CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Technical Report

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

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

CISTER-TR-190202

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

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

CISTER Research Centre

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 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. However, a server's computational requirements may greatly vary in different modes, but 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 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, compared to static "one-size-fits-all-modes" budgets.

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

Muhammad Ali Awan
CISTER Research Centre and ISEP, Porto, Portugal
muaan@isep.ipp.pt

Konstantinos Bletsas
CISTER Research Centre and ISEP, Porto, Portugal
ksbs@isep.ipp.pt

Pedro F. Souto
University of Porto, Faculty of Engineering and CISTER Research Centre, Porto, Portugal
pfs@fe.up.pt

Benny Akesson
ESI (TNO), Eindhoven, the Netherlands
benny.akesson@tno.nl

Eduardo Tovar
CISTER Research Centre and ISEP, Porto, Portugal
emt@isep.ipp.pt

Abstract

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. However, a server's computational requirements may greatly vary in different modes, but 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 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, compared to static "one-size-fits-all-modes" budgets.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Real-time operating systems; Computer systems organization → Real-time system architecture

Keywords and phrases Mixed-criticality scheduling, Varying execution server, Vestal model

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Mixed-criticality systems are an important niche of real-time embedded systems. Their defining characteristic is the fact that computing tasks of different criticalities execute on the same hardware

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:30
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and share system resources, typically in order to reduce system costs[1]. 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 it 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.

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 application, especially if the former has low criticality and the latter has high criticality. 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. This is evidently grossly resource-inefficient and such over-engineering can have severe real-world costs. Most mixed-criticality systems are embedded, often meaning that profit margins for the end product can be thin and/or that engineering constraints on size, weight and power (SWaP) can be tight. Examples of such domains include automotive, aerospace and avionics.

Fortunately, the relevant guidelines (e.g., [1, 2] for the avionics domain), do not insist in 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 served by the particular server are allowed to use the core within the confines of that time window, scheduled under, e.g., a fixed-priority or an EDF policy. Conversely, task are not allowed to execute at all outside the confines of their respective server's time window. This arrangement both 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). Crucially, the implications of any task misbehaving temporally are localised to the respective server.

Meanwhile, mode-based scheduling arrangements [6] based on Vestal's model [35] can be applied in order to more efficiently use the available processing capacity. Rather than using extremely pessimistic worst-case execution time (WCET) estimates for both low- and high-criticality tasks that interact, this approach uses less pessimistic estimates, by default – *probably*, but not *provably* safe. In the statistically unlikely case of a task executing for longer than its assumed WCET, a carefully managed *mode change* is triggered. The less critical (and/or less important [9]) 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 mode-based model improves resource efficiency without compromising the system requirements.

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 (which is known as AMC [5], 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 naive 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.

[1] When tasks of different criticalities exist but are completely isolated, such systems are *multiple-criticality*, as opposed to mixed-criticality, and they constitute a different class of systems. See Footnote 1 in [12] and in [11].

However, it can still be inefficient if the execution time budgets used for the servers remain the same in different modes, because **a given server can have very different processing needs in one mode than in another**. 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 and the provision of safety guarantees. 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[2], 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 providing analysis for this arrangement, our work hence allows for greater resource efficiency and cost savings.

The main contributions of our work are the following: First, we formulate new schedulability analysis for uniprocessor systems using periodic servers with AMC as their scheduling policy and whose execution time budgets are dynamically adjusted in response to a mode change. 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 execution budgets in the different modes, for good schedulability performance. 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 by demonstrating up to 52.8% improvement in scheduling success ratio over the baseline heuristic.

The rest of this paper is stuctured as follows. Section 2 discusses related work. Section 3 presents the task model and the server scheduling model assumed. Section 4 contains the formulation of our new scheduling analysis. Section 5 enumerates the different scheduling arrangements and heuristics considered in our experimental evaluation, while Section 6 contains the evaluation itself and discusses our findings. Section 7 concludes.

2 Related work

In the literature, several works try to combine an adaptive mixed-criticality task scheduling model with servers in different ways. There also exist other works that are more loosely related.

Papadopoulos et al. [31] consider the adaptive mixed-criticality model of Vestal [6] with two criticality levels and a uniprocessor whose processing time supply is partitioned into two servers, one for the tasks of each criticality level, with corresponding target execution time budgets. At run-time, the execution times of the tasks are monitored, and in a control feedback loop, the execution budgets of each successive server instance are adjusted, according to the principles of control theory [32]. The budget adjustments may introduce some deviation from strict server periodicity. The aim of this approach is to preserve as much service to lower-criticality tasks as can be afforded by the state of the system, instead of dropping them altogether when a mode change occurs.

Ren et al. [33] consider the same task model [6] with similar aims, but with a different approach. In a multicore, each core is assigned different high- and low-criticality tasks (i.e., task to core partitioning). On each core, disjoint groups of one high-criticality task and one or more low-criticality tasks are formed. For each group, different servers for the high- and low-criticality tasks are employed, such that the provision of scheduling guarantees to the former has the least impact on the latter. At the task group level, groups are scheduled on the corresponding core using EDF.

[2] We are refering to the use of a “fake interfering task", exactly as done for EDF servers in [3], that models the periodic unavailability of the server.

The on-the-fly fast overrun budgeting mechanism by Hu et al. [23] improves the system's quality of service for low-criticality tasks by postponing the mode-switch instance. This approach (inspired by procrastination techniques [4, 29]) utilises the collected static and dynamic slack for job overruns. Similarly, Gu et al. [18] compute the sufficient L-mode budget for high-criticality applications collectively at design time. This budget is utilised at run time to schedule the high-criticality applications in L-mode with the objective of postponing the mode switch as much as possible. The budget assignment can be tuned for system-wise objectives of schedulability and service guarantees for low-criticality applications. Similarly, Hu et al. [22] regulate the low-criticality workload considering the online demand of high-criticality applications with the objective of improving the quality of service for low-criticality applications.

Lipari and Buttazzo's [17] reservation-based approach, assigns to each high-criticality task a server with a computation bandwidth equal to its high-criticality-mode utilisation. A single low-criticality server (initially assigned the leftover utilisation) serves all low-criticality tasks. The bandwidth reclaimed from high-criticality servers is assigned to the low-criticality server. Fei et al. [20] adjust server budgets recurrently, based on a predictor of future job execution times. Evripidou and Burns [16] consider a uniprocessor platform with multiple partitions. There exist a periodic server for the periodic tasks and a deferable higher-priority server for the sporadic tasks in each partition. Missimer et al. [30] similarly employ sporadic servers and priority-inheritance bandwidth-preserving servers integrate I/O- and task-scheduling, albeit under a fixed-priority scheduling policy.

Gu et al. [19] focused on component-based systems. Within a component, as long as the number of low-criticality mode execution time overruns does not exceed its predefined tolerance level, other tasks (of any criticality) in other components are unaffected. Collectively, tasks are scheduled with an EDF-based policy. Some recent studies [13, 21, 24, 27] explored the implementation-level details of scheduling approaches (including hierarchical ones) for mixed-criticality systems and how to combine adaptive mixed-criticality scheduling with predictable hardware.

In [3], we considered different server-based arrangements for strict isolation between criticalities and good schedulability on multicores, using scaled-deadline EDF [15]. One approach uses one server per core for high-criticality tasks and uses the spare capacity for low-criticality task servers. At mode change, the latter are dropped and the high-criticality servers can use the entire core. Another approach uses fixed-budget servers serving tasks of mixed-criticalities. At mode change, low-criticality tasks in each server are dropped but the servers and their budgets persist, serving the remaining tasks.

