τ_{24}	24	P	1	32000		32000	0.1
τ_{25}	25	P	1	1000		1000	0.1
τ_{26}	26	S	20	1000		1000	0.1
τ_{27}	27	S	40	2000		2000	0.1
τ_{28}	28	S	1.5	2000		2000	0.1
τ_{29}	29	S	1.5	2000		2000	0
τ_{30}	30	P	0.2	32000		32000	0

Configuring the timing attributes of the recovery tasks represents a challenging timing issue for the real-time architect. It is important to guarantee that the reconfiguration and recovery tasks can accomplish their functions, which are related to the safety of the spacecraft. At the same time, it is necessary to preserve a sound timing behavior for the nominal tasks.

Moreover, the timing behavior of the recovery tasks must be established and assessed as early as possible in the development, as in later phases the development of the rest of the software system approaches completion, with little freedom for significant modifications.

The reader should note that the problem statement regards finding a convenient method for assigning attributes and guarantees to such tasks, rather than establishing a complete fault tolerance strategy [19][14] for the on-board software and the satellite. The latter requires a much more global reasoning at system level and it is not in the scope of this paper. Those tasks would be just some among several mechanisms (hardware and software) that are devoted to the implementation of such global fault tolerance strategy for a given satellite, and the method we seek would simply concur to their definition in a convenient manner.

To solve this problem, there is a need for a timing verification method fulfilling two conditions:

(i) applicability at early design stages;
(ii) a guarantee on the provided upper bounds for the tasks' response times.

Worst-case response time analysis seems well adapted to solve the timing challenge mentioned above, since its applicability already starts with the early conceptual design phases and it provides formal proofs based on a mathematical model of the system timing behavior. These proofs allow calculating safe lower and upper bounds on the response times, thus guaranteeing corner-case coverage.

Classic worst-case response-time analysis would however not be able to take into account the weakly-hard nature of some tasks, and would just check that deadlines of those tasks are met in the worst case. This would lead to an under-estimation of the timing budget available for the recovery tasks (and therefore to ensure the safety functions), which could be delicate, especially in case of a system with a high CPU load.

A method that takes into account the weakly-hard nature of some tasks and can provide to the real-time architect means to perform tradeoffs on the budget to be assigned to the recovery tasks would be considered attractive in this context.

Let us now formulate our problem. We consider a single processing resource which schedules a task set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ according to a Fixed Priority Preemptive (FPP) policy. Each task $\tau_i \in \mathcal{T}$ is modeled by its

- worst-case execution time C_i
- worst-case activation pattern η_i^+ (see below)
- priority π_i
- constrained deadline D_i

The tasks described in the motivating example of Section 2 are either periodic or sporadic. We will in this paper use the more general model of *arrival curves* to describe activation patterns, such that we can model sporadic tasks (in particular the recovery tasks that we want to budget) less conservatively than using a model based on the minimum interarrival time. We do not however handle offsets and conservatively assume that all periodic tasks can be activated at the same time. We leave to future the formal proof that offset analysis is compatible, as we conjecture, with the analysis presented in this paper. In contrast, blocking times are not mentioned in the rest of the paper for readability but they can easily be included in the analysis (and they are accounted for in the experiments).

▶ **Definition 1.** *Arrival curves* are functions $\eta_i^+, \eta_i^- : \mathbb{N} \to \mathbb{N}$ to model the possible activations of a task τ_i such that for any time window Δ, $\eta_i^+(\Delta)$ defines the maximum number of activations of τ_i that might occur within Δ, and , and η_i^- the minimum (in this paper we only use η_i^+). The pseudo-inverse of arrival curves, namely $\delta_i^-, \delta_i^+ : \mathbb{N} \to \mathbb{N}$, such that $\delta_i^-(k)$ (respectively $\delta_i^+(k)$) defines the minimum (respectively maximum) time that might pass between the first and the last activation in any sequence of k consecutive activations of τ_i.

In this context, the fact that deadlines are constrained translates into $D_i \leq \delta_i^-(2)$.

Our task set $\mathcal{T}$ is partitioned into *nominal tasks*, which are fully specified, and *recovery tasks*, for which only priorities and deadlines are known, such that we call these *under-specified tasks*. We denote by $\mathcal{N}$ the set of nominal tasks and $\mathcal{R}$ the set of under-specified tasks. Under-specified tasks are considered to be sporadic. Weakly-hard constraints are assumed to be given for nominal tasks.

Our problem is to provide a set of constraints on the execution times and the activation patterns of the tasks in $\mathcal{R}$ that is sufficient (and ideally necessary too) to guarantee (m, k)-schedulability of all tasks in $\mathcal{N}$, where a task is said to be (m, k)*-schedulable* if it cannot miss more than m deadlines out of a sequence of k consecutive executions.

4 Preliminaries on Response-Time Analysis

In this section we recall some state-of-the-art definitions and results on response-time analysis which we will use in the rest of the paper, based on the notations introduced at the end of our problem statement. We specifically present results related to worst-case response-time analysis, Typical Worst-Case Analysis (TWCA) and slack analysis.

Note that we suppose throughout this paper a representation of time based on natural numbers. This is reasonable since we consider single processor systems, which operate according to a unique, discrete clock.

4.1 Worst-case response-time analysis

A standard approach to establish schedulability of a system is to compute the worst-case response time of each task based on the concept of *busy window*. In this section we present results for the case where deadlines are arbitrary as we will need these later.

▶ **Definition 2.** A *level-i busy window* (originally called busy period in [10]) is a maximal time interval during which the resource still has activations of tasks of equal or higher priority than τ_i pending.

The longest such window, called *worst-case level-i busy window* and denoted BW_i, is built by assuming the occurrence of a so-called *critical instant*, where τ_i and higher-priority tasks are all activated at the same time, inducing maximum interference with τ_i. It is also assumed that all tasks are activated as early as possible after the critical instant, and that they always use their maximum execution time. The maximum level-i busy window stops at the first instant when no activation of τ_i or any higher priority task remains incomplete. It has been proven that the worst-case response time of task τ_i can be found in the longest level-i busy window.

▶ **Definition 3.** For a task τ_i and $q \geq 1$, the *multiple event busy time*, denoted $B_i(q)$, represents the maximum time it may take to process q activations of τ_i within a level-i busy window starting with the first of these q activations.

$$B_i(q) = \min\{\Delta T \geq 0 \mid \Delta T = q \times C_i + \sum_{\tau_j \in hp(i)} \eta_j^+(\Delta T) \times C_j\} \tag{1}$$

where $hp(i)$ denotes the set of tasks with higher priority than τ_i (we assume that all tasks have distinct priorities).

