CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Techniques and Analysis for Mixed-criticality Scheduling with Mode-dependent Server Execution Budgets

Muhammad Ali Awan
Konstantinos Bletsas
Pedro F. Souto
Benny Akesson
and Eduardo Tovar

CISTER-TR-190906

Techniques and Analysis for Mixed-criticality Scheduling with Mode-dependent Server Execution Budgets

Muhammad Ali Awan, Konstantinos Bletsas, Pedro F. Souto, Benny Akesson, and Eduardo Tovar

CISTER Research Centre

Polytechnic Institute of Porto (ISEP P.Porto)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

https://www.cister-labs.pt

Abstract

In mixed-criticality systems, tasks of different criticality share system resources, mainly to reduce cost. Cost isfurther reduced by using adaptive mode-based scheduling arrangements, such as Vestal 19s model, to improveresource efficiency, while guaranteeing schedulability of critical functionality. To simplify safety certification,servers are often used to provide temporal isolation between tasks. In its simplest form, a server is a periodicallyrecurring time window, in which some tasks are scheduled. A server 19s computational requirements maygreatly vary in different modes, although state-of-the-art techniques and schedulability tests do not allowdifferent budgets to be used by a server in different modes. This results in a single conservative executionbudget for all modes, increasing system cost.The goal of this paper is to reduce the cost of mixed-criticality systems through three main contributions:(i) a scheduling arrangement for uniprocessor systems employing fixed-priority scheduling within periodicservers, whose budgets are dynamically adjusted at run-time in the event of a mode change, (ii) a newschedulability analysis for such systems, and (iii) heuristic algorithms for assigning budgets to servers indifferent modes and ordering the execution of the servers. Experiments with synthetic task sets demonstrateconsiderable improvements (up to 52.8%) in scheduling success ratio when using dynamic server budgets vs.static 1cone-size-fits-all-modes" budgets.

Techniques and Analysis for Mixed-criticality Scheduling with Mode-dependent Server Execution Budgets*

MUHAMMAD ALI AWAN*, CISTER Research Centre and ISEP/IPP, Portugal
KONSTANTINOS BLETSAS*, CISTER Research Centre and ISEP/IPP, Portugal
PEDRO F. SOUTO†, University of Porto, Faculty of Engineering and CISTER Research Centre, Portugal
BENNY AKESSON‡, Embedded Systems Innovation, Eindhoven and University of Amsterdam, Amsterdam, the Netherlands
EDUARDO TOVAR*, CISTER Research Centre and ISEP/IPP, Portugal

In mixed-criticality systems, tasks of different criticality share system resources, mainly to reduce cost. Cost is further reduced by using adaptive mode-based scheduling arrangements, such as Vestal's model, to improve resource efficiency, while guaranteeing schedulability of critical functionality. To simplify safety certification, servers are often used to provide temporal isolation between tasks. In its simplest form, a server is a periodically recurring time window, in which some tasks are scheduled. A server's computational requirements may greatly vary in different modes, although state-of-the-art techniques and schedulability tests do not allow different budgets to be used by a server in different modes. This results in a single conservative execution budget for all modes, increasing system cost.

The goal of this paper is to reduce the cost of mixed-criticality systems through three main contributions: (i) a scheduling arrangement for uniprocessor systems employing fixed-priority scheduling within periodic servers, whose budgets are dynamically adjusted at run-time in the event of a mode change, (ii) a new schedulability analysis for such systems, and (iii) heuristic algorithms for assigning budgets to servers in different modes and ordering the execution of the servers. Experiments with synthetic task sets demonstrate considerable improvements (up to 52.8%) in scheduling success ratio when using dynamic server budgets vs. static "one-size-fits-all-modes" budgets.

CCS Concepts: • **Computer systems organization → Real-time systems**; **Real-time operating systems**; **Real-time system architecture**.

Additional Key Words and Phrases: real-time systems, mixed-criticality scheduling, vestal model, mode change, server-based scheduling, dynamic execution Budget, timing isolation

ACM Reference Format:
Muhammad Ali Awan, Konstantinos Bletsas, Pedro F. Souto, Benny Akesson, and Eduardo Tovar. 2019. Techniques and Analysis for Mixed-criticality Scheduling with Mode-dependent Server Execution Budgets. *J. ACM* 37, 4, Article 111 (August 2019), 23 pages. https://doi.org/10.1145/1122445.1122456

*This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.

Authors' addresses: Muhammad Ali Awan, awa@isep.ipp.pt, CISTER Research Centre and ISEP/IPP, Rua Dr. António Bernardino de Almeida 431, Porto, Portugal, 4249-015; Konstantinos Bletsas, CISTER Research Centre and ISEP/IPP, Porto, Portugal, ksbs@isep.ipp.pt; Pedro F. Souto, University of Porto, Faculty of Engineering and CISTER Research Centre, Porto, Portugal; Benny Akesson, Embedded Systems Innovation, Eindhoven and University of Amsterdam, Amsterdam, the Netherlands; Eduardo Tovar, CISTER Research Centre and ISEP/IPP, Porto, Portugal, emt@isep.ipp.pt.

0004-5411/2019/8-ART111 $15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Mixed-criticality systems are an important class of real-time embedded systems. Their defining characteristic is that computing tasks of different criticalities execute on the same hardware and share system resources, typically to reduce system costs. A task's criticality is a measure of the severity of the consequences of a task failing, which in the context of real-time scheduling means missing its deadline. The higher the criticality is, the more conservative, and costlier, in terms of effort, time and money, the approach employed to upper-bound that task's worst-case execution time (WCET).

Crucially for the certifiability of a mixed-criticality system, sufficient isolation must exist by design between the timing behavior of different applications. Namely, the timing behavior of one application must not possibly compromise the timeliness of a different one, especially if the former has low criticality and the latter has high criticality [31]. If tasks of different criticalities share the same resources, even on a uniprocessor platform, which is the focus of this work, they must all be engineered to the same strict standard of safety as the highest-criticality task among them [31]. This is evidently resource-inefficient and such over-engineering can have severe real-world costs. Most mixed-criticality systems are embedded, in domains like automotive, aerospace and avionics, where end-product profit margins are often thin and/or engineering constraints on size, weight and power (SWaP) can be tight.

Fortunately, the relevant guidelines (e.g., [1] [2] for the avionics domain), do not insist on zero interference among mixed-criticality applications, but instead expect any intra-application interference to be carefully accounted for and adequately mitigated. Two scheduling arrangements that can be useful to a designer faced with the above considerations, and trying to achieve both safety and efficiency, are *servers* and adaptive *mode-based* scheduling.

Servers provide scheduling isolation via *time partitioning*. In the simplest arrangement, a server is a periodically repeating fixed-length contiguous time window on a given core. Only the tasks assigned to the particular server are allowed to use the core within the confines of that time window, scheduled under, e.g., fixed priorities or EDF. Conversely, tasks cannot execute at all outside the confines of their respective server's time window. This arrangement provides a predictable supply of processing time to the set of tasks served *and* also ensures that they cannot interfere with other applications (tasks served by other servers).

Meanwhile, mode-based scheduling [5] based on Vestal's model [35] can more efficiently use the available processing capacity. This model, popular in academia, is already seeing industrial adoption for safety-critical systems, such as avionics engine control [26]. Under this model, rather than using extremely pessimistic worst-case execution time estimates for both low- and high-criticality tasks that interact, less pessimistic estimates are used, by default – *probably*, but not *provably* safe. In the statistically unlikely case of a task exceeding its assumed WCET, a carefully managed *mode change* is triggered. Less critical (and/or less important [11]) tasks, as specified at design time, are dispensed with. The remaining tasks must then be provably schedulable, assuming more pessimistic WCET estimates. In the general case, there can be many such mode changes. A mode switch is *not* a failure – it constitutes system behavior explicitly accounted for at design time (including the set of tasks to drop, and the implications of doing so). This adaptive approach improves resource efficiency without compromising safety.

Servers and Vestal's mode-based model can be combined. In this work, we consider a time-partitioned system, with multiple servers that share the same period. To each server, one or more mixed-criticality applications are assigned, in turn consisting of multiple tasks. Every server is scheduled using fixed-priority scheduling, known as Adaptive Mixed Criticality, or AMC [6], in the context of Vestal's model. The use of Vestal's model can reduce the processing budget requirements

for the different servers (compared to a naïve approach that would always assume pessimistic WCETs for all tasks) and the use of servers can provide timing isolation between applications assigned to different servers. However, it can still be inefficient if the same execution time budgets are used for the servers in all modes, because **a given server can have very different processing needs in different modes**. This realisation motivates the present work, which considers a server- and AMC-based scheduling arrangement whereby the server budgets are dynamically adjusted, at mode change, for greater resource efficiency, at no detriment to the predictability of the system or to safety. Such an arrangement requires new analysis, because even if the original analysis for AMC can be applied to periodic server-based scheduling with minor changes[1], this is no longer the case when the server budgets change in response to a mode change. This is a short-coming of the state-of-the-art. By proposing varying server budgets and analysis for this arrangement, our work hence promotes greater resource efficiency and cost savings.

Our main contributions are the following:

(1) First, we formulate a new schedulability analysis for uniprocessor systems using periodic servers with an AMC scheduling policy and whose execution time budgets are dynamically adjusted in response to a mode change. Periodic servers are a standard scheduling arrangement; what is novel is the combined use of AMC scheduling and server budget adjustment at mode change.
(2) Secondly, we discuss the complex interdependencies between the parameters of different servers and propose heuristics for the ordering of servers in the schedule and the assignment of server budgets in the different modes, for good schedulability performance.
(3) Thirdly, we explore via experiments with synthetic task sets, the schedulability performance of dynamic server budgets under different server orderings and budget assignment heuristics, compared to the baseline of static-budget servers. The results strongly validate our approach, with up to 52.8% higher scheduling success ratios.

2 RELATED WORK

Several existing works try to combine an adaptive mixed-criticality task scheduling model with servers. We note the works by Ren et al. [30], Hu et al. [19, 20], Gu et al. [15, 16], Lipari and Buttazzo [27], Awan et al. [3], Evripidou and Burns [14] and Missimer et al. [28]. Most of these approaches try to provide scheduling guarantees to high-criticality tasks with minimal impact on the quality-of-service for low-criticality tasks, e.g., by postponing mode changes or via slack reclamation. Fei et al. [17] and Papadopoulos et al. [29] adjust server budgets recurrently, based on a predictor of future execution times or automatic control theory, respectively. Unlike most of the above works that use separate servers for tasks of different criticalities, one of the approaches in [3] allows servers to serve mixed-criticality task sets. However, the server budgets are static in all modes. Meanwhile, recent studies [13, 18, 21, 24] also explore implementation-level aspects of mixed-criticality scheduling approaches (including hierarchical ones) and how to combine adaptive mixed-criticality scheduling with predictable hardware.