By comparison with [3], our present work brings fully adjustable server budgets at mode change and targets a fixed-priority-scheduled [5] uniprocessor platform. Multiple servers exist, each serving tasks of mixed criticalities. At mode change, the low-criticality tasks are still discarded, however 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.

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 [5]. In particular, 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 relative deadline D_i that is constrained (i.e., $D_i \leq T_i$) 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

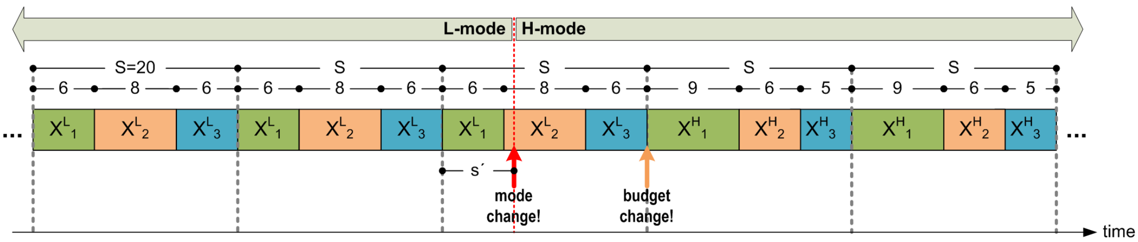

Figure 1 At mode change, the L-tasks are dropped and the remaining H-tasks must be schedulable as long as they execute for up to their C_i^H estimates (including any jobs thereof caught up in the mode change). However, server execution budgets are only adjusted from X_q^L to X_q^H at the start of the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

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 (called the "timeslot length", to stick to the terminology used in related work [3]) and they execute one after the other in the same order, in a form of cyclic executive with a periodicity of S. This fixed order in which the servers execute is specified by the designer, 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.

The system conforms to the task model defined in Section 3.1, meaning that there exist two modes, L and H. Each server $\tilde{P}_q$ has a fixed time budget X_q^L in the L-mode and a respective fixed time budget X_q^H for the H-mode. 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 that 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 are adjusted to their respective X_q^H values *at the start of the next timeslot*. This implies that there is a time interval of s' time units ($0 \leq s' < S$) after the mode change, during which the system is already in H-mode, but the server budgets are not yet adjusted from the values (X_i^L) used in the L-mode. Figure 1 illustrates this arrangement via an example schedule.

4 Schedulability Analysis

4.1 Schedulability analysis for an individual server

In this subsection, we are going to derive a sufficient schedulability test for a server conforming to the model described earlier. More specifically, 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 will establish whether the server is mixed-criticality-schedulable. The questions 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 upon AMC-max [5] 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 [26]. However, as in [34] 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:

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

In (1), the server index is omitted for clarity of presentation and $hp(i)$ is the set of higher-priority tasks served by 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, in the general case, also entails a potential server budget change (though not necessarily coincident), we have to consider four mutually exclusive and jointly exhaustive cases (elaborated below). The task under analysis must meet its deadline in all of those cases. The reason for having to consider four separate cases is the following: If the server budgets change after a mode change (not necessarily immediately), then the worst-case scenario, maximising the response time of a job that completes in the H-mode, does not necessarily involve that job being released before the mode change – unlike what holds for 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, in the general case $t_{hp(i)}^{idle}$ might be located before the server budget change instant or after it, or may coincide with it. The four cases to consider are:

- **Case 1:** The H-task underconsideration is released before the mode change instant s (or even at the mode change instant, as a corner case), but completes after the mode change.
- **Case 2:** The H-task is released after the mode change, but before $t_{hp(i)}^{idle}$.
- **Case 3:** The H-task is released at or after $t_{hp(i)}^{idle}$ **and** also before the server budget change instant.

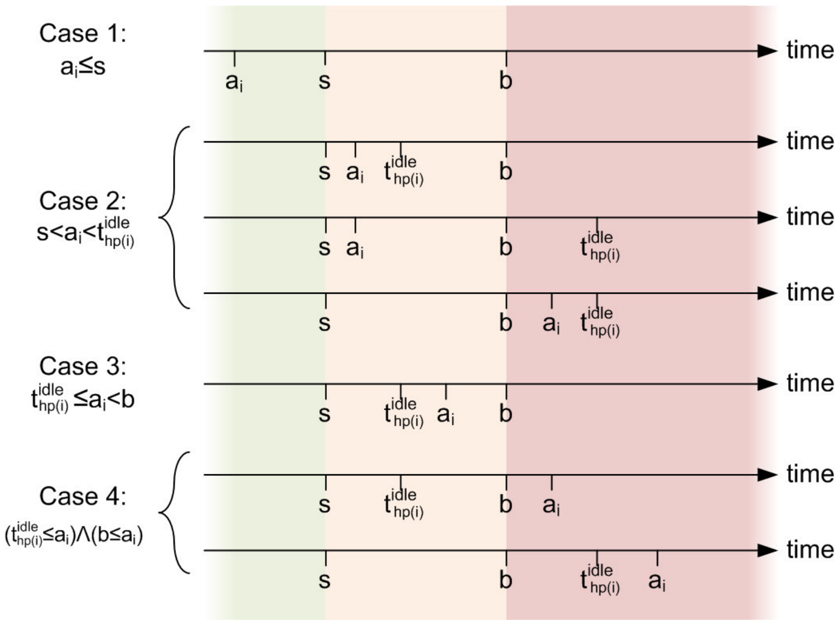

Figure 2 All possible relative orderings between the mode change instant (s), the budget change instant (b), the arrival time (a_i) of the H-task under analysis and $t^{idle}_{hp(i)}$, for the four cases.

- **Case 4:** The H-task is released at or after $t^{idle}_{hp(i)}$ **and** also at or after the budget change instant.

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 $t^{idle}_{hp(i)}$, for the four cases.

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). Below, we will upper-bound that separately for in each mode.

Assume that the mode change occurs some s' time units after the last timeslot boundary. Then, $s' \in [0, S)$. Let $W_f^{H|tr.}(s', \Delta t)$ denote the worst-case workload function of the fake task modelling the unavailability of the reserve, for the interval $[s, s + \Delta t)$. The first parameter denotes the phasing of the mode change relative to the timeslot boundary, as explained earlier. Since s' cannot be known offline (but only established *a posteriori*), to upper-bound that workload function in the general case would need to upper bound $W_f^{H|tr.}(s', \Delta t)$ for all $s' \in [0, S)$. Fortunately, since the scheduling is by fixed priorities, we need only consider two values for s'. Namely, $s' = 0$ and $s' = O^L + X^L$. 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). For any $s' > 0$, up to the value of O^L, this initial interference would be smaller (i.e., $O^L - s' < O^L$). The other value that we need to consider ($s' = O^L + X^L$) corresponds to the mode change occurring just as the server has ran out budget. 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; 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. Figure 3 illustrates these cases. Accordingly,

$$W_f^{H|tr.}(\Delta t) = \max_{s' \in [0,S)} W_f^{H|tr.}(s', \Delta t) = \max\left(W_f^{H|tr.}(0, \Delta t), W_f^{H|tr.}(O^L + X^L, \Delta t)\right) \quad (2)$$