The maximum number K_i of activations of τ_i in a level-i busy window is then

$$K_i = \min\{q \geq 1 \mid B_i(q) < \delta_i^-(q+1)\}$$

K_i is the smallest number such that the resource would be able to start processing the $(K_i + 1)$-th activation before this activation can occur according to δ_i, which implies an idle time. The worst-case level-i busy window can then be determined as $BW_i = B_i(K_i)$. The response time of every activation of τ_i is bounded by

$$RT_i(q) = B_i(q) - \delta_i^-(q)\,.$$

The response time of τ_i is bounded by

▶ **Theorem 4.**

$$WCRT_i = \max_{1 \leq q \leq K_i} \{RT_i(q)\}\,. \tag{2}$$

We refer the reader to [21] for detailed explanations about the FPP response-time analysis.

4.2 Typical Worst-Case Analysis

Typical Worst-Case Analysis (TWCA) as presented e.g. in [17] [23] aims at providing weakly-hard guarantees for real-time systems, where a weakly-hard guarantee states that in no more than m out of k consecutive executions of a task, a deadline is missed. TWCA relies on the assumption that deadline misses in a system are due to transient overload resulting e.g. from sporadic activations.

We present here a specific application scenario of TWCA where activations of some specific tasks are considered as overload while activations of all other tasks are classified as typical.

We say that the system is in the *typical case* in a time interval in which there are no past or currently pending/executing overload activations which could impact the behavior of the system. We require the system to be schedulable in the typical case. The alternative case is called the *worst case* scenario where some overload activations may incur transient overload and therefore deadline misses.

The objective of TWCA is to compute a *deadline miss model* (DMM) for each task.

▶ **Definition 5.** A *deadline miss model* (DMM) for task τ_i is a function $dmm_i : \mathbb{N} \to \mathbb{N}$, with the property that out of any sequence of k consecutive activations (called k-sequence) of τ_i, at most $dmm_i(k)$ might miss their deadline D_i.

In the basic TWCA as introduced in [17], $dmm_i(k)$ is computed in four steps:

1. Computation of N_i, the number of deadline misses that occur in the longest level-i busy window BW_i. Note that one overload activation of any task cannot result in more than N_i deadline misses of τ_i as it can only impact activations of τ_i which are in the same level-i busy window.

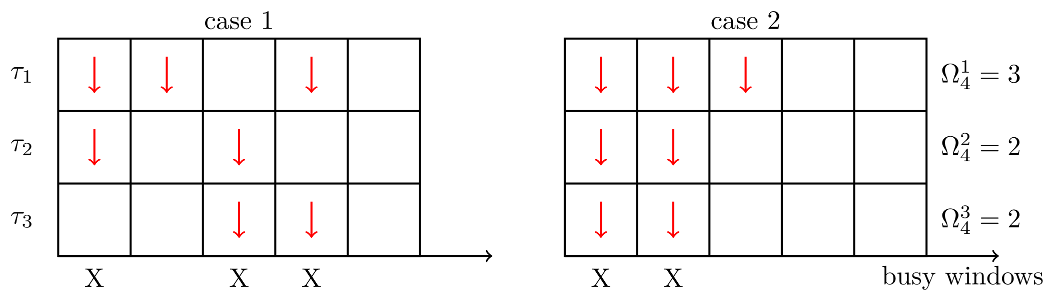

Figure 1 Packing overload activations into busy windows of task τ_4 (X = deadline miss).

2. Computation of ΔT_k^i, the longest time window during which an overload activation (of any task) can impact the response time of activations in the k-sequence. Activations in different busy windows cannot influence each other's response time. As a result, only activations of an overload task occurring at most BW_i time before the first activation of τ_i in the k-sequence and before the last activation finishes can have an impact. This time interval is thus bounded by:

$$\Delta T_k^i = BW_i + \delta_i^+(k) + WCRT_i\,.$$

3. Computation of Ω_i, the maximum number of higher-priority overload activations that may occur within a window of size ΔT_k^i:

$$\Omega_i = \sum_{\tau_j \in hp(i) \cap \mathcal{O}} \eta_j^+(\Delta T_k^i)$$

where $\mathcal{O}$ denotes the set of overload tasks.

4. We can then safely define $dmm_i(k) = \min\{k, \Omega_i \times N_i\}$.

The improved TWCA of [23] uses an additional concept called *combinations* to improve the accuracy of DMMs.

▶ **Definition 6.** A *combination* is a subset of the overload tasks, the idea being that one overload activation alone is usually not sufficient to cause a deadline miss as most tasks have some slack. Here we distinguish the overload due to different overload tasks:

$$\forall \tau_j \in \mathcal{O} \cap hp(i),\ \Omega_i^j = \eta_j^+(\Delta T_k^i)\,.$$

A bound on the maximum number of deadlines that τ_i may miss in a k-sequence is then obtained by packing overload activations into level-i busy windows.

▶ **Example 7.** As an example see Figure 1 where we consider a system with 4 tasks: τ_1, τ_2, τ_3 are overload tasks while τ_4 is a typical task. To bound the maximum number of deadlines that τ_4 may miss within a given time interval, we pack respectively Ω_i^1, Ω_i^2 and Ω_i^3 activations into the busy windows of τ_4. Figure 1 shows two possibilities of packing where in case 1 the number of deadline misses is 3 while in case 2 τ_4 may miss only 2 deadlines. Notice that not all combinations may lead to deadline misses.

▶ **Theorem 8.** *The following function is a DMM.*

$$dmm_i(k) = \min\{k, N_i \times \max\{\sum_{\bar{c} \in \mathcal{U}} x_{\bar{c}} \mid \forall \tau_j, \sum_{\{\bar{c} \in \mathcal{U} \mid \tau_j \in \bar{c}\}} x_{\bar{c}} \le \Omega_i^j\}\}$$

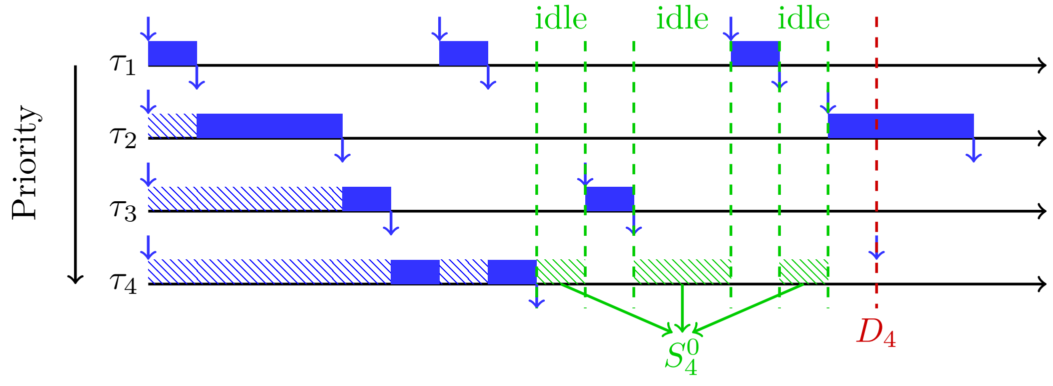

Figure 2 Worst-case busy window analysis. The slack S_4^0 of τ_i is shown.

where $\mathcal{U}$ is the set of unschedulable *combinations, i.e. combinations $\bar{c}$ which may lead to a deadline miss if all tasks in $\bar{c}$ are activated in the same level-i busy window. Note that in this approach it is assumed that all unschedulable may result in N_i deadline misses.*

[23] additionally provides an efficient criterion to determine whether a combination is schedulable as well as an efficient ILP solution to compute the above DMM.