This work advances the state-of-the by proposing a design arrangement with periodic servers with fully adjustable server budgets at mode change, and schedulability analysis that accounts for that. A uniprocessor platform is targeted. Each server serves tasks of mixed criticalities using fixed priorities (AMC). At mode change, the low-criticality tasks are still discarded and the server budgets are adjusted (some upwards, others downwards) to account for the different processing needs in the new mode. This improves the efficiency in the use of processing capacity, allowing more demanding task sets to be schedulable, without using a faster processor.

[1]Namely, a “fake" interfering task modelling the periodic unavailability of the server.

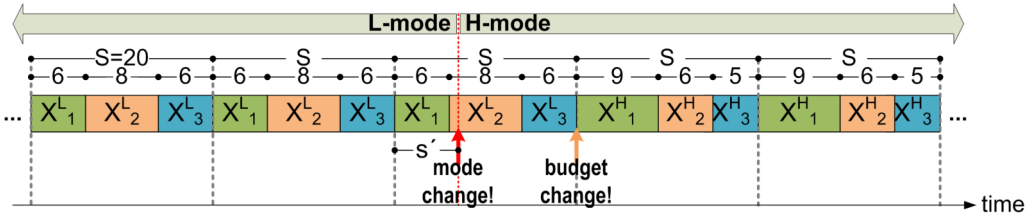


Fig. 1. At mode change, the L-tasks are dropped and the remaining H-tasks must be schedulable as long as they execute for up to their C_i^H (incl. any jobs thereof caught up in the mode change). However, server budgets are only adjusted from X_q^L to X_q^H at the start of the next timeslot.

Our approach considers the use of the Simulated Annealing metaheuristic [25] for assigning dynamic execution budgets to the servers. Simulated Annealing has already been used before for the problem of execution budget assignment both inside and outside the mixed-criticality context. Tamas and Pop [34] considered a problem of mapping mixed-criticality applications on a heterogeneous multicore platform. Applications are only allowed to run in separate partitions to achieve sufficient temporal and spatial isolation. Unlike our work, where a single kind of server is used for mixed-criticality workloads, in [34], non-preemptive static-cyclic scheduling is used to schedule critical applications while preemptive fixed-priority scheduling is used for non-critical applications. Given the task-to-core mapping and size of major frame, the devised simulated annealing meta-heuristic computes the time-slices for each partition on each core.

Beckert and Ernst [8] considered a similar server arrangement as the one assumed in this paper. It consists of a repeating TDMA schedule for fixed execution budget and fixed execution offset servers, all sharing the same period, as commonly done in ARINC653-compliant systems [2], and using fixed-priority scheduling. Our approach differs in that we use the Vestal task-model, and server budgets and offsets are adjusted after a mode change. On the other hand, Beckert and Ernst [8] incorporate scheduler and interrupt overheads into the schedulability analysis and consider them during the optimisation. This can also be integrated to our work analogously to Sousa et al. [32] and Souto et al. [33]. In more recent work of Beckert and Ernst [9], a different scheduling arrangement with sporadic servers is considered, for improved response time, which also supports background execution for tasks whose budgets are depleted.

3 TASK MODEL AND SYSTEM MODEL

3.1 Task model

This paper assumes the established adaptive variant of Vestal's mixed-criticality model, with execution time monitoring and mode changes [6]. We assume a set $\tau \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_n\}$ of n mixed-criticality sporadic tasks. Each task has a minimum interarrival time T_i, a constrained relative deadline $D_i \leq T_i$ and a criticality level κ_i. In the general case, these tasks may be grouped together into disjoint applications, possibly consisting of tasks of different criticalities.

At run-time, the system operation is based on different *modes* wherein only tasks of a given criticality or higher execute. For each task, different execution time estimates are assumed with corresponding confidence in their safety. For simplicity, in this paper, we consider only two criticality levels, high (H) and low (L), hence two modes of operation (L-mode and H-mode). The H-mode worst-case execution time estimate (H-WCET) of a task τ_i is denoted by C_i^H and it is demonstrably unexceedable, but typically very pessimistic. The L-mode worst-case execution time estimate (L-WCET) is denoted as $C_i^L \leq C_i^H$. The system boots in L-mode, wherein all tasks execute and

all their deadlines must be met. If any task attempts to execute for more than its execution time estimate for that mode, a *mode change* is triggered, whereupon all low-criticality tasks (L-tasks) are dispensed with. In H-mode, only the H-tasks execute and the deadlines of all jobs by H-tasks must be met, including any jobs released before the mode change.

In this work, we consider scheduling based on fixed priorities. This implies that each task has a unique static priority assigned to it, which is the basis of scheduling decisions. Fixed-priority scheduling, in the context of the above mixed-criticality model is known as AMC. However, in our work, we assume that tasks are partitioned to *servers* that use AMC as their internal scheduling policy. We next introduce the server model that we assume.

3.2 Server-based system model

Consider a uniprocessor platform and a set of periodic servers $\{\tilde{P}_q\}$, $q = 1, 2, \ldots, Q$ assigned to it. All servers share the same period S (also called the "timeslot length") and they execute one after the other, in a form of cyclic executive with a periodicity of S. The order in which the servers execute is fixed and specified at design time. Each server $\tilde{P}_q$ is assigned a mixed-criticality set of tasks $\tau[\tilde{P}_q] \subseteq \tau$, which are scheduled within the server according to their fixed priorities.

Each server $\tilde{P}_q$ has a fixed time budget X_q^L in the L-mode and a fixed time budget X_q^H for the H-mode, and corresponding fixed starting offsets, $O_q^L \leq S$ and $O_q^H \leq S$. Additionally, $\sum_{q=1}^{Q} X_q^L \leq S$ and $\sum_{q=1}^{Q} X_q^H \leq S$, i.e., in each mode, the servers are sized such that they fit into the timeslot S. When the system is in L-mode, all servers execute with their X_q^L budgets and all tasks present in the L-mode must provably meet their deadlines under those budgets, as long as no job executes for more than its C_i^L. However, if such an execution overrun occurs, a transition to H-mode is triggered. Then, all L-tasks are *immediately* dispensed with. The remaining tasks (including any jobs thereof released before the mode change) must meet their deadlines, assuming they can execute for up to their C_i^H estimates. Additionally, the server budgets and starting offsets are adjusted to their respective X_q^H and O_q^Hvalues *at the start of the next timeslot*. In other words, if a mode change occurs s' time units ($0 \leq s' < S$) after the last timeslot boundary, then for the next $S - s'$ time units the system is already in H-mode, but the servers' budgets and offsets are not yet adjusted from the values (X_q^L and O_q^L) used in the L-mode. Figure 1 illustrates this.

4 SCHEDULABILITY ANALYSIS

4.1 Schedulability analysis of an individual server

In this section, we derive a sufficient schedulability test for a server conforming to the model described in Section 3. Given as input the tasks assigned to a server $\tilde{P}_i$, its period S and its execution time budgets (X_i^L and X_i^H) and starting offsets (O_i^L and O_i^H) for the two modes, our analysis tests whether the server is mixed-criticality-schedulable. The question of how these attributes (X_i^L, X_i^H, O_i^L and O_i^H) are determined for each server and how the derivations of these attributes of different servers inter-depend is discussed later in Section 4.2, where the schedulability test for the entire system is formulated. Our analysis builds on AMC-max [6] and tests the schedulability of a task (i) in L-mode and (ii) in H-mode separately.

4.1.1 Steady L-mode analysis. In L-mode (i.e., prior to the occurence of a mode switch), tasks behave as conventional Liu-and-Layland tasks with a WCET of C_i^L. Therefore, as in AMC-max, to test the schedulability of a task τ_i in L-mode, we use the standard worst-case response time (WCRT) recurrence [23]. However, as in [32] and [3], we add a "fake" top-priority periodic interfering task τ_f that equivalently models the fact that the tasks do not execute directly on the processor, but within a periodic server:

$$R_i^L = C_i^L + \sum_{\tau_j \in hp(i)} \left\lceil \frac{R_i^L}{T_j} \right\rceil C_j^L + \underbrace{\left\lceil \frac{R_i^L}{S} \right\rceil (S - X^L)}_{\text{fake task}} \quad (1)$$

In (1), we omit the server index for clarity; $hp(i)$ is the set of higher-priority tasks in the same server as τ_i. The fake task's WCET is $(S - X^L)$ (equal to the time interval between the ending of one server instance and the start of the next instance of the same server); its interarrival time is S. A server $\tilde{P}_q$ is schedulable in L-mode if all of its tasks are schedulable in that mode (i.e., if $R_i^L \leq D_i \; \forall \; i \; \in \tau[\tilde{P}_q]$).

4.1.2 Schedulability testing in H-mode. To test for the schedulability of an H-task in the event of a mode change, which may also entail a server budget change (not necessarily coincident), we must consider four mutually exclusive and jointly exhaustive cases (elaborated below). The task must meet its deadline in all those cases. The reason for considering four separate cases is this: If the server budgets change after a mode change (not necessarily immediately), then the worst-case scenario, maximising the response time of a job completing in the H-mode, does not necessarily involve that job being released before the mode change, as under classic AMC.

Let $t_{hp(i)}^{idle}$ denote the first instant after the mode change that the server is active and no task in $hp(i)$ is executing inside it. Note that $t_{hp(i)}^{idle}$ might be located before the server budget change instant b or after it, or may coincide with it. The four cases to consider are:

- **Case 1:** The H-task under consideration is released before or at the mode change instant s, but completes after s.
- **Case 2:** The H-task is released after s, but before $t_{hp(i)}^{idle}$.
- **Case 3:** The H-task is released at or after $t_{hp(i)}^{idle}$ **and** before the server budget change instant b.
- **Case 4:** The H-task is released at or after $t_{hp(i)}^{idle}$ **and** at or after the budget change instant b.