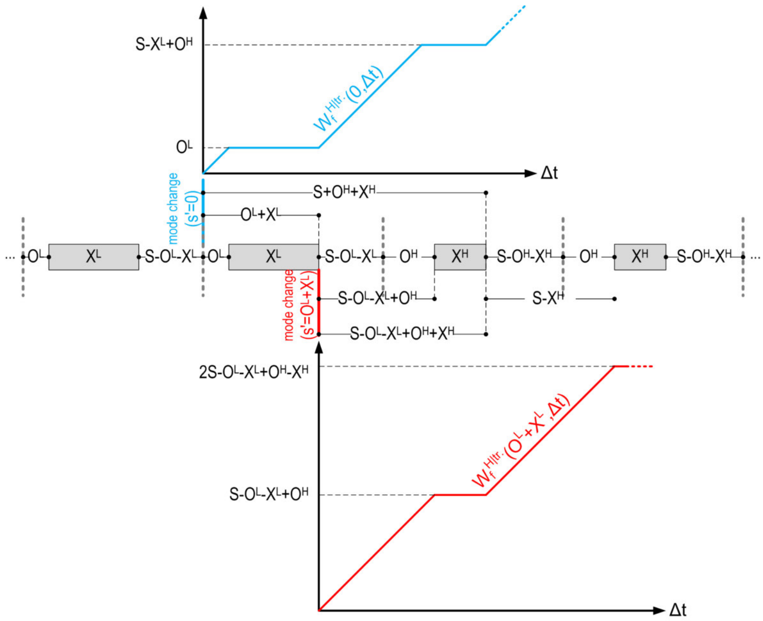

Figure 3 A mode change occurs s' time units after a timeslot boundary ($0 \leq s' < S$). However, the joint consideration of cases ($s'=0$) and ($s'=O^L+X^L$) dominates all other $s' \in [0, S)$ under our analysis, because (i) shifting s' from $s' = 0$ to the right, by up to O^L time units can only decrease the post-mode-change interference from the fake task and, similarly, (ii) shifting s' from $s' = O^L + X^L$ to the left by up to X^L time units, or by any amount to the right, within the same timeslot, has the same effect. Plotted at the top (blue) and bottom (red) are the corresponding workload curves, $W_f^{H|tr.}(0, \Delta t)$ and $W_f^{H|tr.}(O^L + X^L, \Delta t)$ for the fake task after the mode change.

Equation (2) upper-bounds the fake task's "execution" (i.e., unavailability of the server) over any interval of length Δt starting at s, the mode change instant. By inspection of Figure 3 (blue plot):

$$W_f^{H|tr.}(0, \Delta t) = \begin{cases} \Delta t, & \Delta t \leq O^L \\ O^L, & O^L < \Delta t \leq O^L + X^L \\ \Delta t - X^L, & O^L + X^L < \Delta t \leq S + O^H \\ S + O^H - X^L + \left\lfloor \frac{\Delta t - (S + O^H)}{S} \right\rfloor (S - X^H) & \\ + \max\left(0, \Delta t - (S + O^H) - \left(\left\lfloor \frac{\Delta t - (S+O^H)}{S} \right\rfloor S\right) - X^H\right), & S + O^H < \Delta t \end{cases} \quad (3)$$

Likewise, by inspecting Figure 3 (red plot), we obtain for $W_f^{H|tr.}(O^L + X^L, \Delta t)$ the expression:

$$W_f^{H|tr.}(O^L + X^L, \Delta t) = W_f^{H|tr.}(0, \Delta t + O^L + X^L) - O^L \quad (4)$$

Accordingly, the corresponding equivalent request-bound functions for any interval $[s, s + \Delta t))$ are:

$$\begin{aligned} I_f^{H|tr.}(0, \Delta t) &= O^L + \min\left(1, \left\lceil \frac{\Delta t - O^L - X^L}{S} \right\rceil_0\right)(S - X^L - O^L + O^H) \\ &+ \left\lceil \frac{\Delta t - S - O^H - X^H}{S} \right\rceil_0 (S - X^H) \end{aligned} \quad (5)$$

$$
\begin{aligned}
I_f^{H|tr.}(O^L + X^L, \Delta t) &= (S - X^L - O^L + O^H) \\
&+ \left\lfloor \frac{\Delta t - S + O^L - O^H + X^L - X^H}{S} \right\rfloor_0 (S - X^H)
\end{aligned} \tag{6}
$$

Analogously as before,

$$
I_f^{H|tr.}(\Delta t) = \max\left(I_f^{H|tr.}(0, \Delta t), I_f^{H|tr.}(O^L + X^L, \Delta t) \right) \tag{7}
$$

Next, we will incorporate $I_f^{H|tr.}(\Delta t)$ 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

$$
R_i^{H|1} = \max(R_i^{s|1}),\ \forall s \in [0, R_i^L) \tag{8}
$$

where

$$
R_i^{s|1} = C_i^H + \sum_{\tau_j \in hpL(i)} IL_j(s) + \sum_{\tau_k \in hpH(i)} IH_k(s, R_i^{s|1}) + \underbrace{IL_f(s) + I_f^{H|tr.}(R_i^{s|1} - s)}_{fake\ task} \tag{9}
$$

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

$$
\begin{aligned}
IL_j(s) &= \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j^L & (10) \\
IL_f(s) &= \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) (S - X^L) & (11) \\
IH_k(s,t) &= M(k,s,t) C_k^H + \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k,s,t) \right) C_k^L & (12)
\end{aligned}
$$

where, classically from [5], $M(k,s,t) = \min\left(\left\lceil \frac{t - s - T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right)$.

By replacing IH_k in (9) with the RHS of (12), we obtain:

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

Splitting the second summation and reordering the terms:

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

Finally, in a slight accuracy optimisation of ours,

$$
R_i^{s|1} = \ C_i^H + \left[\!\left[IL(s, R_i^{s|1}) \right]\!\right]^s + IH(s, R_i^{s|1}) \tag{13}
$$

where:

$$IL(s, R_i^{s|1}) = \sum_{\tau_j \in hpL(i)} IL_j(s) + IL_f(s) + \sum_{\tau_k \in hpH(i)} \left(\left\lceil \frac{t}{T_k} \right\rceil - M(k, s, R_i^{s|1}) \right) C_k^L$$

$$IH(s, R_i^{s|1}) = \sum_{\tau_k \in hpH(i)} M(k, s, R_i^{s|1}) C_k^H + I_f^{H|tr.}(R_i^{s|1} - s)$$

and the operator $\llbracket \cdot \rrbracket^{\max}$ is defined as $\llbracket x \rrbracket^{\max} \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \leq \max \\ \max & \text{if } x > \max \end{cases}$.

Under the original AMC-max, the operator $\llbracket \cdot \rrbracket^{s}$ is not used. Our slight improvement acknowledges that the interference from all jobs completed before the mode change cannot exceed s. [3]

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 $t^{idle}_{hp(i)}$.

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 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 1.** *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. Assume that a H-task τ_i is schedulable in L-mode and in H-mode under Case 1. Then, consider some 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 hold:

- The instant t'' is located at or before the mode change (i.e., $t'' \leq s$).
- 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 tasks), 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

[3] Not enclosing the expression by the operator $\llbracket \cdot \rrbracket^{s}$, would allow the analysis to hold even for a variant model that permits any L-jobs caught up in the mode change to complete, executing for up to the respective L-WCETs. This follows from the original AMC-max – see [5].

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 $a'_i <^{idle}_{hp(i)}$. 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 it is schedulable, which we proved earlier. ◀

This means that the schedulability test for Case 2 is subsumed by the one for Case 1. 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. Accordingly, since we have to test for Case 1 anyway, it is redundant to test for Case 2.

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, given that in Case 3 the release time a_i of τ_i is before the server budget change instant, it is upper-bounded by (7), as in Case 1 (using the exact same reasoning).