4.3 Slack analysis

Finally, we now recall some results related to slack analysis [5],[11],[18], [20].

▶ **Definition 9.** The *slack* S_i^0 of task τ_i is the maximum amount of processing time which may be stolen from any job of τ_i without causing its deadline to be missed.

The slack of a task τ_i can be computed by noticing that any level-i idle time between the completion of a job of τ_i and its deadline can be used for computation of that job without causing it to miss its deadline.

▶ **Definition 10.** By *level-i idle time* we refer to any maximal time interval between two level-i busy windows.

▶ **Theorem 11.** *For FPP scheduling, the slack of τ_i is equal to the sum of all level-i idle times between the critical instant and D_i in the worst-case busy window.*

This is illustrated in Figure 2.

5 Budgeting with Hard Real-Time Constraints

In this section, we first focus on the problem of providing a set of constraints on the load incurred by the tasks in $\mathcal{R}$ (i.e. recovery tasks, a.k.a. under-specified tasks) that is sufficient to guarantee schedulability of all tasks in the nominal mode, before we move to discuss weakly-hard schedulability.

Let us first focus on a task τ_i in the nominal mode. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i. We can directly reuse the concept of slack to budget the under-specified tasks.

▶ **Lemma 12.** *Let S_i^0 be the slack of τ_i in the system made of only nominal tasks (i.e. excluding under-specified tasks). If $\sum_{\tau_r \in \mathcal{R}_i} C_r \leq S_i^0$ and $\delta_r^-(2) > D_i$ then τ_i is schedulable.*

Proof. This follows directly from the definition of slack. Note that we need to ensure that at most one activation of any under-specified task will interfere with a given job of τ_i for the result to hold. ◀

We can generalize the above result by splitting the load allocated to an under-specified task among several of its jobs.

▶ **Lemma 13.** *Let BW_i^0 be the longest level-i busy window obtained by analyzing the nominal task set with an additional load of size S_i^0. That is:*

$$BW_i^0 = \min\{\Delta T \geq 0 \mid \Delta T = C_i + S_i^0 + \sum_{\tau_j \in \mathcal{N} \cap\, hp(i)} \eta_j^+(\Delta T) \times C_j\}\,.$$

If $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^0) \times C_r \leq S_i^0$ then τ_i is schedulable.

Proof. Again, this follows directly from the definition of slack. In this case the slack used by an under-specified task τ_r is shared among several of its jobs. ◀

We can now state our general result on how to budget under-specified tasks to guarantee hard real-time schedulability of all nominal tasks.

▶ **Theorem 14.** *If for all $\tau_i \in \mathcal{N}$*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^0) \times C_r \leq S_i^0 \tag{3}$$

then the system is schedulable.

Proof. The above equation and Lemma 13 guarantee together than all nominal tasks remain schedulable in presence of under-specified tasks satisfying the given constraints. ◀

If this budget is acceptable then there is no need to consider budgeting for the weakly-hard case. The rest of this paper is dedicated to proposing solutions if a larger budget is needed for execution times of the under-specified tasks.

6 Budgeting with Weakly-Hard Real-Time Constraints

Our problem is now to provide a set of constraints on the load incurred by the tasks in $\mathcal{R}$ that is sufficient to guarantee *weakly-hard* schedulability of all tasks in the nominal mode rather than (hard) schedulability.

Again, we first focus on a task τ_i in the nominal mode, this time supposing that it has an (m, k) weakly-hard requirement, i.e. τ_i may miss no more than m out of k deadlines. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i.

As recalled in Section 4.2, the standard way to establish (m, k)-schedulability using Typical Worst-Case Analysis [23] is to consider a sequence of k consecutive activations of τ_i and to prove that no more than m activations in this sequence may miss their deadline. In our case the activations of under-specified tasks can be considered as overload since they are not taken into account by the initial worst-case analysis. We can therefore adapt TWCA to our context. We reuse in particular the following notations.

- N_i, the number of deadline misses that occur in the longest level-i busy window BW_i of the system with nominal and under-specified tasks.

- ΔT_k^i, the longest time window during which an activation of an under-specified task can impact the response time of activations in the k-sequence.

  $$\Delta T_k^i = BW_i + \delta_i^+(k) + WCRT_i\,.$$

- Ω_i^r, the maximum number of activations of higher-priority under-specified task τ_r that may occur within a window of size ΔT_k^i:

  $$\Omega_i^r = \eta_r^+(\Delta T_k^i)$$

  and Ω_i the sum over all higher-priority under-specified tasks:

  $$\Omega_i = \sum_{\tau_r \in \mathcal{R}_i} \Omega_i^r\,.$$

Notice here that budgeting according to constraints on N_i and ΔT_i^k is not easy as these parameters themselves depend on the parameters of the under-specified tasks. In the next section we first focus on how to relate the load budget of recovery tasks and N_i, i.e. the maximum number of deadline misses in a single busy window.

6.1 Extending the concept of slack to weakly-hard systems

Let us start with a few lemmas.

▶ **Lemma 15.** *There can be more than one activation of a given task τ_i in one level-i busy window only if that task misses its deadline in that busy window. Formally: $K_i \geq 2$ only if $WCRT_i > D_i$.*

Proof. By definition of K_i, $B_i(K_i) \leq \delta_i^-(K_i+1)$ and for any $q < K_i$, $B_i(q) > \delta_i^-(q+1)$. For $q = 1 : B_i(1) > \delta_i^-(2)$. We work with constrained deadlines so $D_i \leq \delta_i^-(2)$ so $B_i(1) > D_i$. As $B_i(1) = RT_i(1)$ and therefore $WCRT_i \geq B_i(1)$ we can conclude that $WCRT_i > D_i$. ◀

This lemma is easily generalized to consecutive deadlines misses: $\forall q < K_i, RT_i(q) > D_i$. This result is useful for us as it directly relates the number of deadline misses in a busy window with the length of that busy window. In particular, we obtain that $N_i = K_i - 1$ if $B_i(K_i) \leq \delta_i^-(K_i) + D_i$.

Let us now go one step further and extend the slack analysis of Section 4 to systems in which a bounded number of deadline misses are allowed.

▶ **Definition 16.** For $\mu \in \mathbb{N}$, the μ-slack of a task τ_i, denoted S_i^μ, is the maximum amount of processing time which may be stolen from τ_i in a level-i busy window without causing more than μ deadlines of τ_i to be missed in a row.

The μ-slack of a task τ_i can be computed in a way similar to the usual slack but focusing on the $(\mu+1)$-th deadline instead of the first deadline.

▶ **Theorem 17.** *For FPP scheduling, the μ-slack of τ_i is equal to the sum of all level-i idle times between the critical instant and $\delta_i^-(\mu+1) + D_i$ in the worst-case busy window.*

Proof. The above condition guarantees that the $(\mu+1)$-th deadline is met. ◀

Let us now introduce a definition which will be useful to bound BW_i and $WCRT_i$.