Figure 2 depicts the possible relative orderings between the mode change instant (s), the budget change instant (b), the arrival time (a_i) of the H-task under analysis and $t_{hp(i)}^{idle}$, for the four cases. To analyse any of these cases, we need to upper-bound the amount of time that the server is unavailable, during any time interval that starts at the mode change instant s, as a function of the interval length Δt. We call this *the workload of the fake task in H-mode*. Lemma 4.1 provides that.

Lemma 4.1. *The workload of the fake task modelling the unavailability of a given server, in any time interval $[s, s + \Delta t)$, where s is the mode change instant and $\Delta t > 0$, is maximal if the mode change occurs either (i) at a timeslot boundary or (ii) at the end of the server's execution, i.e., $O^L + X^L$ time units after the last timeslot boundary, where O^L is the starting offset of the server in L-mode and X^L is its L-mode budget.*

Proof. Assume the mode change occurs some $s' \in [0, S)$ time units after the last timeslot boundary. Let $W_f^{H|tr.}(s', \Delta t)$ denote the workload function of the fake task over interval $[s, s + \Delta t)$ as a function of s'. For a given s' this function is *deterministic*, but the actual value of s' can only be known at run-time. To prove the lemma it therefore suffices to show that

$$W_f^{H|tr.}(0, \Delta t) \geq W_f^{H|tr.}(s', \Delta t), \; \forall \Delta t > 0, \; \forall s' \in [0, O^L)$$

and

$$W_f^{H|tr.}(O^L + X^L, \Delta t) \geq W_f^{H|tr.}(s', \Delta t), \; \forall \Delta t > 0, \; \forall s' \in [O^L, S)$$

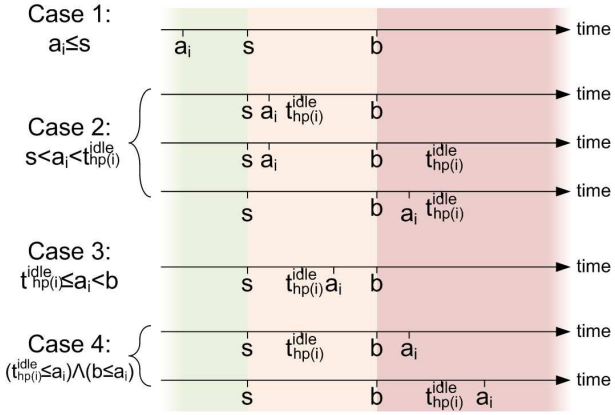


Fig. 2. All possible relative orderings between the mode change instant (s), budget change instant (b), and arrival time (a_i) of the H-task under analysis and $t^{idle}_{hp(i)}$, for 4 cases.

For visual reference, see Figure 3, which shows the workload functions of the fake task for relevant values of s'. The lines at the bottom, under each of the two graphs, represent the schedules of the server under analysis, and show the relevant values of s'. The server is scheduled during the boxes, and the label inside each box is the respective server budget. The fake task "executes" in the intervals when the server is not scheduled. The mode change occurs in the first period.

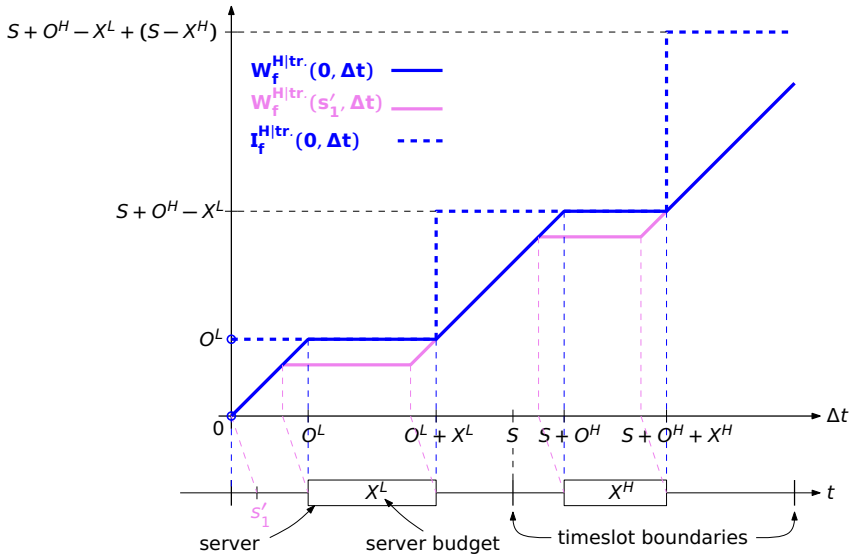
The first value ($s' = 0$) involves a mode change coincident with a timeslot boundary; the server is denied the processor for the next O^L time units (i.e., until its starting offset). The fake task's workload for $s' = 0$, $W_f^{H|tr\cdot}(0, \Delta t)$, is plotted in blue in the graph of Figure 3(a). For any $s' > 0$, up to the value of O^L, the fake task's workload would never be larger, as illustrated by the pink line in the same graph.

The other value to consider ($s' = O^L + X^L$), corresponds to the mode change occurring just as the server has run out of budget. The fake task's workload for $s' = O^L + X^L$, $W_f^{H|tr\cdot}(O^L + X^L, \Delta t)$, is plotted in blue in the graph of Figure 3(b). Smaller values of s' (i.e., $O^L \leq s' < O^L + X^L$) would mean that the server is executing immediately after the mode change and, as illustrated by the pink line in the same graph, the workload of the fake task would never be larger. Greater values ($O^L + X^L < s' < S$) would only decrease the amount of time (i.e., $S - O^L - X^L + O^H$) until the server gets to execute for the first time after the mode change, leading to a workload function that is related to the workload for $s' = O^L + X^L$, as the workload function for $s' < O^L$ is related to the workload for $s' = 0$ (see Figure 3(a)). □

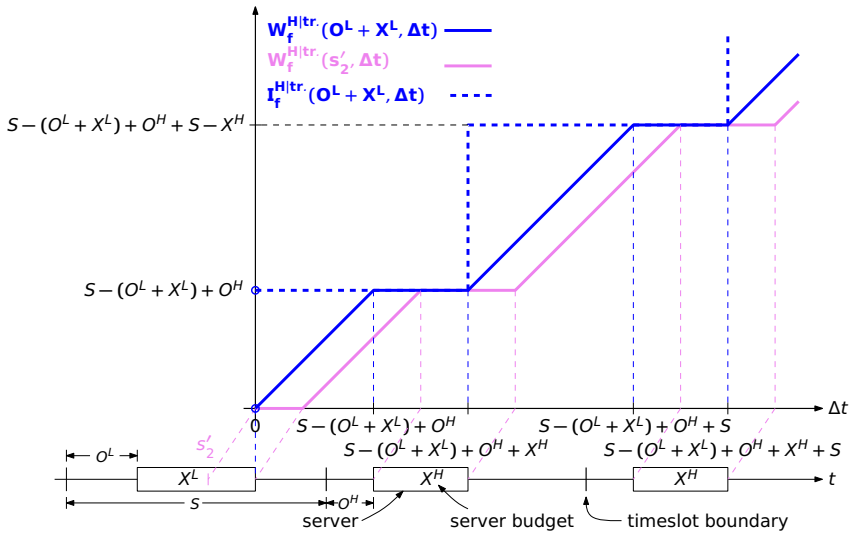
Lemma 4.2. *Consider a server. The processor request of the fake task, modelling the unavailability of the processor, over any time interval* $[s, s + \Delta t)$*, where* s *is the mode change instant and* $\Delta t > 0$ *is upper-bounded by*

$$I_f^{H|tr\cdot}(\Delta t) = \max\left(I_f^{H|tr\cdot}(0, \Delta t), I_f^{H|tr\cdot}(O^L + X^L, \Delta t)\right) \tag{2}$$

where



(a)



(b)

Fig. 3. Workload (thicker solid lines) and processor request (thicker dashed lines) after the mode change of the fake task for different values of s', $0 \leq s' < S$, the offset of a mode change with respect to the beginning of the timeslot. The lines at the bottom represent the schedule of the server under analysis. (a) Illustrates case 1 of Lemma 4.1, that the workload of the fake task for $s' = 0$, $W_f^{H|tr.}(0, \Delta t)$, plotted in blue, dominates the workload for all other $s' \in [0, O^L)$. The pink line shows the workload for a particular such value, s'_1, $W_f^{H|tr.}(s'_1, \Delta t)$. (b) Illustrates one of two subcases of case 2 of Lemma 4.1, that the workload of the fake task for $s' = O^L + X^L$, $W_f^{H|tr.}(O^L + X^L, \Delta t)$, plotted in blue, dominates the workload for all other $s' \in [O^L, S)$. The pink line shows the workload for a particular such value, $s'_2 \in [O^L, O^L + X^L)$, $W_f^{H|tr.}(s'_2, \Delta t)$.

$$I_f^{H|tr\cdot}(0,\Delta t) = O^L + \min\left(1, \left\lceil \frac{\Delta t - O^L - X^L}{S} \right\rceil_0\right)(S - X^L - O^L + O^H) + \left\lceil \frac{\Delta t - S - O^H - X^H}{S} \right\rceil_0 (S - X^H) \quad (3)$$

$$I_f^{H|tr\cdot}(O^L + X^L, \Delta t) = (S - X^L - O^L + O^H) + \left\lfloor \frac{\Delta t - S + O^L - O^H + X^L - X^H}{S} \right\rfloor_0 (S - X^H) \quad (4)$$

where, $\lfloor x \rfloor_0 = \max(0, \lfloor x \rfloor)$ *and* $\lceil x \rceil_0 = \max(0, \lceil x \rceil)$.