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 the WCRT of the task, 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 WCRT of $S - X^H$ and a period of S. The corresponding equation is:

$$R_i^{H|4} = C_i^H + \sum_{\tau_j \in hpH(i)} \left\lceil \frac{R_i^{H|4}}{T_j} \right\rceil C_j^H + \underbrace{\left\lceil \frac{R_i^{H|4}}{S} \right\rceil (S - X^H)}_{fake\ task} \tag{14}$$

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 $I_f^{H|tr.}(\Delta t)$ (Equation (7)) 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, so we need to test for Case 4 separately.

4.2 Schedulability analysis at the system level

Having formulated how to test for the schedulability of a given server, 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 server schedulability test from Section 4.1 with the assigned execution budgets and starting offsets. In notation:

$$\begin{aligned}
& (\forall\, i:\ \tilde{P}_i \text{ is schedulable with } (X_i^L, X_i^H, O_i^L, O_i^H)) \\
\wedge \quad & (\forall\, i:\ (O_i^L + X_i^L \leq S) \wedge (O_i^H + X_i^H \leq S)) \\
\wedge \quad & (\forall\, i, j, i \neq j : (O_i^L + X_i^L \leq O_j^L) \vee (O_j^L + X_j^L \leq O_i^L)) \\
\wedge \quad & (\forall\, i, j, i \neq j : (O_i^H + X_i^H \leq O_j^H) \vee (O_j^H + X_j^H \leq O_i^H))
\end{aligned}$$

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

$$O_1^L = 0;\ O_i^L = O_{i-1}^L + X_{i-1}^L,\ \forall i > 1 \tag{15}$$

$$O_1^H = 0;\ O_i^H = O_{i-1}^H + X_{i-1}^H,\ \forall i > 1 \tag{16}$$

This allows the schedulability condition to be simplified to

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

4.2.1 Interdependencies between the parameters of different servers

From (15) and (16), 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 for the system schedulability.

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, then $O_i^L - O_i^H \geq 0$ 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

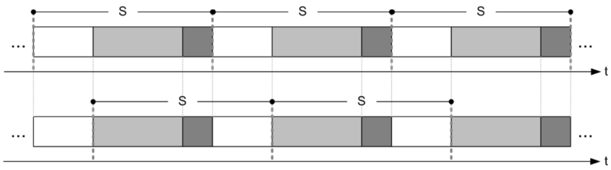

Figure 4 The two server schedules depicted are identical (since each server occupies exactly the same time windows in both), the only difference being which time instant is considered as the start of the periodic timeslot of length S. Therefore, one system being (un)schedulable implies that the other one is too.

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 Server budget assignment heuristics

We consider two different scheduling arrangements (static-server budgets and dynamic server budgets) and different heuristics for assigning the server budgets in each case.

5.1 Static server budgets (SSB)

Under the static server budget (SSB) arrangement, the execution budget of a server remains 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 $\forall i$. In other words, neither the starting offset nor the execution budget of any server ever changes. Consequently, there is no need to apply the analysis formulated in Section 4. Rather, we can apply the original AMC-max schedulability test [5], for sizing each server, with the addition of a top-priority fake H-task τ_f with attributes C_f^L=C_f^H=$S - X_i$ and T_f=S. The minimum feasible server budget X_i for a given server can be identified via binary search [3, 34] over the interval $(0, S]$. Note that for this arrangement, each server can be sized independently of other servers and their attributes. Moreover, the order in which the servers are arranged on a processor is irrelevant.

The previously mentioned independence between servers also holds when our analysis is used instead of the original AMC-max analysis to test the feasibility while sizing servers with static budgets. Indeed, the offsets of the server under analysis (and all other servers) can be disregarded because any static-budget server execution pattern can be transformed via shift-rotation along the temporal axis into an equivalent schedule where the server under consideration has a given offset (which can conveniently be $O_i^L = O_i^H = 0$), as Figure 4 illustrates. Crucially, the unavailability intervals for the server all have a duration of $S - X_i$ and occur strictly periodically with a period of S.

Nevertheless, even if our new analysis can accommodate static server budgets as a special case, it does not necessarily perform better than the original 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 9). In any case, for greater insight, in Section 6, we plot results for SSB using either the original AMC-max or the new analysis.

5.2 Dynamic server budgets (DSB)

Under this arrangement, for which our analysis was developed, *a server can have a different budget after a mode switch*. As in the SSB arrangement, we assume that 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 budgets of a server are computed offline. To compute the minimum feasible budget X_i^H of a server in H-mode, 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. In Section 5.2.1, we discuss heuristics for selecting that ordering.

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 [34]). 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 a metaheuristic (Simulated Annealing [28]), 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.

Simulated annealing attempts to replace the current solution of a problem with another (randomly obtained) solution in each iteration. A candidate solution that improves on the current one is always accepted. However, occasionally, the algorithm will also accept a "worse" candidate solution with a probability that depends on the value of a probability function. This function takes as parameters a variable Θ (dubbed "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 helps avoid the pitfall of getting stuck at a local optimum of the optimization problem. The temperature Θ is gradually decreased with the number of iterations. In our particular problem, 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 $O_i^L = O_i^H = 0$.

In more detail, the pseudocode for both heuristics is presented in Algorithm 1.

Initial Phase (Simple heuristic): Initially, we determine the minimum feasible L-mode budget X_i^{min} for each server. To do that, for each server separately, we assume that its H-mode budget is equal to S (the entire timeslot) and its starting offsets are equal to 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 with a binary search algorithm [34]. If the sum of X_i^{min} for all servers is greater than 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. Otherwise, once X_i^{min} has been computed for all servers, we set $X_i^L = X_i^{min} * \frac{S}{\sum_{\forall i} X_i^{min}}$ for each server. 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 initial X_i^L server budgets in L-mode 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 with a binary search algorithm assuming offsets of $O_i^L = \sum_{j=1}^{i-1} X_j^L$ and $O_i^H = \sum_{j=1}^{i-1} X_j^H$. If for the computed X_i^H values it holds that $\sum_{\forall i} X_i^H \leq S$, we declare a success. Otherwise, we try the metaheuristic (Simulated Annealing), implemented in the main loop:

Main Loop (Simulated Annealing): In any iteration k, two servers ($\tilde{P}_a$ and $\tilde{P}_b$) are selected randomly. This heuristic increments X_a^L of $\tilde{P}_a$ and decrements X_b^L of $\tilde{P}_b$ by a same value of β, where β represents the server variation length parameter for this iteration. Adding and subtracting the same value of β from two selected servers 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 through Algorithm 2 in such a way that this condition is never violated. In Algorithm 2, initially, $\beta_{max} = -(\Delta * S) + (2 * \gamma)$ gives the maximum length to vary in this iteration, where $\Delta \in (0, 1]$ is an input parameter, and γ is a randomly generated variable in $(0, \Delta * S]$. Afterwards, β_a and β_b are selected