▶ **Definition 18.** Let BW_i^μ be the longest level-i busy window obtained by analyzing the nominal task set with an additional load of size S_i^μ. We know that such a busy window contains exactly $\mu + 1$ activations of τ_i so:

$$BW_i^\mu = \min\{\Delta T \geq 0 \mid \Delta T = (\mu + 1) \times C_i + S_i^\mu + \sum_{\tau_j \in \mathcal{N} \cap \, hp(i)} \eta_j^+(\Delta T) \times C_j\}\,.$$

Since τ_i may not miss more than m deadlines in a row, we can conclude that $BW_i \leq BW_i^m$. Similarly $WCRT_i$ is bounded by the response times of τ_i observed in BW_i^m. We thus know how to define ΔT_i^k. Let us now state the condition which guarantees that τ_i may not miss more than m deadlines in a row, and thus $N_i = m$.

▶ **Lemma 19.** *If $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(BW_i^m) \times C_r \leq S_i^m$ then τ_i cannot miss more than m deadlines in a row.*

Proof. This is a direct consequence of the definition of m-slack. ◀

At this point, it may seem that the intuitive, if pessimistic, way to budget the under-specified tasks is to require that $\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leq S_i^m$. This, however, is not a sufficient condition for (m, k)-schedulability. The reason is that the same load incurred by under-specified tasks may result in more deadline misses if they happen in different busy windows. This is the meaning of the following lemma.

▶ **Lemma 20.**

$$\forall \mu \in \mathbb{N}^+ : S_i^\mu \geq (\mu + 1) \times S_i^0\,.$$

Proof. Consider a sequence of $\mu + 1$ consecutive activations of τ_i. Remember that S_i^0 is the sum of all level-i idle times between the critical instant and D_i in the worst-case busy window. Because deadlines are constrained, this is smaller than or equal to the sum of all level-i idle times between the critical instant and $\delta_i^-(2)$ in the worst-case busy window. Allowing only S_i^0 slack for each activation in the sequence furthermore assumes that the critical instant may repeat for each activation, which is pessimistic compared to the way S_i^μ is computed. As a result, S_i^μ provides more slack than $(\mu + 1) \times S_i^0$. ◀

The consequence of this is that a safe bound on the budget for the under-specified tasks must be based for now on S_i^0.

▶ **Lemma 21.** *Let $\Lambda_i = (m + 1) \times S_i^0$. If*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leq \Lambda_i$$

then τ_i is (m, k)-schedulable.

Proof. We have to prove that a load of Λ_i within ΔT_i^k causes no more than m consecutive deadline misses if it occurs in one level-i busy window of τ_i, and no more than m non-consecutive deadline misses if it distributes over several busy windows.

- The first condition is directly satisfied by Lemmas 19 and 20.
- Suppose now that Λ_i is distributed over n level-i busy windows with l_b denoting the load in each busy window: $\sum_{b=1}^n l_b = \Lambda_i$. For each l_b let μ_b denote the maximum number of (consecutive) deadline misses that may be caused by l_b ($\mu_b \geq 0$). We have to prove that

$\sum_{b=1}^{n} \mu_b \leq m$. By definition we know that $l_b > S_i^{\mu_b - 1}$ for all l_b so from Lemma 20 we can derive that $l_b > \mu_b \times S_i^0$. If we now sum this over all l_b we get

$$\sum_{b=1}^{n} l_b > \sum_{b=1}^{n} \mu_b \times S_i^0 \,.$$

Since $\sum_{b=1}^{n} l_b = \Lambda_i = (m+1) \times S_i^0$ we can conclude that $m + 1 > \sum_{b=1}^{n} \mu_b$, which is what we had to prove. ◀

▶ **Theorem 22.** *If for all $\tau_i \in \mathcal{N}$ with an (m, k) schedulability constraint*

$$\sum_{\tau_r \in \mathcal{R}_i} \eta_r^+(\Delta T_i^k) \times C_r \leq (m+1) \times S_i^0 \tag{4}$$

then the system satisfies its hard and weakly-hard requirements.

Proof. This results is a direct consequence of Lemma 21. ◀

This result is obviously quite pessimistic. It is clear at this point that obtaining better bounds requires us to use a more fine-grained model of how load distributes over busy windows. We investigate this possibility in the next section.

6.2 Budgeting for multiframe tasks

In the following, we focus on a specific application scenario and assume that each under-specified task performs two activities:

- A frequent monitoring activity with a relatively short execution time aiming at analyzing deviations from safe state in the system and perform some rapid recovery or triggering higher-level recovery, characterized by a short minimum distance between two consecutive occurrences.
- A less frequent failure recovery activity (e.g., an avionics reconfiguration procedure) which requires a longer execution time and characterized by a longer minimum time distance between two consecutive executions.

Based on the behavior described above, the execution time model of any under-specified task τ_r can be characterized by (C_r^l, C_r^s, x) where:

- C_r^s is the short execution time corresponding to the recovery activity of the task;
- C_r^l is the long execution time corresponding to the error handling activity of the task;
- x is the number of short execution times between two long execution times.

Based on this new model we again address the problem of providing a set of constraints on the execution times and activation patterns of the tasks in $\mathcal{R}$ that is sufficient to guarantee weakly-hard schedulability of all tasks τ in the nominal mode.

Let us first focus on a task τ_i in the nominal mode with an (m, k) weakly-hard requirement, i.e. τ_i may miss no more than m out of k deadlines. Denote $\mathcal{R}_i$ the set of under-specified tasks with a priority higher than τ_i, $\Omega_i^r = \eta_r^+(\Delta T_i^k)$ for all $\tau_r \in \mathcal{R}_i$ and $\Omega_i = \sum_{r \in \mathcal{R}_i} \Omega_i^r$.

Let us first by formulating a hypothesis which is consistent with the application scenario mentioned at the beginning of this section.

▶ **Hypothesis 1.** *For each task $\tau_r \in \mathcal{R}_i$, we allow only one instance out of Ω_i^r to have a long execution time C_r^l. The other $\Omega_i^r - 1$ activations of τ_r within ΔT_i^k will be bounded by the short execution time bound C_r^s.*

In a way that is similar to the state of the art in TWCA as explained in Section 4.2 we now introduce the concept of combinations.

▶ **Definition 23.** A *level-i combination* is a tuple $\bar{c} = (c_1, c_2, \ldots, c_{|\mathcal{R}_i|})$ such that each task $\tau_r \in \mathcal{R}_i$ corresponds to one c_r in the tuple and $c_r = 0$ or $c_r = C_r^s$ or $c_r = C_r^l$.

We use the notation $c_r^{\bar{c}}$ to refer to the execution time of τ_r in combination $\bar{c}$. Note that we exclude here the possibility for several activations of the same under-specified task to be in the same level-i busy window. That is, we suppose that $\forall \tau_r \in \mathcal{R}_i : \delta_r^-(2) > BW_i^m$.