Proof. Equations (3) and (4), describe the processor request of the fake task if the mode change occurs s'=0 or s'=O^L+X^L time units after a timeslot boundary – the two values of s' one of which (by Lemma 4.1) maximises the fake task's workload. The maximum of the two therefore upper-bounds that request over $[s, s + \Delta t)$, $\forall \Delta t > 0$. Equation (2) follows directly from Lemma 4.1, through the relation between processor request function and workload function. Figures 3 a) and b) show (3) and (4), respectively, as thick dashed blue lines. $I_f^{H|tr\cdot}(0, \Delta t)$ is initially O^L, increases once by $(S - O^L - X^L + O^H)$ time units at $\Delta t = O^L + X^L$ and thenceforth periodically by $(S - X^H)$ time units at every $\Delta t = kS + O^H + X^H$, $k = 1, 2, 3, \ldots$. Likewise, $I_f^{H|tr\cdot}(O^L + X^L, \Delta t)$ is initially $S - O^L - X^L + O^H$, increasing by $(S - X^H)$ at every Δt=kS−O^L−X^L+O^H+X^H, k=1,2,. . .. □

Now, we can analyse the 4 cases (Figure 2) one by one:

Case 1: This case considers H-tasks that are caught in their execution window by the mode switch, and hence may suffer the interference both from L-tasks and H-tasks of higher priority. The interference from such tasks can be upper-bounded according to the existing AMC-max analysis. However, the task in consideration also suffers interference from the unavailability of the server (which we model as a fake task). Let us therefore incorporate $I_f^{H|tr\cdot}(\Delta t)$ into a hybrid AMC-max schedulability test for this case. Namely, we can upper-bound the response time of an H-task τ_i, released at or before the mode change instant s but not yet completed by s, as

$$R_i^{H|1} = \max(R_i^{s|1}),\ \forall s \in [0, R_i^L) \quad (5)$$

where

$$R_i^{s|1} = C_i^H + \sum_{\tau_j \in hpL(i)} IL_j(s) + \sum_{\tau_k \in hpH(i)} IH_k(s, R_i^{s|1}) + \underbrace{IL_f(s) + I_f^{H|tr\cdot}(R_i^{s|1} - s)}_{fake\ task} \quad (6)$$

where $hpL(i)$ and $hpH(i)$ are the sets of higher-priority low- and high-criticality tasks in the same server, respectively, and

$$IL_j(s) = \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1\right) C_j^L \quad (7)$$

$$IL_f(s) = \left(\left\lfloor \frac{s}{S} \right\rfloor + 1\right)(S - X^L) \quad (8)$$

$$IH_k(s,t) = M(k,s,t)C_k^H + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t)\right) C_k^L \quad (9)$$

where, as in [6], $M(k,s,t) = \min\left(\left\lceil \frac{t-s-T_k-D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil\right)$.

By replacing IH_k in (6) with the RHS of (9), we obtain:

$$\begin{aligned} R_i^{s|1} =& C_i^H + \sum_{\tau_j \in hpL(i)} IL_j(s) \\ &+ \sum_{\tau_k \in hpH(i)} \left(M(k,s,R_i^{s|1}) C_k^H + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,R_i^{s|1}) \right) C_k^L \right) \\ &+ IL_f(s) + I_f^{H|tr} \cdot (R_i^{s|1} - s) \end{aligned}$$

Splitting the second summation and reordering the terms:

$$\begin{aligned} R_i^{s|1} =& C_i^H + \sum_{\tau_j \in hpL(i)} IL_j(s) + IL_f(s) \\ &+ \sum_{\tau_k \in hpH(i)} \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,R_i^{s|1}) \right) C_k^L \\ &+ \sum_{\tau_k \in hpH(i)} M(k,s,R_i^{s|1}) C_k^H + I_f^{H|tr} \cdot (R_i^{s|1} - s) \end{aligned}$$

Finally, in a slight accuracy optimisation of ours,

$$R_i^{s|1} = C_i^H + \left[\!\left[IL(s, R_i^{s|1}) \right]\!\right]^s + IH(s, R_i^{s|1}) \tag{10}$$

where:

$$\begin{aligned} IL(s, R_i^{s|1}) &= \sum_{\tau_j \in hpL(i)} IL_j(s) + IL_f(s) \\ &\quad + \sum_{\tau_k \in hpH(i)} \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,R_i^{s|1}) \right) C_k^L \\ IH(s, R_i^{s|1}) &= \sum_{\tau_k \in hpH(i)} M(k,s,R_i^{s|1}) C_k^H + I_f^{H|tr} \cdot (R_i^{s|1} - s) \end{aligned}$$

and the operator $[\![\cdot]\!]^{\max}$ is defined as $[\![x]\!]^{\max} \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \le \max \\ \max & \text{if } x > \max \end{cases}$.

Under the original AMC-max, the operator $[\![\cdot]\!]^s$ is not used. Our slight improvement acknowledges that the interference from all jobs completed before the mode change cannot exceed s. [2]

Case 2: In this case, the H-task τ_i under analysis is released at some instant a_i, after the mode change instant s, but before $t_{hp(i)}^{idle}$.

Since τ_i is not released before the mode change, it does not suffer any *direct* interference from jobs of L-tasks. However, in the general case, it may suffer indirect *push-through* interference by such tasks that executed before its release. By this, we mean that any higher-priority H-task jobs released before the mode change instant s and not completed before time a_i (the release of τ_i) may have suffered interference from L-tasks of even higher priority (if any), before the mode change. This would comensurately push their execution to the right, along the time axis. In any case, we do not need to quantify the push-through interference from L-tasks in Case 2 (or even test

[2]Not enclosing the expression by $[\![\cdot]\!]^s$, would mean the analysis holds even for a variant model that allows any L-jobs caught up in the mode change to complete, executing for up to their L-WCETs. This follows from the original AMC-max [6].

schedulability in Case 2 at all!), because, as we will prove below, if τ_i is proven to be schedulable in the L-mode and in Case 1, it will always also be schedulable in Case 2.

Lemma 4.3. *If an H-task τ_i is schedulable in L-mode and schedulable in H-mode under the assumptions of Case 1, then it is also schedulable in H-mode under the assumptions of Case 2.*

Proof. Consider a schedule σ wherein a_i is the first time instant strictly after the mode change s that τ_i is released and it also holds that $a_i < t^{idle}_{hp(i)}$. This schedule then fulfills the assumptions of Case 2. Any immediately preceding job by τ_i will have been released no later than s, so it would fall under Case 1 or will have completed before the mode change, so in either case, it would have been schedulable; this means that the job by τ_i released at time a_i suffers no interference by previous jobs of the same task; it only suffers interference from higher-priority tasks (including the fake task).

Let us then transform this original schedule σ to another schedule σ' where, all other things remaining equal, the release of the job by τ_i under analysis is shifted earlier, to some time instant t'' (with the releases of all other jobs by τ_i also shifted earlier by the same amount), such that the following things both hold: (i) The instant t'' is located at or before the mode change (i.e., $t'' \leq s$); and (ii) The entire interval $[t'',\ a_i)$ is occupied by execution of tasks in $hp(i)$ or the fake task. The fact that in the original schedule, the entire interval $[s,\ a_i]$ was busy by higher-priority tasks (including the fake task), means that such an instant t'' exists. It could be time instant s itself, or some even earlier time instant.

Then, the response time of the job under analysis cannot decrease as a result of the schedule transformation. Namely, in the transformed schedule σ', the task τ_i does not execute at all over $[t'',\ a_i)$ (where a_i refers to its release in the original schedule σ), and from time a_i onwards, its execution intervals are the same as in the original schedule. Therefore its absolute completion time f_i is unchanged, even though its release is shifted earlier, to time $t'' < a_i$. In turn, the transformed schedule σ' belongs to Case 1, analysed earlier (i.e., τ_i being released no later than s but completing after s). Therefore, the increased response time of the job is upper-bounded by D_i, from the assumption that τ_i is schedulable in Case 1. Therefore the original response time of the job in schedule σ was also upper-bounded by D_i.

We will now show by contradiction that, if τ_i is schedulable in L-mode and in H-mode under Case 1, there cannot be more than one job of τ_i released strictly after s and strictly before $t^{idle}_{hp(i)}$. Assume that in schedule σ the next job by τ_i, after the one released at a_i, was released at time a'_i and that $a'_i < t^{idle}_{hp(i)}$. Then, the job released at time a_i does not receive any execution time at all during the interval $[a_i,\ a'_i)$, therefore it misses its deadline at time $a_i + D_i \leq a'_i$. This contradicts the fact that, as proven earlier, it is schedulable. □

Therefore, the schedulability test for Case 1 subsumes Case 2. In other words, if τ_i provably meets its deadline in L-mode and in H-mode under Case 1, then it also does so under Case 2. Since we test for Case 1 anyway, testing for Case 2 is hence redundant.

Case 3: In this case, because the H-task τ_i under analysis is released at $t^{idle}_{hp(i)}$ or later, there can be no push-through interference from L-tasks. Therefore, there is only direct interference, from the tasks in $hpH(i)$ and from the unavailability of the server (i.e., from the fake task). The worst-case, in terms of interference from tasks in $hpH(i)$, is when these are released simultaneously as τ_i, at time a_i. This is the same as the worst-case interference from those tasks when we are in Case 1 and $s = 0$.

As for the worst-case interference from the fake task, since in Case 3 the release time a_i of τ_i is before the budget change instant, it is upper-bounded by (2), as in Case 1 (by Lemma 4.2).

Combining our observations, the schedulability test for Case 3 is also subsumed by the test for Case 1.

Case 4: Since τ_i is released at $t^{idle}_{hp(i)}$ or later and also at the server budget change or later, it is not subject to any transitive effects either from the mode change or from the budget change. Therefore, to compute its WCRT, we can apply classic fixed-priority response time analysis, considering (i) the tasks present in the H-mode and their respective WCET estimates for that mode, and (ii) a fake top-priority task, modelling the unavailability of the server, with a WCET of $S - X^H$ and period of S. The corresponding equation is:

$$R_i^{H|4} = C_i^H + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^{H|4}}{T_j} \right\rceil C_j^H + \underbrace{\left\lceil \frac{R_i^{H|4}}{S} \right\rceil (S - X^H)}_{fake\ task} \tag{11}$$

Note that the worst-case processor request by the fake task in Case 4, during an interval of Δt time units, which is $\left\lceil \frac{R_i^{H|4}}{S} \right\rceil (S - X^H)$ is not necessarily upper-bounded, in the general case, by the expression $I_f^{H|tr\cdot}(\Delta t)$ (Equation (2)) that describes the request by the fake task in Cases 1, 2 and 3 (i.e., when τ_i is released before the budget change). Therefore, the schedulability test for Case 4 is not dominated by the schedulability test for Case 1; we must test for Case 4 separately.

4.2 Schedulability analysis at the system level

We can now explain how the schedulability of the entire system is tested and how the assignments of execution budgets and starting offsets for the different servers in the two modes interdepend. A system is schedulable if all servers are assigned non-overlapping execution windows inside the timeslot in both modes and if they are all found schedulable by the schedulability test from Section 4.1 with the assigned execution budgets and starting offsets.