Algorithm 1 Simple heuristic and Simulated Annealing algorithm

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

<u>**Initial Phase:**</u>
Generate X_i^{min} for each server $\tilde{P}_i$ using a binary search algorithm [34] assuming X_i^H=S.
if ($\sum_{\forall i} X_i^{min} > S$ || any $\tilde{P}_i$ infeasible with X_i^H=S) **then return** Failure
else ▷ scale up X_i^L values
    Set $X_i^L = X_i^{min} \times \frac{S}{\sum_{\forall i} X_i^{min}}, \forall i$
Compute X_i^H for each server, given X_i^L values $\forall i$, with offsets
if ($\sum_{\forall i} X_i^H \leq S$) **then return** $X_i^L, X_i^H, \forall i$
<u>**Main Loop:**</u>
while ($\Theta > 1$) **do**
    Select two random servers $\tilde{P}_a$ and $\tilde{P}_b$
    Compute β for servers $\tilde{P}_a$ and $\tilde{P}_b$ with Algorithm 2
    Set $X_a^L(k) = X_a^L(k\text{-}1) + \beta$ and $X_b^L(k) = X_b^L(k\text{-}1) - \beta$
    Set other servers $X_i^L(k) = X_i^L(k-1), \forall i \notin a, b$
    Compute $X_i^H(k), \forall i$ in k^{th} iteration with offsets
    if ($\sum_{\forall i} X_i^H \leq S$) **then return** $X_i^L(k), X_i^H(k), \forall i$
    else
        Compute $X_i^H(0,k), \forall i$ with $O_i^L = O_i^H = 0$ offset in iteration k
        if ($\sum_{\forall i} X_i^H(0,k) < \sum_{\forall i} X_i^H(0,k-1)$) **then**
            Keep $X_i^L(k), X_i^H(0,k), \forall i$ for k+1th iteration
        else if (z∈(0,1]≤$e^{\frac{\sum_{\forall i} X_i^H(0,k-1) < \sum_{\forall i} X_i^H(0,k)}{\Theta}}$) **then**
            Keep $X_i^L(k), X_i^H(0,k), \forall i$ for k+1th iteration
        else
            Discard this and keep $(k\text{-}1)^{th}$ iteration solution
    $\Theta = \Theta * (1-$ Cooling Rate$)$
    $k = k + 1$
On while loop termination without success, **return** Failure

such that $X_a^L + \beta_a$ remains in $[X_a^{min}, S]$ and $X_b^L - \beta_b$ remains in $[X_b^{min}, S]$. Between β_a and β_b, the one that gives the least change in server size is selected as β.

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., $X_i^L(k) = X_i^L(k-1)$, $\forall i \neq a$, $i \neq b$. 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 $\tilde{P}_i$.

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. 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 $O_i^L = O_i^H = 0$. 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. 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.

Algorithm 2 Server variation length parameter (β) computation algorithm

Input: $\tilde{P}_a$, $\tilde{P}_b$, S and Δ
Output: Server size variation factor β

$\beta_{max} = -(\Delta * S) + (2 * \gamma)$ ▷ Parameter $\Delta \in (0, 1]$; γ uniformly distributed over $(0, \Delta * S]$.
if $(X_a^L + \beta_{max} > S)$ **then**
  $\beta_a = S - X_a^L$
else if $(X_a^L + \beta_{max} < X_a^{min})$ **then**
  $\beta_a = X_a^{min} - X_a^L$
else
  $\beta_a = \beta_{max}$
if $(X_b^L - \beta_{max} > S)$ **then**
  $\beta_b = X_b^L - S$
else if $(X_b^L - \beta_{max} < X_b^{min})$ **then**
  $\beta_b = X_b^L - X_b^{min}$
else
  $\beta_b = \beta_{max}$
if $(\beta_{max} \geq 0)$ **then**
  $\beta = \min\{\beta_a, \beta_b\}$
else
  $\beta = \max\{\beta_a, \beta_b\}$
return β

Finally, if the system cools downs without finding any feasible solution, a failure is declared.

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 4.2.1, 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. In our set of experiments, we also experimented with server ordering by non-increasing $\frac{U^H(\tilde{P}_i)}{U^L(\tilde{P}_i)}$, but it performed slightly worse.

6 Evaluation

6.1 Experimental setup

We have developed a Java tool to implement the proposed analysis and explore the scheduling performance of different scheduling arrangements and budget assignment heuristics and the effectiveness of the schedulability analysis. Our tool has two modules. The first module generates the synthetic workload (task sets and servers) for the given platform parameters. A second module performs the feasibility analysis with the proposed techniques.

Task set generation: Task periods are generated with a log-uniform distribution in the 10-100

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 factor (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
Server size variation (β)	$\{.001, .005, .01, .02, .05, .1, .2, .3, .4, .5, .6, .7, .8, .9\}$	0.5
Server ordering	$\{U^H(\tilde{P}_i) - U^L(\tilde{P}_i), \frac{U^H(\tilde{P}_i)}{U^L(\tilde{P}_i)}\}$	1^{st}
Inter-arrival time (T_i)	$10ms$ to $100ms$	N/A
Nominal utilisation	$\{0.1:0.1:1\}$	N/A

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 by the UUnifast algorithm [8, 14] in an unbiased way. Then, a task's L-WCET C_i^L is computed to $T_i * U_i^L$, where U_i^L is the task's 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 priorities based on their arrival rates (Deadline Monotonic). The generated task set is indexed in order of increasing deadlines and tasks 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, we assume that the designer may not have control over this, since the grouping of tasks is functional, on the basis of appplication. Finally, for each system, the timeslot length (S), corresponding to the common period of all servers, is set to the shortest task interarrival time in the task set.

The target L-mode system utilisation is varied within a range of $(0, 1]$. Different random class objects are used to generate period and utilisation values. Each random object is seeded with a different odd number and reused in successive replications [25]. For each set of input parameters, we generate 1000 random task sets. The default values of the aforementioned parameters are: $q = 3$, $n = q * 3$, 30% H-tasks and $\kappa = 3$. The range of values considered for all the parameters is presented in Table 1. By default the servers are sorted in order of non-decreasing $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$. The triple given in this table corresponds to {minimum : increment granularity : maximum} values of a parameter. In our experiments, we vary one parameter from Table 1 at a time, while the others conform to their default values.

Instead of providing plots comparing the approaches in terms of scheduling success ratio (i.e., the fraction of task-sets deemed schedulable under the respective schedulability test), we condense this information by providing plots of *weighted schedulability*. This performance metric [7, 10] condenses what would have been three-dimensional plots into two dimensions. It is a weighted average that gives more weight to task-sets with higher utilisation, which are supposedly harder to schedule. Specifically, using notation from [10], let $S_y(\tau, p)$ represent the result (0 or 1) of the schedulability test y for a given task-set τ with an input parameter p. Then $W_y(p)$, the weighted schedulability for that test y as a function p, is $W_y(p) = \sum_{\forall \tau} (U(\tau) * S_y(\tau, p)) \,/ \sum_{\forall \tau} U(\tau)$, where $U(\tau)$ is the utilisation of task set τ. In this paper, the weighing is according to the L-mode schedulability. Nevertheless, the schedulability success ratio plots are presented in Appendix A.

6.2 Scheduling arrangements and server ordering heuristics

The following scheduling arrangements and heuristics are compared in this section.

- **Static Server Budgets (SSB)**: This scheduling arrangement (presented in Section 5) maintains the same size of a server in both modes. We use the original AMC-max analysis [5] for the schedulability test, for fair comparison. A server's budget is set to the minimal value that ensures schedulability, according to sensitivity analysis (binary search).
- **Static Server Budgets with our analysis (SSBO)**: This is the same as SSB, except that in place of the original AMC-max test, 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 assess the pessimism, compared to AMC-max, when the new analysis deals with the special case of $X_i^L = X_i^H\ \forall i$. Any such pessimism can be attributed to the independent bounding of the fake task's interference in the two modes, in the design of our analysis, to be able to analyse the general case of dynamic budgets.
- **Dynamic Server Budgets with Simple heuristic (SH)**: This approach for dynamic server budgets declares a success, if the system is feasible using the simple heuristic for assigning budgets that we described in Section 5, i.e., the "Initial Phase" from Algorithm 1. Our new analysis is used for schedulability testing.
- **Dynamic Budgets with Simulated Annealing (SA)**: The simulated annealing heuristic is also presented in Section 5, in the latter part of Algorithm 1. Once again, our new analysis is used for schedulability testing. The corresponding curve plots success 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 is able to find a feasible assignment of server budgets for the two modes. The variation in three parameters (Temperature, Cooling Rate and Δ) used to configure the simulated annealing metaheuristic is presented in Table 1.
- **Simulated Annealing with all possible server orderings (SAAO)**: Instead of picking a pre-defined server ordering, all possible orderings in which the servers can be arranged are tested for a feasible solution. For each of them, the SA heuristic (Dynamic Budgets with Simulated Annealing) explained earlier is applied, with the specifed maximum number of iterations, until success. The SAAO heuristic allows us: (a) to analyze the effect of the order in which the servers are arranged, and (b) to quantify the quality of the default ordering of $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$ and, indirectly, the validity of the intuition behind its selection.
- **Simple heuristic assuming all possible server orderings (SHAO)**: It is similar to SH except that all possible ordering of servers on the given processor are checked instead of the default order.