▶ **Definition 24.** Let $\mu(\bar{c})$ denote the maximum number of deadlines misses which may be caused by a combination $\bar{c}$. Formally we have:

$$S_i^{\mu(\bar{c})-1} < \sum_{\tau_r \in \mathcal{R}_i} c_r^{\bar{c}} \leq S_i^{\mu(\bar{c})}$$

with the convention that $S_i^{-1} = 0$. If $\mu(\bar{c}) = 0$ then $\bar{c}$ is called *schedulable*, otherwise it is said to be *unschedulable*.

Of course $\mu(\bar{c})$ depends on the values chosen for the various execution times C_r^l and C_r^s for $\tau_r \in \mathcal{R}_i$. Our strategy for budgeting the under-specified tasks is to first assign values on $\mu(\bar{c})$ for all combinations and then in a second step to assign execution time budgets.

▶ **Hypothesis 2.** *We suppose that a combination containing only short execution times of under-specified tasks cannot be unschedulable. That is,* $\sum_{\tau_r \in \mathcal{R}_i} C_r^s \leq S_i^0$.

Again this hypothesis seems realistic given the application context.

Based on the notion of combination we can define *gangs* which correspond to distributions of the Ω_i instances within ΔT_i^k. More specifically, a gang is a packing of activations of the under-specified tasks into the level-i busy windows of ΔT_i^k.

▶ **Definition 25.** A *gang* $\mathcal{G}$ is a set of combinations which contain at least one long execution time and such that for all $\tau_r \in \mathcal{R}_i$

- $\#\{\bar{c} \in \mathcal{G} \mid c_r^{\bar{c}} > 0\} \leq \Omega_i^r$
- $\#\{\bar{c} \in \mathcal{G} \mid c_r^{\bar{c}} = C_r^l\} = 1$

Notice that we ignore combinations which do not contain any long execution time as they cannot lead to deadline misses. Note also that each combination appears at most once in a gang (since there can be only one long execution time of each task within ΔT_i^k).

We use $\mathbb{G}_i$ to denote all possible gangs with respect to τ_i.

▶ **Lemma 26.** *If* $\forall \mathcal{G} \in \mathbb{G}_i : \sum_{\bar{c} \in \mathcal{G}} \mu(\bar{c}) \leq m$ *then* τ_i *is* (m,k)*-schedulable.*

Proof. The above condition guarantees that no matter how activations of under-specified tasks align, they can never result in more than m deadline misses. ◀

This lemma trivially extends to upper bounds on the $\mu(\bar{c})$ as we formulate now.

▶ **Lemma 27.** *For all* $\bar{c}$, *let* $\mu_{\bar{c}}$ *be an upper bound on* $\mu(\bar{c})$. *If* $\forall \mathcal{G} \in \mathbb{G}_i : \sum_{\bar{c} \in \mathcal{G}} \mu_{\bar{c}} \leq m$ *then* τ_i *is* (m,k)*-schedulable.*

Now, one thing which does not appear in the above lemma is that the $\mu(\bar{c})$ are not independent from each other.

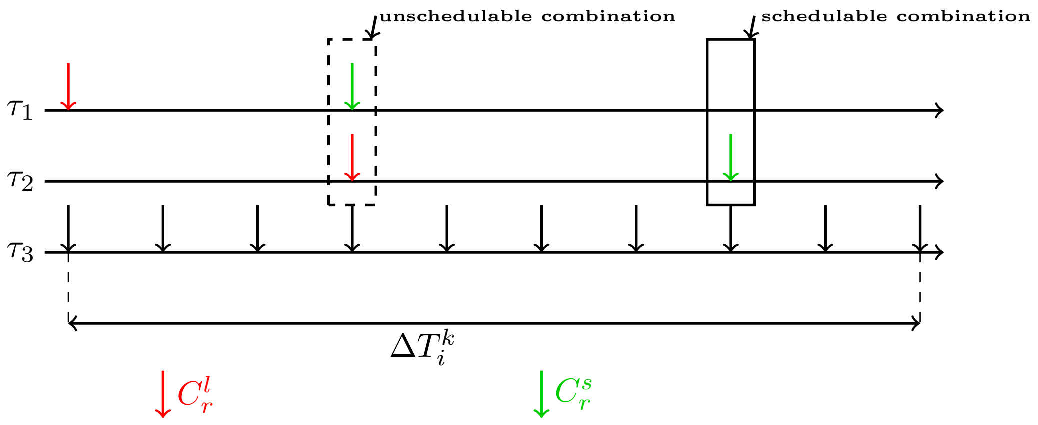

Figure 3 A gang of τ_1 and τ_2 within ΔT_i^k where τ_3 has a real time constraints (2,10).

▶ **Definition 28.** There exists a partial order $\leq$ on combinations such that $\bar{c}_1 \leq \bar{c}_2$ if and only if the execution times in $\bar{c}_1$ are all smaller than their counterpart in $\bar{c}_2$, i.e.,

$$\forall \tau_r \in \mathcal{R}_i : c_r^{\bar{c}_1} \leq c_r^{\bar{c}_2} .$$

▶ **Lemma 29.** *If $\bar{c}_1 \leq \bar{c}_2$ then $\mu(\bar{c}_1) \leq \mu(\bar{c}_2)$.*

Proof. This directly follows from the fact that $\bar{c}_1 \leq \bar{c}_2$ implies that the load incurred within one level-i busy window by the under-specified tasks in $\bar{c}_1$ is smaller than that in $\bar{c}_2$. ◀

▶ **Theorem 30.** *Suppose that you have assigned the $\mu_{\bar{c}}$ such that $\forall \mathcal{G} \in \mathbb{G}_i : \sum_{\bar{c}\in\mathcal{G}} \mu_{\bar{c}} \leq m$. Then any assignment of the $c_r^{\bar{c}}$ such that for all combination $\bar{c}$, $\sum_{r\in\mathcal{R}_i} c_r^{\bar{c}} \leq S_i^{\mu_{\bar{c}}}$ guarantees the (m,k)-schedulability of τ_i.*

Proof. This follows directly from Lemma 27 and the definition of $\mu_{\bar{c}}$-slack. ◀

Note that there always exists such an assignment.

Now that we have presented our solution for budgeting under-specified tasks based on the multiframe execution time model, let us show how it proceeds on an illustrative example.

▶ **Example 31.** Consider as an example a system with only one task τ_3 in the nominal mode and two under-specified tasks τ_1 and τ_2, as illustrated in Figure 3. Task τ_3 has a $(2,10)$ weakly-hard requirement. τ_1 and τ_2 have priorities higher than the priority of τ_3, and no more than 2 instances within ΔT_i^k.