If the servers appear in the same order inside the timeslot in both modes (as we already assume) and their time windows are arranged back-to-back, with the first server aligned with the start of the timeslot, then (if, without loss of generality, the servers are indexed from left to right), we have

$$O_1^L = O_1^H = 0;\ O_i^L = O_{i-1}^L + X_{i-1}^L;\ O_i^H = O_{i-1}^H + X_{i-1}^H,\ \forall i > 1 \tag{12}$$

Then, the schedulability condition can be expressed as

$$\forall\ i:\ \left(\tilde{P}_i \text{ schedulable with } (X_i^L, X_i^H, O_i^L, O_i^H)\right) \\ \wedge\left(\sum X_i^L \le S\right) \wedge \left(\sum X_i^H \le S\right)$$

Interdependencies between the parameters of different servers. From (12), one can see that for a given ordering (indexing) of the servers, the execution budgets of preceding servers in one mode determine the starting offset of a given server in that mode. In turn, these offsets (O_i^L and O_i^H) are inputs (along with the budgets X_i^L and X_i^H) to the schedulability test for that server $\tilde{P}_i$. Therefore the execution budgets of all preceding servers indirectly affect whether or not a server $\tilde{P}_i$ is schedulable with a given budget pair (X_i^L, X_i^H). The ordering of the servers within the timeslot thus matters a lot.

Intuitively, one would expect that ordering the servers such that they appear in the timeslot by non-decreasing $X_i^H - X_i^L$ would be a helpful heuristic for achieving good scheduling performance. The reasoning is that if the servers appear in order of $X_i^H - X_i^L$ in the timeslot and the timeslot is fully utilised in both modes, then $O_i^L - O_i^H \ge 0$ for all servers – and, in the timeslot where a mode change occurs, this "benign" jitter would mean that the interval between a server completing

with a budget of X_i^L and the start of the execution of the next server instance, with budget X_i^H, would be S or (in most cases) smaller than S. This implies a shorter effective transition time to the new budgets (greater responsiveness to the new processing requirements), compared to the case of $O_i^L - O_i^H < 0$. However, in the general case, we cannot know a priori in which order to arrange the servers such that they appear in order of $X_i^H - X_i^L$, because the budgets can only be computed a posteriori, using the offsets O_i^L and O_i^H as input, which themselves depend on the server ordering, as we just explained earlier.

Additionally, in the general case, and for a given pair of starting offsets (O_i^L, O_i^H) there may exist multiple budget pairs (X_i^L, X_i^H) for which a server is schedulable. If sensitivity analysis (e.g., binary search) is used to determine the least feasible budget X_i^H for a given offset pair (O_i^L, O_i^H) as a function of X_i^L, then the pair (X_i^L, X_i^H) will exhibit the Pareto property. Namely, a more generous L-mode budget X_i^L might require a smaller X_i^H budget for the L-mode (as, intuitively, the server will have comparatively less "catching up" to do with the tasks' demand) – and vice versa.

All these different interdependencies between the parameters of different servers and their ordering, complicate the task of devising good heuristics for ordering the servers inside the timeslot and assigning budgets to them for the two modes. However, as we will show in the next section, it is still possible to leverage them in a useful way, and attain good performance.

5 BUDGET ASSIGNMENT HEURISTICS

We consider two scheduling arrangements (static and dynamic server budgets), with different budget assignment heuristics.

5.1 Static server budgets (SSB)

Under this arrangement, the execution budget of a server is the same in both modes (i.e., $X_i^L{=}X_i^H{=}X_i\ \forall i$). Additionally, the first server is positioned at the beginning of timeslot ($O_1^L{=}O_1^H{=}0$) and every subsequent server starts when its predecessor ends. These properties imply that $O_i^L{=}O_i^H\ \forall i$, i.e, neither the starting offset nor the execution budget of any server ever changes. Consequently, the analysis of Section 4 is not needed. Rather, we can use the original AMC-max test [6] to size each server, with the addition of a top-priority fake H-task τ_f with $C_f^L{=}C_f^H{=}S - X_i$ and $T_f{=}S$. The minimum feasible budget X_i for a given server can be found via binary search over $(0, S]$. Note that for this arrangement, each server can be sized independently of other servers and their attributes and the order in which they are arranged in the timeslot is irrelevant.

Such independence between servers also holds when our analysis is used instead of the original AMC-max, to test the feasibility while sizing servers with static budgets. Indeed, the offsets of the server under analysis (and all others) can be disregarded because any static-budget server execution pattern is transformable via shift-rotation along the time axis into an equivalent schedule where the server under consideration has a given offset (conveniently, $O_i^L = O_i^H = 0$). Crucially, the unavailability intervals for the server all have a duration of $S - X_i$ and occur strictly periodically.

Nevertheless, even if our new analysis can accommodate static server budgets as a special case, it does not necessarily outperform AMC-max for this arrangement because of the slight pessimism introduced by upper-bounding the interferences from the unavailability of the server in L-mode and in H-mode separately from each other (see Equation 6). In any case, for greater insight, in Section 6, we plot results for SSB using both analyses.

5.2 Dynamic server budgets (DSB)

Under this arrangement, which our analysis was devised for, *a server can have a different budget after a mode switch.* Once again, the servers execute back-to-back and the first server is aligned

with the start of the timeslot. Both the L-mode and H-mode server budgets are computed offline. To compute the minimum feasible H-mode budget X_i^H of a server, we need to know its starting offsets in each mode (O_i^L and O_i^H) and its L-mode budget X_i^L. In turn, the offset of a given server in a given mode can only be computed if we already know the budgets of all predecessor servers. This implies that the order in which the servers are arranged in the schedule is already decided. This is one of the reasons why the particular budget assignment problem is, in our view, not amenable to a tractable Integer Linear Programming (ILP). See related discussion in Section 5.3. Therefore, we devised heuristics for selecting the server order (described in Section 5.2.1) and for server budget assignment.

For the DSB arrangement, we explore two different heuristics. First, we consider a simple heuristic that assigns L-mode budgets (X_i^L) to all servers, proportionally to the minimum feasible L-mode budget X_i^{min} for each server (identified with a binary search algorithm [32]). All server offsets and H-mode budgets are eventually computed from that set of L-mode budgets, directly or indirectly. As a second option, we explore the *Simulated Annealing* (SA) [25] metaheuristic, which accepts the output of the previous heuristic as a starting solution (if not already feasible), and tries to mutate the original set of X_i^L budgets until it becomes feasible.

SA attempts to replace the current solution of a problem with a new (randomly obtained) solution in each iteration. A candidate solution that improves on the current one is always accepted but, occasionally, the algorithm also accepts a "worse" candidate solution, with a probability that depends on the value of a probability function. This function takes as parameters a variable Θ (the "temperature") and the difference of the "utilities" of the current solution and the candidate solution. Higher temperatures and lower reduction in utility raise the acceptance probability for a "worse" solution. Occasionally accepting "worse" solutions prevents getting stuck at a local optimum of the optimization problem. The temperature Θ is gradually decreased with the number of iterations. In our implementation, a solution is represented by the set of X_i^L values (which uniquely determines all eventual O_i^L and O_i^H offsets and X_i^H budgets). As utility of a given solution, we define the sum of the X_i^H budgets calculated separately for each server, assuming the corresponding X_i^L value and $O_i^L = O_i^H = 0$.

Algorithm 1 provides detailed pseudocode for both heuristics.

Initial Phase (Simple heuristic): Initially, we determine the minimum feasible L-mode budget X_i^{min} for each server separately (line 1). To do that, we assume that its H-mode budget is equal to S (the entire timeslot) and its starting offsets are zero (i.e., $X_i^H = S$ and $O_i^L = O_i^H = 0$). With these assumptions, we compute, using our new analysis as feasibility test, the corresponding minimum feasible L-mode size X_i^{min} for each server using binary search [32]. If the sum of X_i^{min} for all servers exceeds S or if any of the servers are infeasible with the maximal H-mode server size of S, we declare failure as the system is provably unschedulable, with any assignment of server budgets (line 2). Otherwise, once X_i^{min} is computed for all servers, we set $X_i^L = X_i^{min} * \frac{S}{\sum_{\forall i} X_i^{min}}$ for each server (line 5). The factor $\frac{S}{\sum_{\forall i} X_i^{min}}$ proportionally scales up the X_i^{min} value of each server to fill up entirely the L-budget timeslot S. So, by construction, $\sum_{\forall i} X_i^L \leq S$.

The selected L-mode budgets are in turn used to compute the actual H-mode server budgets. With a server order given a priori, the H-mode budget (X_i^H) of any i^{th} server can be computed using binary search over $[0, S]$, assuming offsets of $O_i^L = \sum_{j=1}^{i-1} X_j^L$ and $O_i^H = \sum_{j=1}^{i-1} X_j^H$ (line 6). If for the computed X_i^H values it holds that $\sum_{\forall i} X_i^H \leq S$, we declare success (line 7). Otherwise, we try the metaheuristic (Simulated Annealing), implemented in main loop:

Main Loop (Simulated Annealing): In any iteration k, two servers ($\tilde{P_a}$ and $\tilde{P_b}$) are selected randomly (line 10). This heuristic increments X_a^L of $\tilde{P_a}$ and decrements X_b^L of $\tilde{P_b}$ (line 12) by the same

Algorithm 1 Simple heuristic and Simulated Annealing algorithm

Input: Sorted $\tilde{P}$, S, Δ, Θ and Cooling Rate
Output: $X_i^L, X_i^H, \forall i$