6.3 Results

Figures 5 and 6 present the weighted schedulability for different number of servers. The number of tasks per server is constant (3/Server) in Figure 5, while the total number of tasks to distribute among servers via a round robin policy is constant (task set size = 18) in Figure 6. 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 indicates that the selected server ordering by non-decreasing $(U^H(\tilde{P}_i) - U^L(\tilde{P}_i))$ is a reasonable 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 finding 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 can be seen when comparing SHAO and SAAO. In the majority of the cases, SHAO performs better than SA, which indicates that the order of servers

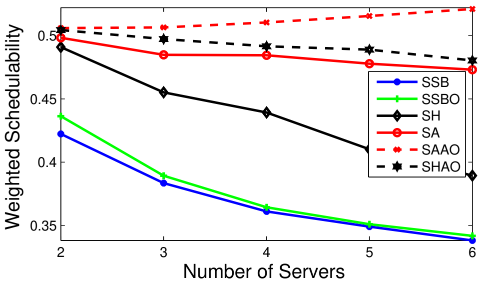

Figure 5 (Fixed number of tasks per server)

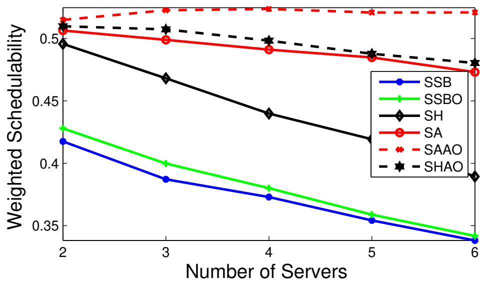

Figure 6 (Fixed number of total tasks)

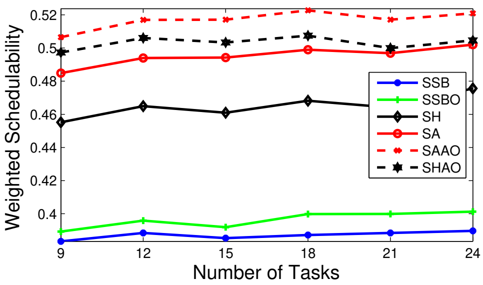

Figure 7

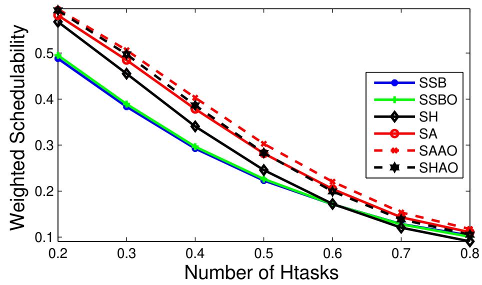

Figure 8

has high impact on the feasibility of the solution. SSB and SSBO behave almost the same. The slight improvement of SSBO over SSB, where present, is attributed to the optimisation in (13), where all L-mode interference is upper-bounded by s. This seems to mask the pessimism from the independent bounding of the fake task's interference in the two modes. SSB and SSBO perform similarly when the servers are fewer. However, their slight performance difference increases with more servers in the system. On the other hand, 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 diplay superior performance to SSB and other variants.

The number of tasks per server is larger in Figure 6 when compared to Figure 5. 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 6 against Figure 5. This observation is consistent with the result shown in Figure 7, as the weighted schedulablity improves with a larger task set size. The number of servers is constant in Figure 7 and the increase in task set size only increases the number of tasks per server, and consequently, the improvement in weighted schedulability.

A higher number of H-tasks (Figure 8) in a task set and a higher H-mode utilisation scaling factor κ (Figure 9) both increase the H-mode utilisation, making the task sets harder to schedule. Hence, the weighted schedulability decreases with an increase in these parameters for all heuristics. A higher temperature and a lower cooling rate increases the number of solution mutations during the iterations of the main loop in SA, and appear to improve the weighted schedulability, as shown in Figures 10 and 11, respectively. A larger value of Δ provides a bigger area of the design space to explore in each iteration of the simulated annealing metaheuristic, and, from the results, this improves the weighted schedulability, as shown in Figure 12. The order of servers is important for both SH and SA. The servers sorted in non-increasing order of $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$ show up to 0.5% and 0.8% better schedulability success ratio than non-increasing order of $\frac{U^H(\tilde{P}_i)}{U^L(\tilde{P}_i)}$, for SA and SH, respectively.

When compared to the baseline SSB, the absolute difference in terms of *schedulability success*

Table 2 Maximum absolute difference in schedulability ratio of all heuristics from baseline SSB.

SA	SH	SAAO	SSBO	SHAO
39.6%	27.9%	52.8%	5.6%	40.4%

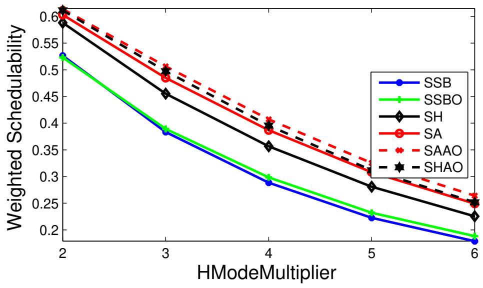

Figure 9

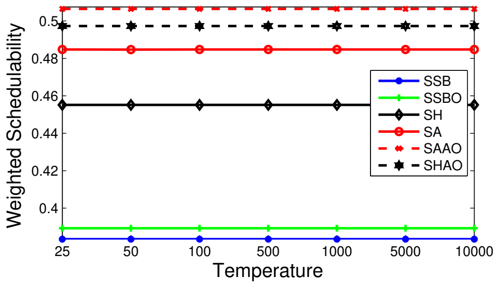

Figure 10

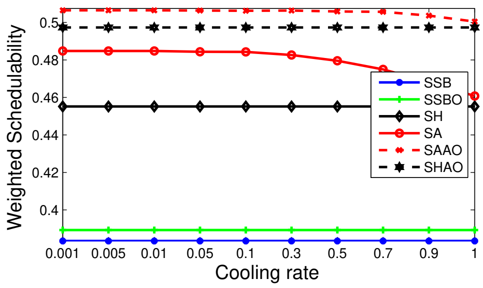

Figure 11

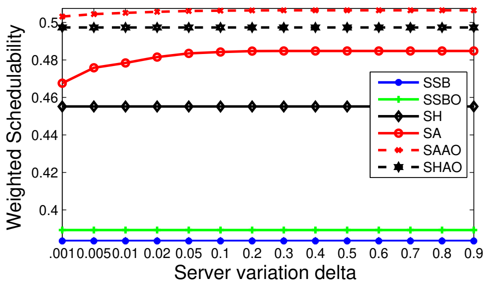

Figure 12

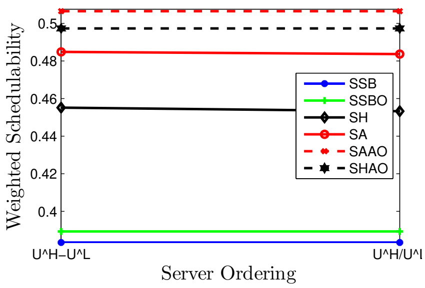

Figure 13

ratio of all heuristics is presented in Table 2. The change in server's budget can achieve up to 52.8% more in schedulability success ratio. For more insights, please see the schedulability success ratio plots in Appendix A.