Figure 3 shows gang $\mathcal{G} = \{\bar{c}_1, \bar{c}_4, \bar{c}_7\}$ where $\bar{c}_1 = (C_1^l)$, $\bar{c}_4 = (C_1^s, C_2^l)$ and $\bar{c}_7 = (C_2^s)$ – to improve readability we omit 0s in the representation of combinations.
There are five combinations containing at least one long execution time:

$$\bar{c}_1 = (C_1^l), \bar{c}_2 = (C_2^l), \bar{c}_3 = (C_1^l, C_2^s), \bar{c}_4 = (C_1^s, C_2^l), \bar{c}_5 = (C_1^l, C_2^l) .$$

There are three more combinations containing at least one short execution time:

$$\bar{c}_6 = (C_1^s), \bar{c}_7 = (C_1^s), \bar{c}_8 = (C_1^s, C_2^s) .$$

Let us now focus on gangs. Remember that gangs consist of combinations containing at least one long execution time and that two combinations with the long same execution time cannot be in the same gang. We only list here maximal gangs.

$$\mathcal{G}_1 = \{\bar{c}_1, \bar{c}_2\}, \mathcal{G}_2 = \{\bar{c}_1, \bar{c}_4\}, \mathcal{G}_3 = \{\bar{c}_2, \bar{c}_3\}, \mathcal{G}_4 = \{\bar{c}_3, \bar{c}_4\}, \mathcal{G}_5 = \{\bar{c}_5\} .$$

This yields the following constraints, the first five of which are directly derived from the gangs while the remaining four constraints are obtained by comparing combinations.

1. $\mu_{\bar{c}_1} + \mu_{\bar{c}_2} \leq 2$
2. $\mu_{\bar{c}_1} + \mu_{\bar{c}_4} \leq 2$
3. $\mu_{\bar{c}_2} + \mu_{\bar{c}_3} \leq 2$
4. $\mu_{\bar{c}_3} + \mu_{\bar{c}_4} \leq 2$
5. $\mu_{\bar{c}_5} \leq 2$
6. $\mu_{\bar{c}_1} \leq \mu_{\bar{c}_3}$
7. $\mu_{\bar{c}_3} \leq \mu_{\bar{c}_5}$
8. $\mu_{\bar{c}_2} \leq \mu_{\bar{c}_4}$
9. $\mu_{\bar{c}_4} \leq \mu_{\bar{c}_5}$

One solution to this set of constraints is e.g. $\mu_{\bar{c}_1} = 1, \mu_{\bar{c}_2} = 1, \mu_{\bar{c}_3} = 1, \mu_{\bar{c}_4} = 1, \mu_{\bar{c}_5} = 2$.

Assuming we have chosen the above assignment for the $\mu_{\bar{c}}$ we now the define the constraints to be satisfied by the execution times of tasks, one per combination and then one for the short execution times.

1. $C_1^l \leq S_i^1$
2. $C_2^l \leq S_i^1$
3. $C_1^l + C_2^s \leq S_i^1$
4. $C_1^s + C_2^l \leq S_i^1$
5. $C_1^l + C_2^l \leq S_i^2$
6. $C_1^s + C_2^s \leq S_i^0$

Any solution to this set of constraints guarantees (m, k)-schedulability of τ_i.

7 Methodology and Discussion

Let us now summarize the methodology that we propose to provide the architect with *simple answers* helping him/her dimension the tasks that are still under-specified in the system.

1. We first compute an execution time budget for the under-specified tasks which guarantees *hard real-time* constraints (zero deadline misses). If this execution time budget is acceptable for the architect then we do not need to go further.
2. If, however there is a need for larger execution times for the under-specified tasks, we then compute a second execution time budget which guarantees weakly-hard constraints. Taking into account *weakly-hard constraints* we can allow more load within shorter time windows but over longer time windows the load available for under-specified tasks is still limited.
3. If the activation patterns of the under-specified tasks are known and a *multiframe execution time model* is meaningful we can propose more relaxed bounds on execution times budgets.

8 Experimental Results

Let us now provide some experimental results we have obtained using the cplex constraint solver on budgeting under-specified tasks. We first address the motivational example of Section 2 and then present experiments made on synthetic test cases.

8.1 The OBSW case study

The case study presented in Section 2 is a system made of a single resource and a task set shown in Table 1 where 27 tasks are in the nominal mode and there are 3 recovery and reconfiguration tasks τ_{10}, τ_{11}, τ_{21} which are under-specified.

As discussed before in Section 2, all on-board software is currently typically analyzed with hard real-time techniques; and yet by experience, the overall system is still quite robust to

Table 2 Real-time constraints of tasks in $\mathcal{T}'$. (m, w) represents the maximum number of allowed deadline misses m every w seconds, (m, k) means that a task may miss at most m deadline out of k consecutive activations.

task	τ_{12}	τ_{13}	τ_{14}	τ_{15}	τ_{16}	τ_{17}	τ_{18}	τ_{19}	τ_{20}
(m, w)	(1,2)	(1,4)	(1,8)	(1,4)	(1,4)	(1,4)	(1,8)	(1,8)	(1,8)
(m, k)	(1,16)	(1,16)	(1,8)	(1,8)	(1,16)	(1,8)	(1,8)	(1,8)	(1,8)
task	τ_{22}	τ_{23}	τ_{24}	τ_{25}	τ_{26}	τ_{27}	τ_{28}	τ_{29}	τ_{30}
(m, w)	(1,16)	hard	hard	(1,8)	(1,8)	(1,16)	(1,16)	(1,16)	hard
(m, k)	(1,8)	hard	hard	(1,8)	(1,8)	(1,8)	(1,8)	(1,8)	hard

occasional deadline misses, although at the moment there is no necessity to formally evaluate such tolerance in the state-of-the-practice process.

For the sake of the case study we propose some weakly-hard constraints for tasks that are purposely quite aggressive: the reader could notice that in some cases a tolerance of 1 deadline every 2 seconds is admitted for some tasks. This would permit to ascertain the robustness (at least from the point of view of real-time constraints) of such representative task set even in case of severe degradation (which would require high sporadic load for the recovery activities).

The worst-case response time analysis of the nominal mode shows that the system is schedulable. Our goal is to synthesize a load budget for the under-specified tasks τ_{10}, τ_{11}, τ_{21} which guarantees that all weakly-hard real-time constraints described in Table 2 are satisfied. We show first the constraints on the execution times and activation models of the tasks in $\mathcal{R}$ which guarantee absence of any deadline miss before providing the same result when a few deadline misses are tolerated.

Note that tasks $\{\tau_1, \ldots, \tau_9\}$ have higher priority than the recovery and reconfiguration tasks so their timing properties do not depend on the budget of tasks in $\mathcal{R}$. They will therefore be excluded from our study. We denote by $\mathcal{T}'$ the remaining tasks with lower priority, that is: $\mathcal{T}' = \mathcal{T} \setminus \{\tau_1, \ldots, \tau_9\}$.

8.1.1 Budgeting with hard real-time constraints

If we want to guarantee that the system is schedulable then the budget to be shared between the under-specified tasks is $S_i^0 = 48.01$ms. If this budget is not sufficient for the architect we can propose a budget with weakly-hard real-time guarantees.