1: **Initial Phase:**
2: Find X_i^{min} for each server $\tilde{P}_i$ using binary search, assuming $X_i^H = S$.
3: **if** $(\sum_{\forall i} X_i^{min} > S \;||\; \exists\, \tilde{P}_i$ infeasible with $X_i^H = S)$ **then return** Failure
4: **else** ▷ scale up X_i^L values
5: Set $X_i^L = X_i^{min} \times (S/\sum_{\forall i} X_i^{min}), \forall i$
6: Compute X_i^H for each server, given X_i^L values $\forall i$, with offsets
7: **if** $(\sum_{\forall i} X_i^H \leq S)$ **then return** $X_i^L, X_i^H, \forall i$
8: **Main Loop:**
9: **for** $(k = 0, \Theta > 1; k = k + 1)$ **do**
10: Select two random servers $\tilde{P}_a$ and $\tilde{P}_b$
11: Compute β for servers $\tilde{P}_a$ and $\tilde{P}_b$ with Algorithm 2
12: Set $X_a^L(k) = X_a^L(k\text{-}1) + \beta$ and $X_b^L(k) = X_b^L(k\text{-}1) - \beta$
13: Set other servers $X_i^L(k) = X_i^L(k-1), \forall i \notin a, b$
14: Compute $X_i^H(k), \forall i$ in k^{th} iteration with offsets
15: **if** $(\sum_{\forall i} X_i^H \leq S)$ **then return** $X_i^L(k), X_i^H(k), \forall i$
16: **else**
17: Compute $X_i^H(0,k), \forall i$ with $O_i^L = O_i^H = 0$ offset in iteration k
18: **if** $(\sum_{\forall i} X_i^H(0,k) < \sum_{\forall i} X_i^H(0,k-1))$ **then**
19: Keep $X_i^L(k), X_i^H(0,k), \forall i$ for $k{+}1^{th}$ iteration
20: **else if** $(z \in (0,1] \leq e^{\frac{\sum_{\forall i} X_i^H(0,k-1) < \sum_{\forall i} X_i^H(0,k)}{\Theta}})$ **then**
21: Keep $X_i^L(k), X_i^H(0,k), \forall i$ for $k{+}1^{th}$ iteration
22: **else**
23: Discard this and keep $(k\text{-}1)^{th}$ iteration solution
24: $\Theta = \Theta * (1-$ Cooling Rate$)$
25: On for loop termination without success, **return** Failure

value of β (computed via Algorithm 2), that is the server variation length parameter for this iteration. Adding and subtracting the same value of β from the two budgets keeps the sum of L-mode server budgets in the k^{th} iteration equal to that of the $(k-1)^{th}$ iteration, i.e., $\sum_{\forall i} X_i^L(k) = \sum_{\forall i} X_i^L(k-1)$.

By construction, the heuristic keeps $\sum_{\forall i} X_i^L \leq S$. Hence, the parameter β is computed by Algorithm 2 such that this condition is never violated. In Algorithm 2, initially, $\beta_{max} = -(\Delta * S) + (2 * \gamma)$ gives the maximum variation in this iteration (line 1, Algorithm 2), where $\Delta \in (0, 1]$ is an input parameter, and γ is randomly generated over $(0, \Delta * S]$. Afterwards, β_a and β_b are selected such that $X_a^L + \beta_a$ remains in $[X_a^{min}, S]$ and $X_b^L - \beta_b$ remains in $[X_b^{min}, S]$ (lines 2-13, Algorithm 2). Between β_a and β_b, the one that gives the least change in server size is selected as β (lines 14-17, Algorithm 2).

After selecting β, $X_a^L(k)$ and $X_b^L(k)$ are updated to $X_a^L(k-1) + \beta$ and $X_b^L(k-1) - \beta$, respectively. All other severs get the previous-iteration values, i.e., $X_i^L(k) = X_i^L(k-1),\ \forall i \neq a,\ i \neq b$ (line 13). Once L-server budgets are available for the k^{th} iteration, the corresponding H-mode server sizes are computed (employing binary search and our analysis as schedulability test), using the offsets $O_i^L(k) = \sum_{j=1}^{i-1} X_j^L(k)$ and $O_i^H(k) = \sum_{j=1}^{i-1} X_j^H(k)$, for any server $\tilde{P}_i$ (line 14).

Algorithm 2 Server variation length parameter (β) computation

Input: $\tilde{P_a}$, $\tilde{P_b}$, S and Δ
Output: Server size variation factor β

1: $\beta_{max} = -(\Delta * S) + (2 * \gamma)$ ▷ Parameter $\Delta \in (0, 1]$; $\gamma \sim \text{unif}(0, \Delta * S]$.
2: **if** $(X_a^L + \beta_{max} > S)$ **then**
3: $\quad \beta_a = S - X_a^L$
4: **else if** $(X_a^L + \beta_{max} < X_a^{min})$ **then**
5: $\quad \beta_a = X_a^{min} - X_a^L$
6: **else**
7: $\quad \beta_a = \beta_{max}$
8: **if** $(X_b^L - \beta_{max} > S)$ **then**
9: $\quad \beta_b = X_b^L - S$
10: **else if** $(X_b^L - \beta_{max} < X_b^{min})$ **then**
11: $\quad \beta_b = X_b^L - X_b^{min}$
12: **else**
13: $\quad \beta_b = \beta_{max}$
14: **if** $(\beta_{max} \geq 0)$ **then**
15: $\quad \beta = \min\{\beta_a, \beta_b\}$
16: **else**
17: $\quad \beta = \max\{\beta_a, \beta_b\}$
return β

If the process of computing $X_i^H(k)$ with the above offsets for the k^{th} iteration is successful and $\sum_{\forall i} X_i^H(k) \leq S$, we declare a success and exit the loop (line 15). Otherwise, the "utility" of the current solution is computed. For that purpose, we calculate for each server what its least feasible H-mode budget would be ($X_i^H(0, k)$), with the current $X_i^L(k)$ budget and assuming $O_i^L = O_i^H = 0$ (line 17). The utility of the solution is the sum of those X_i^H values. If $\sum_{\forall i} X_i^H(0, k) < \sum_{\forall i} X_i^H(0, k-1)$ (i.e., if it is a "better" solution than the previous iteration), or if a randomly generated number z in $(0, 1]$ is less than or equal to acceptance probability of $e^{\frac{\sum_{\forall i} X_i^H(0,k-1) < \sum_{\forall i} X_i^H(0,k)}{\Theta}}$ (i.e., occasionally accepting the worse solution), then the $X_i^L(k)$ and $X_i^H(0, k)$, $\forall i$ are accepted for the $(k+1)^{th}$ iteration (lines 18-21). Otherwise, these values are discarded, and $X_i^L(k-1)$ and $X_i^H(0, k-1)$, $\forall i$ are used for the $(k+1)^{th}$ iteration (line 23). Finally, if the system cools downs without finding any feasible solution, a failure is declared (line 25).

5.2.1 Server ordering heuristics. To achieve efficient dynamic server budget assignment, the order of servers is an important initial step. We propose to sort the servers in non-decreasing order of $U^H(\tilde{P_i}) - U^L(\tilde{P_i})$, where $U^H(\tilde{P_i})$ and $U^L(\tilde{P_i})$ represent the H and L-mode utilisation of server $\tilde{P_i}$, respectively. The reason we chose this ordering was because it would result in a server ordering that would approximate an ordering by non-decreasing $X_i^H - X_i^L$. Recall that in Section 4.2, we identified that ordering servers by non-decreasing $X_i^H - X_i^L$ would likely promote good performance. However one can only confirm whether a given server ordering meets this property *a posteriori*, due to the interdependencies of the servers' parameters with each other and the ordering. Therefore, we simply chose $U^H(\tilde{P_i}) - U^L(\tilde{P_i})$ (which is verifiable *a priori*) as a good proxy. We also experimented with server ordering by non-increasing $U^H(\tilde{P_i})/U^L(\tilde{P_i})$, but it performed slightly worse.

5.3 Amenability to ILP formulation

Above, we opted for heuristics and metaheuristics for solving the server budget and offset assignment problem. It is, however, worthwhile to discuss whether an ILP formulation to solve this problem could be devised and whether or not this would be worth pursuing.

As we will show, this specific problem is not particularly amenable to a tractable ILP formulation. This is related to the fact that each server has not one, but two execution time budgets (one per mode) but these also depend on the servers' starting offsets in the two modes and, indirectly, on the budgets and offsets of other servers. In more detail:

The server budget and offset assignment problem can be formulated as a 2-dimensional bin packing problem in which one of the dimensions corresponds to L-mode and the other dimension to H-mode. Each server can be represented by a rectangle, with dimensions (X^L, X^H), and the offsets O^L and O^H can be seen as the coordinates of the bottom-left corner of each rectangle, representing a server. The problem then is to place all the servers in an S by S square, with a proper orientation, such that the servers do not overlap, and the bottom-left corner is the bottom-left corner of the square, for the first server, and the upper-right corner of the server that precedes it, for the remaining servers. A similar formulation, i.e. based on a 2-dimensional bin packing problem, is used in [4] for a problem of budget assignment to servers with two different budgets per server, an execution time budget and a budget for memory accesses. In that work, to make the solution tractable, all feasible budget pairs for each server (characterised by the pareto property) are precomputed and encoded into the ILP as constants. However, in the particular problem that we are targeting in the present work, this does not appear as a viable approach, because, whether or not a budget pair (X^L, X^H) is feasible for a server also depends on the corresponding pair of starting offsets (O^L, O^H) for the server, which in turn depends on the budgets of other servers. Precomputing all feasible tuples (X^L, X^H, O^L, O^H) for each server and encoding into the ILP as constants, in order to break this dependency would be intractable, because the number of tuples would be too large.

6 EVALUATION

6.1 Experimental setup

We implemented the analysis and heuristics in Java, to explore the scheduling performance of different scheduling arrangements and budget assignment heuristics and the effectiveness of the analysis. One module generates the synthetic task sets and servers, for the given parameters. Another module does the schedulability testing.

Task set generation: Task periods are generated with a log-uniform distribution in the 10-100 msec range. We generate implicit-deadline tasks ($D_i = T_i$), even though the analysis holds for the more general constrained deadline model ($D_i \leq T_i$). The given target L-mode utilisation is distributed among tasks using UUnifast [10, 22] in an unbiased way. Then, for each task $C_i^L = T_i * U_i^L$, where U_i^L is its L-mode utilisation. The fraction of H-tasks is a user-defined parameter. The H-WCET of a task is a linearly scaled up value of its L-WCET, according to an an input parameter κ (i.e., $C_i^H = \kappa C_i^L$. Tasks are assigned Deadline Monotonic priorities. The generated tasks are indexed in order of increasing D_i and are assigned to servers using round-robin. To better utilise the system's resources, one can explore the problem of efficiently assigning tasks-to-servers. However, designers may not always have control over this, since the grouping of tasks is functional, on the basis of appplication. The timeslot length (S), corresponding to the common period of all servers, is set to the shortest T_i in the task set.