7 Conclusions

In this paper, we proposed a new schedulability analysis for mixed-criticality uniprocessor systems executing periodic servers (varying budgets in response to mode switch) in a cyclic executive manner and use the AMC-max scheduling policy to schedule the tasks within each server. Our proposed approach provides strict temporal isolation among applications with additional ability to efficiently utilise the available execution capacity across mode switches. We also proposed number of heuristics that assigns budgets to servers in both modes and define the order of the servers in the cyclic executive schedule. The experimental evaluation with synthetic task-sets showed by varying budgets in response to mode switch improves schedulablity ratio by up to 52.8%, compared to the baseline static server budget algorithm. Even with a simple heuristic, we can achieve up to 27% of improvements in schedulability ratio. The order of the servers in the cyclic executive schedule has high impact on the schedulability ratio and the proposed heuristic to select the ordering of servers performs well in our experiments. In the future, we intend to extend this approach to multicore platforms and include the effect of other shared resources in the schedulability analysis.

References

1. Certification authorities software team (cast), position paper (cast-32a) multicore processors. Certification authorities in North and South America, Europe, and Asia, November 2016.
2. AERONAUTICAL RADIO, INC. Avionics Application Software Standard Interface, Part 1, Required Services, ARINC SPECIFICATION 653P1-3 edition, November 2010.
3. Muhammad Ali Awan, Konstantinos Bletsas, Pedro F. Souto, and Eduardo Tovar. Semi-partitioned mixed-criticality scheduling. In Proceedings of the 30th International Conference Architecture of Computing Systems, pages 205–218, 2017.
4. Muhammad Ali Awan and Stefan M. Petters. Intra-task device scheduling for real-time embedded systems. In Journal of Systems Architecture, 2013.
5. S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In Proceedings of the 32nd IEEE Real-Time Systems Symposium, pages 34–43, 2011.
6. Sanjoy Baruah and Alan Burns. Implementing mixed criticality systems in Ada. In 16th Ada-Europe Conference, pages 174–188, 2011.
7. Andrea Bastoni, Björn Brandenburg, and James Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. Proceedings of OSPERT, pages 33–44, 2010.
8. E. Bini and G.C. Buttazzo. Measuring the performance of schedulability tests. Journal of Real–Time Systems, 30(1-2):129–154, 2009.
9. Konstantinos Bletsas, Muhammad Ali Awan, Pedro F. Souto, Benny Akesson, Alan Burns, and Eduardo Tovar. Decoupling criticality and importance in mixed-criticality scheduling. In Proceedings of the 6th Workshop on Mixed-Criticality Systems, 2018.
10. A. Burns and R.I. Davis. Adaptive mixed criticality scheduling with deferred preemption. In Proceedings of the 35rd IEEE Real-Time Systems Symposium, pages 21–30, Dec 2014.
11. Alan Burns and Robert Davis. Mixed criticality systems – a review (11th edition). Technical report, Department of Computer Science, University of York, UK, Aug. 2018.
12. Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. ACM Computing Surveys, 50(6):82:1–82:37, November 2017.
13. Micaiah Chisholm, Namhoon Kim, Stephen Tang, Nathan Otterness, James H. Anderson, F. Donelson Smith, and Donald E. Porter. Supporting mode changes while providing hardware isolation in mixed-criticality multicore systems. In Proceedings of the 25th Conference Real-Time and Networked Systems, RTNS '17, pages 58–67, 2017.
14. Robert I. Davis and Alan Burns. Priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In Proceedings of the 30th IEEE Real-Time Systems Symposium, pages 398–409, 2009.
15. Pontus Ekberg and Wang Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. Journal of Real–Time Systems, 50(1):48–86, 2014.
16. C. Evripidou and A. Burns. Scheduling for mixed-criticality hypervisor systems in the automotive domain. In Proceedings of the 4th Workshop on Mixed-Criticality Systems, 2016.
17. Lipari Giuseppe and C. Buttazzo Giorgio. Resource reservation for mixed criticality systems. In Workshop on Real-Time Systems: The past, The present, and the future, March 2013.

18 X. Gu and A. Easwaran. Dynamic budget management with service guarantees for mixed-criticality systems. In Proceedings of the 37rd IEEE Real-Time Systems Symposium, pages 47–56, 2016.

19 Xiaozhe Gu, Arvind Easwaran, Kieu-My Phan, and Insik Shi. Resource efficient isolation mechanisms in mixed-criticality scheduling. In Proceedings of the 27th Euromicro Conference on Real-Time Systems, pages 13–24, 2015.

20 Fei Guan, Long Peng, Luc Perneel, Hasan Fayyad-Kazan, and Martin Timmerman. Adaptive reservation into mixed-criticality systems. Scientific Programming, 2017, 2017.

21 J. L. Herman, C. J. Kenna, M. S. Mollison, J. H. Anderson, and D. M. Johnson. Rtos support for multicore mixed-criticality systems. In Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium, pages 197–208, April 2012.

22 Biao Hu, Kai Huang, Gang Chen, Long Cheng, and Alois Knoll. Adaptive workload management in mixed-criticality systems. ACM Transactions on Embedded Computing Systems, 16, 10 2016. doi:10.1145/2950058.

23 Biao Hu, Lothar Thiele, Pengcheng Huang, Kai Huang, Christoph Griesbeck, and Alois Knoll. FFOB: Efficient online mode-switch procrastination in mixed-criticality systems. Journal of Real–Time Systems, 2018. doi:10.1007/s11241-018-9323-x.

24 Huang-Ming Huang, Christopher Gill, and Chenyang Lu. 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, pages 23–32, 2012.

25 Raj Jain. The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling. Wiley professional computing. Wiley, 1991.

26 M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. The Computer Journal, 29(5):390–395, 1986.

27 Namhoon Kim, Bryan C. Ward, Micaiah Chisholm, Cheng-Yang Fu, James H. Anderson, and F. Donelson Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. Journal of Real–Time Systems, 53(5):709–759, 2017.

28 S. Kirkpatrick, C. D. Gelatt, , and M. P. Vecchi. Optimization by simulated annealing. Science, 220:671—-680, 1983.

29 Yann-Hang Lee, K.P. Reddy, and C.M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In Proceedings of the 15th Euromicro Conference on Real-Time Systems, pages 105–112, Jul. 2003.

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

31 Alessandro Vittorio Papadopoulos, Enrico Bini, Sanjoy Baruah, and Alan Burns. AdaptMC: A control-theoretic approach for achieving resilience in mixed-criticality systems. In Proceedings of the 30th Euromicro Conference on Real-Time Systems, pages 14:1–14:22, 2018.

32 Alessandro Vittorio Papadopoulos, Martina Maggio, Alberto Leva, and Enrico Bini. Hard real-time guarantees in feedback-based resource reservations. Journal of Real–Time Systems, 51(3):221–246, 2015.

33 Jiankang Ren and Linh Thi Xuan Phan. Mixed-criticality scheduling on multiprocessors using task grouping. In Proceedings of the 27th Euromicro Conference on Real-Time Systems, pages 25–34, 2015.

34 Paulo Baltarejo Sousa, Konstantinos Bletsas, Eduardo Tovar, Pedro Souto, and Benny Åkesson. Unified overhead-aware schedulability analysis for slot-based task-splitting. Journal of Real–Time Systems, 50(5-6):680–735, 2014.