8.1.2 Budgeting with weakly-hard real-time constraints

If the architect can accept to work with weakly-hard rather than hard guarantees then the available budget for the recovery tasks is $(m + 1) \times S_i^0 = 96.02$ ms.

This budget is twice as much as the budget for the hard real-time case. We can obtain even better bounds by using a more fine-grained model of how load distributes over busy windows.

8.1.3 Budgeting for multiframe tasks

Let us assume that for all $\tau_i \in \mathcal{N}$ there are at most $\Omega_{10} = 1$, $\Omega_{11} = 3$ and $\Omega_{21} = 2$ activations of the under-specified tasks within ΔT_i^k. The following execution times guarantee

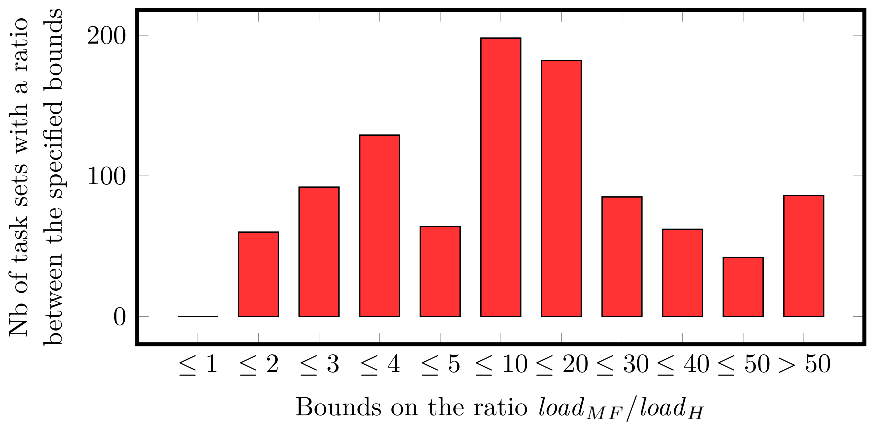

Figure 4 The relation between $load_{MF}$ and $load_H$.

(m, k)-schedulability of all tasks.

$$C^l_{11} = 24.005,\ C^s_{11} = 12.0025$$

$$C^l_{10} = 24.005,\ C^s_{10} = 12.0025$$

$$C^l_{21} = 24.005,\ C^s_{21} = 12.0025$$

This means in particular that the budget that is available for the under-specified tasks within ΔT^k_i is at least 108.015 ms. Note that there are many other possible assignments for the μ values which lead to different execution times.

8.2 Synthetic examples

In this section, we present a set of synthetic test cases to test more extensively our approach on a variety of systems. In this experiment we study the impact of different characteristics such as utilization, (m, k) constraints, system size, etc.

For that purpose we generated 1000 task sets randomly depending on UUniFast [7]. We define a set of tasks $\mathcal{T}$ with a priority, a worst-case execution time, a period, a deadline, and an (m, k) real-time constraint. The standard approach is to first define the system utilization and then assign a share of it to each task [7]. We picked up a utilization among $\{0.4, 0.5, 0.6, 0.7, 0.8\}$, then the number of tasks are chosen to be $\in [1, 20]$ and periods are *harmonic*. The worst-case execution time is then computed $C_i = U_i * T_i$. Deadlines $= \{0.6, 0.8, 1\} * T_i$ as our approach supports only constrained and implicit deadlines. We generate a random (m, k) for each task in the system such that: $k \in [2, 100], m \in [1, k-1]$. The number of under-specified tasks is limited to $r = 3$ and the maximum number of instances of each under-specified task is generated randomly to be in $[1, r^2]$.

8.2.1 Results

Figure 4 shows in the form of a histogram how much we gain in terms of load budget for the under-specified tasks by using a multiframe task model with weakly-hard constraints instead of using a single worst-case execution time with hard real-time constraints. Note that the results in the former case are obviously at least as good as those for the latter case.

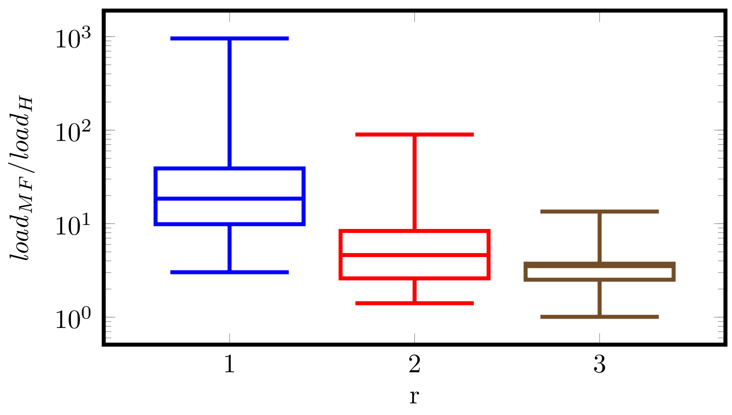

Figure 5 The relation between the gain of load and r.

Figure 4 shows for example that for 198 task sets the load budget in the multiframe case ($load_{MF}$) is between 5 and 10 times larger than the load budget in the hard case ($load_H$), that is:

$$5 < \frac{load_{MF}}{load_H} \leq 10\,.$$

The load we gain, however, is related to the number of under-specified tasks. Figure 5 shows that the larger the number of under-specified tasks the less load we gain, that is due to sharing the available slack among more under-specified tasks which makes the long execution C^l shorter.

The results shows that there is no impact of the utilization on the load we gain. Number of periodic tasks causes no degradation on the load we gain by using multiframe task model. Note that we have repeated our experiment 10 times and observed similar results.

9 Related Work

The work presented in this paper most closely relates to sensitivity analysis, slack analysis, multiframe task systems and weakly-hard real-time systems. Note that determination of bounds on unspecified system parameters is the scope of *Parametric Model Checking* [6] [9]. Even if such approaches are known to have difficulty scaling up to even simple settings, it would be interesting to see if these approaches could apply to our problem.

This work focuses on budgeting under-specified tasks for weakly-hard real-time systems. Although the under-specified tasks in our case study (OBSW) are recovery tasks, schedulability analysis of fault-tolerant real-time systems [4] [12] is not in the scope of this paper.

Sensitivity analysis is used to provide guarantees on the schedulability of a system in case of uncertainty on the system parameters. In [2] Bini et al. introduced an analytical sensitivity analysis for FPP scheduled periodic task sets with constrained deadlines (i.e. $D \leq T$). Work by [22] and [16] propose solutions for sensitivity analysis of systems with activation patterns specified with arrival curves.

In contrast to all these papers, our work proposes for the first time a solution for the sensitivity analysis of weakly-hard real-time systems: We constrain the admissible load that under-specified tasks in the system can use without violating weakly-hard real-time constraints in FPP scheduled task sets with arbitrary activation patterns and constrained deadlines.

Slack stealing is a scheduling algorithm proposed by [11] to schedule aperiodic tasks by stealing all the processing time it can from the periodic tasks without causing their deadlines to be missed. Similar algorithms based on slack stealing have been proposed by other authors [5] [18] [20]. These algorithms do not take into account any weakly hard guarantees and they, therefore, bound the maximum slack in a window of size D_i. In our approach, however, we consider (m, k) weakly-hard requirements and we thus bound the maximum slack in a window of size $\delta_i^-(m+1) + D_i$.