The target L-mode system utilisation varies in a range of $[0.1, 1]$, in steps of 0.1. Different random class objects are used to generate period and utilisation values. They are seeded with a different

Table 1. Overview of Parameters

Parameters	Values	Default
Number of servers (q)	$\{2, 3, 4, 5, 6\}$	3
Task-set size (n)	$\{9, 12, 15, 18, 21, 24\}$	3/server
H-tasks share	$\{20 : 10 : 80\}\%$	30%
HWCET scaling (κ)	$\{2 : 1 : 6\}$	3
Temperature (Θ)	$\{.025, .05, .1, .5, 1, 5, 10\}^*1000$	10000
Cooling rate	$\{.001, .005, .01, .05, .1, .3, .5, .7, .9, 1\}$	0.005
Budget variation (Δ)	$\{.001, .005, .01, .02, .05, .1 : .1 : .9\}$	0.5
Server ordering	$\{U^H(\tilde{P}_i) - U^L(\tilde{P}_i), U^H(\tilde{P}_i)/U^L(\tilde{P}_i)\}$	1^{st}

odd number and reused in successive replications. For each set of input parameters, we generate 1000 random task sets. Table 1 presents the range of values considered for all the parameters and their defaults. The triples in this table corresponds to {min : step : max} values of a parameter. In our experiments, we vary one parameter at a time, while the others conform to their defaults.

To save space, we present the results as plots of *weighted schedulability*. This performance metric [7, 12] condenses three-dimensional plots into two dimensions. It is a weighted average that gives more weight to task sets with higher utilisation – supposedly harder to schedule. Let $S_y(\tau, p)$ denote the result (0 or 1) of a schedulability test y for a given task set τ with an input parameter p. Then $W_y(p)$, the weighted schedulability for test y as a function of p, is $W_y(p) = \sum_{\forall \tau} \left(U(\tau) * S_y(\tau, p)\right) / \sum_{\forall \tau} U(\tau)$, where $U(\tau)$ is the utilisation of τ. Our plots are weighed according to L-mode utilisation.

6.2 Scheduling arrangements and server ordering heuristics compared

Static Server Budgets (SSB): A server has the same budget in both modes. We use the original AMC-max [6] for the schedulability test, for fairness. A server is assigned the smallest budget that ensures schedulability, using sensitivity analysis (binary search).

Static Server Budgets with our analysis (SSBO): Same as above, but instead of AMC-max, our new analysis is used with server offsets set to zero, i.e., $O_i^L = O_i^H = 0 \, \forall i$. Comparing SSBO with SSB assesses the pessimism when the new analysis deals with the special case of $X_i^L = X_i^H \, \forall i$. Such pessimism stems from independently bounding the fake task's interference in the two modes.

Dynamic Server Budgets with Simple heuristic (SH): This approach for DSB declares a success, if the system is feasible using the simple budget assignment heuristic, i.e., the "Initial Phase" of Algorithm 1, using our new analysis.

Dynamic Budgets with Simulated Annealing (SA): This corresponds to the metaheuristic (main loop of Algorithm 1). Our new analysis is used for schedulability testing. Success is plotted if a task set is either schedulable using just SH, or, if it is not schedulable by SH, but the metaheuristic in the second stage finds a feasible assignment of server budgets for both modes. Table 1 presents the ranges used for the configuration parameters (Θ, cooling and Δ).

SA with all possible server orderings (SAAO): Instead of picking a predefined ordering, all possible orderings in which the servers can be arranged are tested for a feasible solution. For each of them, the SA heuristic described above is applied, with the specifed maximum number of iterations. The SAAO heuristic allows us: (a) to analyze the effect of the server ordering, and (b) to quantify the quality of the default ordering by $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$ and, indirectly, the validity of the intuition behind its selection.

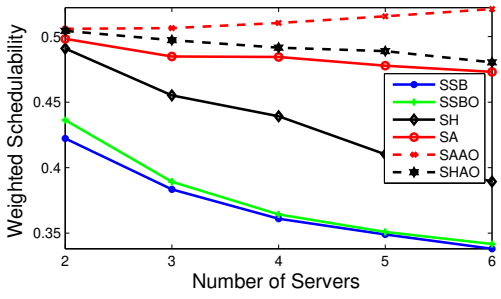


Fig. 4. (Fixed tasks per server)

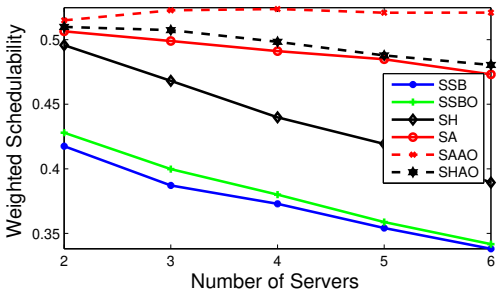


Fig. 5. (Fixed number of total tasks)

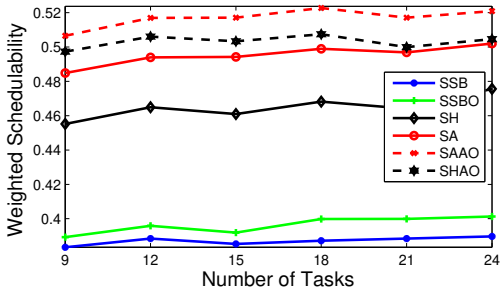


Fig. 6

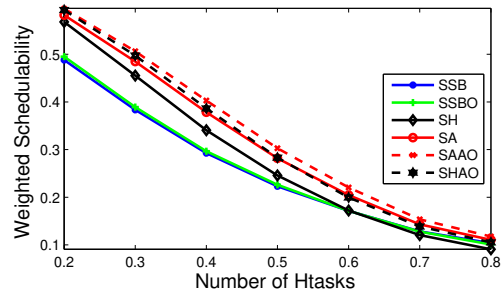


Fig. 7

SH assuming all possible server orderings (SHAO): Similar to SH, except that all possible server orderings are checked.

6.3 Results

Figures 4 and 5 present the weighted schedulability for different number of servers. The number of tasks per server is constant (3/Server) in Figure 4, while the total number of tasks to distribute among servers via a round robin policy is constant (18 tasks) in Figure 5. Increasing the server count results in lower schedulability, due to the reduced average per-server budget in a given time slot. All curves follow this trend. The difference between SAAO and SA is small which validates the selected server ordering by non-decreasing $(U^H(\tilde{P}_i) - U^L(\tilde{P}_i))$ as a good choice. At low server counts, SH performs similar to SA as the scaling of L-mode budgets to fully utilise the timeslot of length S helps find more feasible H-mode budgets, already in the initial phase. An increase in the number of servers makes it harder to fit the servers in the timeslot and hence, the main loop in SA (Algorithm 1) becomes useful. Similar behaviour is seen when comparing SHAO and SAAO. In most the cases, SHAO outperforms, indicating that the server ordering has high impact on scheduling performance. SSB and SSBO behave almost the same. The slight lead of SSBO over SSB, where present, is attributed to the optimisation in (10), upper-bounding the L-mode interference by s. This seems to mask the pessimism from independently bounding the fake task's interference in the two modes. SSB and SSBO perform similarly when there are fewer servers. However, their slight performance difference increases with more servers in the system. Conversely, the difference between SA and SSB increases with more servers. This indicates that the proposed analysis performs better with its intended dynamic setting. In general, SAAO and SA outperform SSB and other variants.

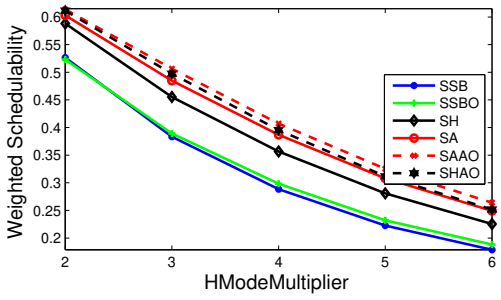


Fig. 8

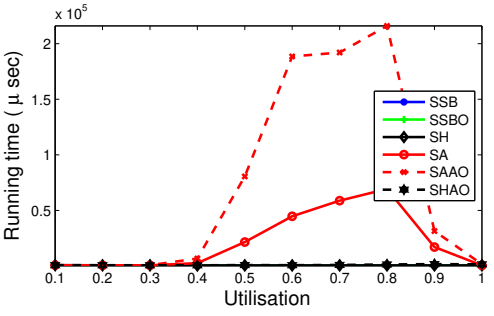


Fig. 9

Table 2. Maximum and minimum average running times in microseconds.

	SSB	SSBO	SH	SA	SAAO	SHAO
Max. (μs)	410	465	690	69300	216060	1633
Min. (μs)	270	220	450	467	506	580

The number of tasks per server is larger in Figure 5 when compared to Figure 4. For a given system utilisation, many light tasks per server are, more often than not, easier to schedule compared to fewer, heavy tasks per server. Hence, the weighted schedulablity is slightly higher in Figure 5 against Figure 4. This observation is consistent with the result shown in Figure 6, as the weighted schedulablity improves with a larger task set size. The number of servers is constant in Figure 6 and the larger task set size only increases the number of tasks per server, and consequently, the improvement in weighted schedulability.

More H-tasks in a task set and a higher H-mode utilisation scaling factor κ (Figures 7 and 8) both increase the H-mode utilisation, making the task sets harder to schedule. Hence, the weighted schedulability decreases accordingly for all heuristics.

The temperature had negligible effect in our experiments. Cooling rates above 0.1 were slighly detrimental for SAAO, and more so for SA. Meanwhile, a larger Δ improved the performance of SAAO and SA, but with diminishing returns above $\Delta = 0.05$. Ordering the servers by non-increasing $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$ performed slighly better than ordering by non-increasing order of $U^H(\tilde{P}_i)/U^L(\tilde{P}_i)$ – up to 0.5% in *unweighted* schedulability for SA; up to 0.8% for SH.

Compared to the baseline SSB, the absolute difference in terms of *unweighted schedulability success ratio* was up to 52.8% for SAAO. Such improvement allows more task sets to be schedulable on less powerful processors, hence reducing overall system cost.