Multiframe task model was invented originally in [15] to provide a less pessimistic schedulability test than [13] for hard real-time systems. This model assigns to each *periodic* task N execution times $(C^0, C^1, \ldots, C^N)$, the execution time alternates between them where the execution time of the i-th instance of the task is $C^{((i-1) mod N)}$ where $i \geq 1$. In this paper we use a specific case of the multiframe task model for *sporadic* tasks which assigns two execution times: long C^l and short C^s where within a time window Δ one instance of the task uses the long execution time while the rest use the short execution times. We propose our multiframe task model in a context of budgeting *under-specified* recovery tasks (sporadic) to provide the recovery tasks with more load for weakly-hard real-time systems.

Weakly-hard systems [1] is a concept which guarantees that out of k consecutive executions of a task, not more than m deadline misses may occur. The approach of [17] and the related articles provide analyses to verify such constraints. In this paper we reuse the concepts developed in these papers to better budget under-specified tasks.

10 Conclusion

In this paper, we have shown how to budget under-specified tasks in the early design of weakly-hard real-time systems by providing sufficient conditions which guarantee (m, k) schedulability. This is particularly useful in industrial practice because it often happens during design that some parts of a task set are fully specified while other parameters, e.g. regarding recovery or monitoring tasks, do not become available before much later. Existing budgeting techniques, which are restricted to hard real-time constraints, can help anticipating how these missing parameters influence the behavior of the whole system, but they are likely to yield execution time budgets that are too tight to be useful. We have shown that using weakly-hard rather than hard guarantees, whenever possible, results in much more applicable execution time budgets. Our results are thus of real practical value for the design of systems such as the on-board software system discussed in the paper.

Note that in this paper we have not at all addressed the issue of the complexity of the analysis. The reason for that is that this does not appear to be a limiting factor for industrial applicability at this point. It would however be interesting to better understand how far the approach presented in this paper can scale and how much we can improve its efficiency.

Finally, we need to acknowledge the need for complementary work related to weakly-hard real-time systems as mentioned in Section 2, in particular in relation with the impact of deadline misses on system functions. Recent work [8, 3] in this direction indicate that this question is indeed considered as relevant in the research community as well as in the industry.

References

1 Guillem Bernat, Alan Burns, and Albert Llamosí. Weakly hard real-time systems. *IEEE Trans. Comput.*, 50(4):308–321, April 2001. `doi:10.1109/12.919277`.

2 Enrico Bini, Marco Di Natale, and Giorgio C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *18th Euromicro Conference on Real-Time Systems,*

ECRTS'06, 5-7 July 2006, Dresden, Germany, Proceedings, pages 13–22, 2006. doi:10.1109/ECRTS.2006.26.

3 Rainer Blind and Frank Allgöwer. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *54th IEEE Conference on Decision and Control, CDC 2015, Osaka, Japan, December 15-18, 2015*, pages 7510–7515, 2015. doi:10.1109/CDC.2015.7403405.

4 A. Burns, R. Davis, and S. Punnekkat. Feasibility analysis of fault-tolerant real-time task sets. In *Proceedings of the Eighth Euromicro Workshop on Real-Time Systems*, pages 29–33, June 1996. doi:10.1109/EMWRTS.1996.557785.

5 R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority pre-emptive systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 222–231, December 1993. doi:10.1109/REAL.1993.393496.

6 Conrado Daws. Symbolic and parametric model checking of discrete-time markov chains. In *Proceedings of the First International Conference on Theoretical Aspects of Computing*, ICTAC'04, pages 280–294. Springer-Verlag, 2005. doi:10.1007/978-3-540-31862-0_21.

7 P. Emberson, R. Stafford, and R. I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, pages 6–11, July 2010.

8 Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 53–62, December 2014. doi:10.1109/RTSS.2014.28.

9 Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits Vaandrager. Linear parametric model checking of timed automata. *The Journal of Logic and Algebraic Programming*, 52:183–220, 2002. doi:10.1016/S1567-8326(02)00037-1.

10 John P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990. doi:10.1.1.379.6163.

11 J. P. Lehoczky and S. Ramos-Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Real-Time Systems Symposium, 1992*, pages 110–123, December 1992. doi:10.1109/REAL.1992.242671.

12 George Lima and Alan Burns. Scheduling fixed-priority hard real-time tasks in the presence of faults. In *Proceedings of the Second Latin-American Conference on Dependable Computing*, LADC'05, pages 154–173, 2005. doi:10.1007/11572329_14.

13 C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973. doi:10.1145/321738.321743.

14 Ma Maode and H. Babak. A fault-tolerant strategy for real-time task scheduling on multi-processor system. In *Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on*, pages 544–546, June 1996. doi:10.1109/ISPAN.1996.509038.

15 Aloysius K. Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Trans. Software Eng.*, 23(10):635–645, October 1997. doi:10.1109/32.637146.

16 Moritz Neukirchner, Sophie Quinton, Tobias Michaels, Philip Axer, and Rolf Ernst. Sensitivity analysis for arbitrary activation patterns in real-time systems. In *Proc. of Design Automation and Test in Europe (DATE)*, March 2013. URL: http://dx.doi.org/10.7873/DATE.2013.041, doi:10.7873/DATE.2013.041.

17 Sophie Quinton, Matthias Hanke, and Rolf Ernst. Formal analysis of sporadic overload in real-time systems. In *DATE*, pages 515–520, March 2012. doi:10.1109/DATE.2012.6176523.

18 S. Ramos-Thuel and J. P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-priority systems. In *Real-Time Systems Symposium, 1993., Proceedings.*, pages 160–171, December 1993. doi:10.1109/REAL.1993.393504.

19 P. Rubel, M. Gillen, J. Loyall, R. Schantz, A. Gokhale, J. Balasubramanian, A. Paulos, and P. Narasimhan. Fault tolerant approaches for distributed real-time and embedded systems. In *MILCOM 2007 – IEEE Military Communications Conference*, pages 1–8, October 2007. doi:10.1109/MILCOM.2007.4455043.

20 Too-Seng Tia, Jane W.-S. Liu, and Mallikarjun Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Systems*, 10(1):23–43, 1996. doi:10.1007/BF00357882.

21 Ken Tindell, Alan Burns, and Andy J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, March 1994. doi:10.1007/BF01088593.

22 Ernesto Wandeler and Lothar Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *Proceedings of the 5th ACM International Conference on Embedded Software*, EMSOFT'05, pages 80–89, 2005. doi:10.1145/1086228.1086246.

23 Wenbo Xu, Zain A. H. Hammadeh, Alexander Kröller, Sophie Quinton, and Rolf Ernst. Improved deadline miss models for real-time systems using typical worst-case analysis. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*, Lund, Sweden, July 2015. doi:10.1109/ECRTS.2015.29.