Finally, we performed experiments to analyse the running time of the proposed heuristics. The hardware platform used in these experiments has 16 GB RAM, 8 i7-4710MQ cores with maximum frequency of 2.5 GHz and runs the Linux Mint 18.3 Sylvia operating system. Figure 9 presents the average running time per taskset with default parameters of the heuristics for different task set utilisation values. Table 2 shows the maximum and minimum average running times of the different heuristics for this experiment. The maximum average running time of SAAO and SA is approximately, 2 to 3 order of magnitude greater than other heuristics. The SHAO heuristic, with up to 42 times / 132 times respectively shorter average running time than SA and SAAO, has an unweighted scheduling success ratio up to 8.7% better than SA and marginally lower than SAAO (for the specified number of iterations). Hence, the SAAO heuristic provides better schedulability

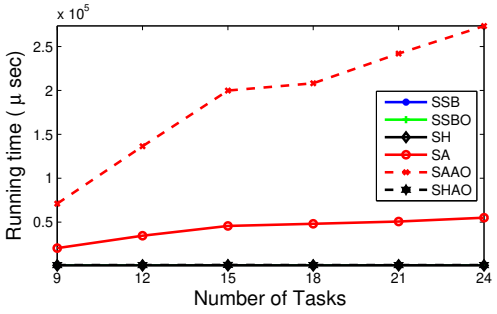


Fig. 10

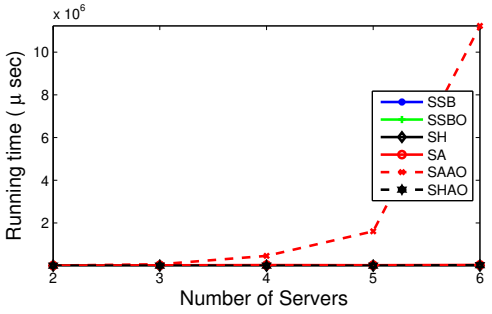


Fig. 11

success ratio, while the SHAO heuristic gives best results for computation time spent. For SA and SAAO, the system takes longer to find feasible solutions for middle values of the utilisation range. The drop at higher utilisation values happens due to early declaration of failure for unschedulable task sets.

Figures 10 and 11 present the average running time of the heuristics for different task set sizes and different number of servers, respectively. The running time of all heuristics increases approximately linearly with the task set size, because the schedulability test requires checking the schedulability of every task. However, this is visible only for SA and SAAO, because their running times are several orders of magnitude higher than those of the remaining heuristics. The increase in the number of servers exponentially increases the number of possible server orderings under both SHAO and SAAO, and therefore also the average running time. Again, this is barely visible for SHAO whose running time is several order of magnitude smaller than that of SAAO.

7 CONCLUSIONS

We proposed new schedulability analysis for mixed-criticality uniprocessor systems employing periodic servers in a cyclic executive manner and using AMC as the scheduling policy within each server. The novelty of the proposed scheduling analysis is that it supports varying server budgets in different modes. Our proposed approach provides strict temporal isolation among applications with the additional ability to efficiently utilise the available execution capacity across mode switches. We also proposed static and dynamic heuristics for assigning budgets to servers in both modes and for optimising the order of the servers in the cyclic executive schedule. Experimental evaluation with synthetic task sets showed that mode-dependent budgets can improve schedulability ratios by up to 52.8%, vs. static budgets. Even a simple heuristic can yield up to 27% of improvement. The order of the servers in the schedule can influence the schedulability ratio and the proposed heuristic for ordering them performs well in our experiments.

ACKNOWLEDGMENTS

This work was partially supported by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), within the CISTER Research Unit (UID/CEC/04234); also by the Operational Competitiveness Programme and Internationalization (COMPETE 2020) under the PT2020 Partnership Agreement, through the European Regional Development Fund (ERDF), and by national funds through the FCT, within project POCI-01-0145-FEDER-029119 (PREFECT). This research is supported by the Netherlands Organisation for Applied Scientific Research TNO.

REFERENCES

[1] 2016. Certification Authorities Software Team (CAST), Position Paper (CAST-32A) Multicore Processors.

[2] AERONAUTICAL RADIO, INC. 2010. *Avionics Application Software Standard Interface, Part 1, Required Services* (ARINC SPECIFICATION 653P1-3 ed.). AERONAUTICAL RADIO, INC.

[3] M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar. 2017. Semi-partitioned Mixed-Criticality Scheduling. In *Proceedings of the 30th International Conference Architecture of Computing Systems*. 205–218.

[4] Muhammad Ali Awan, Pedro F. Souto, Benny Akesson, Konstantinos Bletsas, and Eduardo Tovar. 2019. Uneven memory regulation for scheduling IMA applications on multi-core platforms. *Journal of Real–Time Systems* 55, 2 (01 Apr 2019), 248–292. https://doi.org/10.1007/s11241-018-9322-y

[5] S. K. Baruah and A. Burns. 2011. Implementing mixed criticality systems in Ada. In *16th Ada-Europe Conf.* 174–188.

[6] S. K. Baruah, A. Burns, and R. I. Davis. 2011. Response-Time Analysis for Mixed Criticality Systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium*. 34–43.

[7] A. Bastoni, B. Brandenburg, and J. H. Anderson. 2010. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. *In Proceeding of 6th Operating Systems Platforms for Embedded Real-Time applications*. (2010), 33–44.

[8] M. Beckert and R. Ernst. 2015. Designing time partitions for real-time hypervisor with sufficient temporal independence. In *Proceedings of the 52nd Design Automation Conference*. 1–6.

[9] Matthias Beckert and Rolf Ernst. 2017. Response Time Analysis for Sporadic Server Based Budget Scheduling in Real Time Virtualization Environments. *ACM Transactions on Embedded Computing Systems* 16, 5s, Article 161 (Sept. 2017), 19 pages. https://doi.org/10.1145/3126559

[10] E. Bini and G.C. Buttazzo. 2009. Measuring the Performance of Schedulability tests. *Journal of Real–Time Systems* 30, 1-2 (2009), 129–154.

[11] K Bletsas, MA Awan, PF Souto, B Akesson, A Burns, and E Tovar. 2018. De-coupling criticality and importance in mixed-criticality scheduling. In *Proceedings of the 6th Workshop on Mixed-Criticality Systems*.

[12] A. Burns and R.I. Davis. 2014. Adaptive Mixed Criticality Scheduling with Deferred Preemption. In *Proceedings of the 35rd IEEE Real-Time Systems Symposium*. 21–30.

[13] M. Chisholm, N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter. 2017. Supporting Mode Changes While Providing Hardware Isolation in Mixed-criticality Multicore Systems. In *Proceedings of the 25th Conference Real-Time and Networked Systems*. 58–67.

[14] C. Evripidou and A. Burns. 2016. Scheduling for Mixed-criticality Hypervisor Systems in the Automotive Domain. In *Proceedings of the 4th Workshop on Mixed-Criticality Systems*.

[15] X. Gu and A. Easwaran. 2016. Dynamic Budget Management with Service Guarantees for Mixed-Criticality Systems. In *Proceedings of the 37rd IEEE Real-Time Systems Symposium*. 47–56.

[16] X. Gu, A. Easwaran, K. Phan, and I. Shin. 2015. Resource Efficient Isolation Mechanisms in Mixed-Criticality Scheduling. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*. 13–24.

[17] F. Guan, L. Peng, L. Perneel, H. Fayyad-Kazan, and M. Timmerman. 2017. Adaptive Reservation into Mixed-Criticality Systems. *Scientific Programming* 2017 (2017).

[18] J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson. 2012. RTOS Support for Multicore Mixed-Criticality Systems. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*.

[19] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll. 2016. Adaptive Workload Management in Mixed-Criticality Systems. *ACM Transactions on Embedded Computing Systems* 16 (2016).

[20] B. Hu, L. Thiele, P. Huang, K. Huang, C. Griesbeck, and A.s Knoll. 2018. FFOB: Efficient online mode-switch procrastination in mixed-criticality systems. *Journal of Real–Time Systems* (2018).

[21] H.-M. Huang, C. Gill, and C. Lu. 2012. Implementation and Evaluation of Mixed-Criticality Scheduling Approaches for Periodic Tasks. In *Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium*. 23–32.

[22] Davis R. I. and Burns A. [n.d.]. Priority Assignment for Global Fixed Priority Pre-emptive Scheduling in Multiprocessor Real-Time Systems. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*.

[23] M. Joseph and P. Pandya. 1986. Finding Response Times in a Real-Time System. *Comput. J.* 29, 5 (1986), 390–395.

[24] N. Kim, B.C. Ward, M. Chisholm, C.-Y. Fu, J.H. Anderson, and F.D. Smith. 2017. Attacking the One-Out-Of-m Multicore Problem by Combining Hardware Management with Mixed-Criticality Provisioning. *Journal of Real–Time Systems* 53, 5 (2017).

[25] S. Kirkpatrick, C. D. Gelatt, , and M. P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220 (1983), 671—-680.

[26] Stephen Law, Iain Bate, and Benjamin Lesage. 2019. Industrial Application of a Partitioning Scheduler to Support Mixed Criticality Systems. In *Proceedings of the 31th Euromicro Conference on Real-Time Systems*.

[27] G. Lipari and G.C. Buttazzo. 2013. Resource reservation for mixed criticality systems. In *Workshop on Real-Time Systems: The past, The present, and the future*.

[28] E. Missimer, K. Missimer, and R. West. 2016. Mixed-criticality scheduling with I/O. In *Proceedings of the 28th Euromicro Conference on Real-Time Systems*. 120—-130.

[29] Alessandro Vittorio Papadopoulos, Enrico Bini, Sanjoy Baruah, and Alan Burns. 2018. AdaptMC: A Control-Theoretic Approach for Achieving Resilience in Mixed-Criticality Systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems*. 14:1–14:22.

[30] Jiankang Ren and Linh Thi Xuan Phan. 2015. Mixed-Criticality Scheduling on Multiprocessors Using Task Grouping. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems*. 25–34.

[31] RTCA, Inc. 2012. *RTCA/DO-178C*. U.S. Dept. of Transportation, Federal Aviation Administration.

[32] P.B. Sousa, K. Bletsas, E. Tovar, P. F. Souto, and B. Åkesson. 2014. Unified overhead-aware schedulability analysis for slot-based task-splitting. *Journal of Real–Time Systems* 50, 5-6 (2014), 680–735.

[33] P. Souto, P.B. Sousa, R.I. Davis, K. Bletsas, and E. Tovar. 2015. Overhead-Aware Schedulability Evaluation of Semi-Partitioned Real-Time Schedulers. In *Proceedings of the 21st IEEE Conference on Embedded and Real-Time Computing and Applications*. 110–121.

[34] Domitian Tamas-Selicean and Paul Pop. 2011. Optimization of Time-Partitions for Mixed-Criticality Real-Time Distributed Embedded Systems. In *Proceedings of the 14thIEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. 1–10.

[35] S. Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium*.