35 S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In Proceedings of the 28th IEEE Real-Time Systems Symposium, 2007.

Appendix A

The following figures present the schedulability ratio of the proposed heuristics with different parameters. The variation in a parameter is given in the label of a figure, while the other parameters are considered to be default.

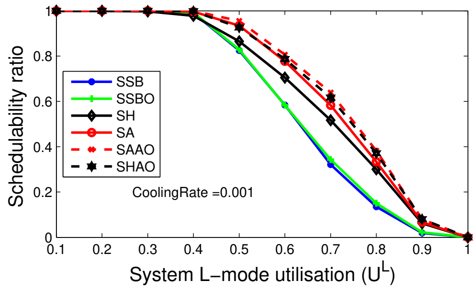

Figure 14 Cooling rate = 0.001

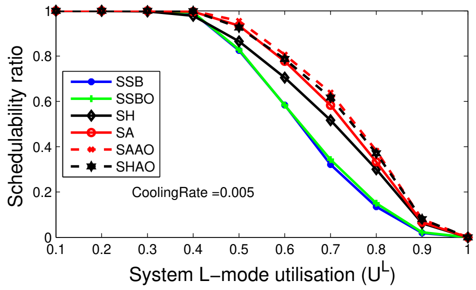

Figure 15 Cooling rate = 0.005

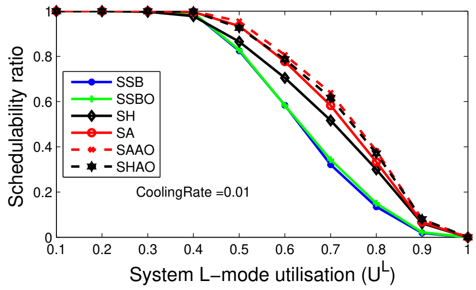

Figure 16 Cooling rate = 0.01

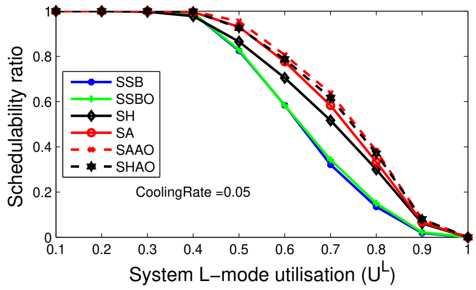

Figure 17 Cooling rate = 0.05

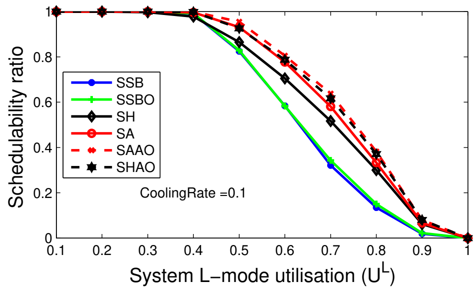

Figure 18 Cooling rate = 0.1

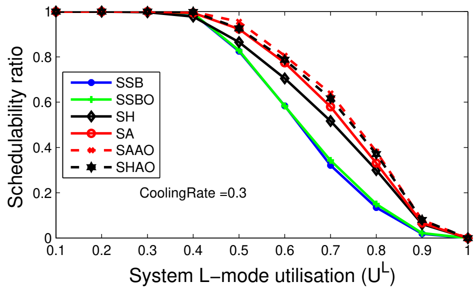

Figure 19 Cooling rate = 0.3

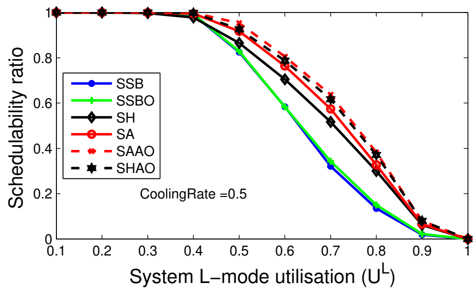

Figure 20 Cooling rate = 0.5

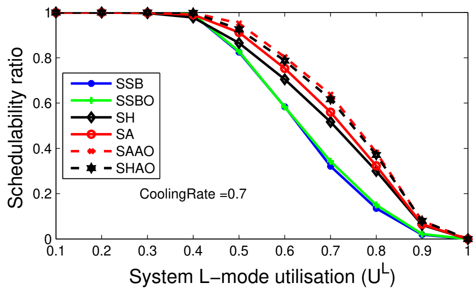

Figure 21 Cooling rate = 0.7

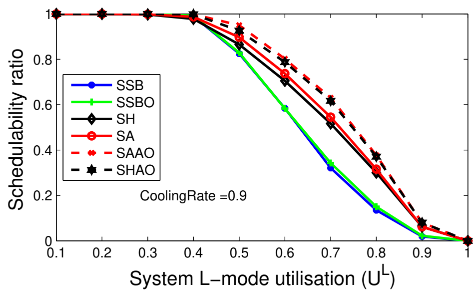

Figure 22 Cooling rate = 0.9

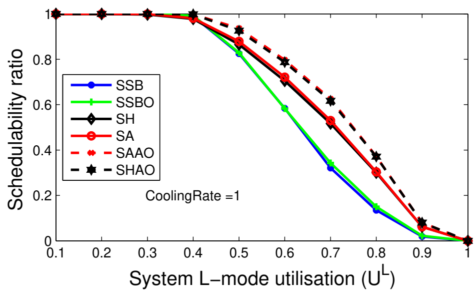

Figure 23 Cooling rate = 1

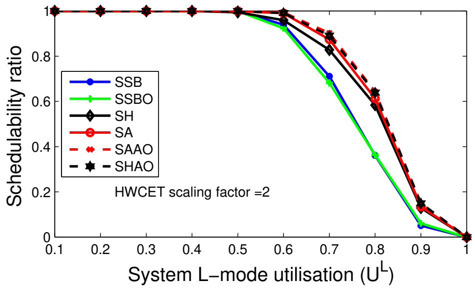

Figure 24 HWCET scaling = 2

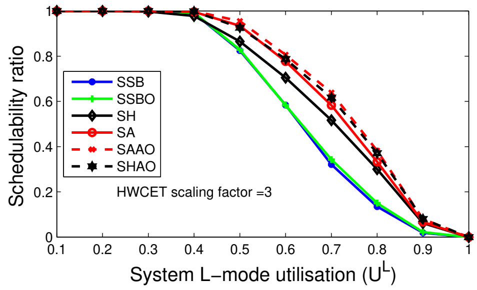

Figure 25 HWCET scaling = 3

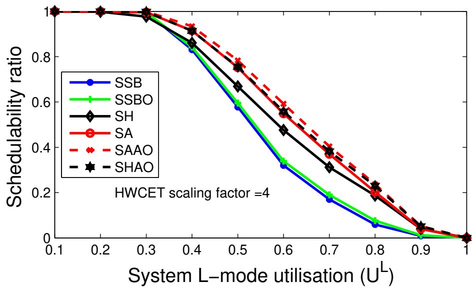

Figure 26 HWCET scaling = 4

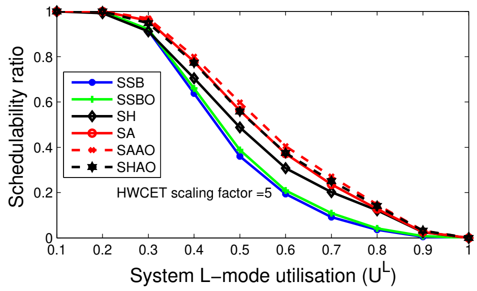

Figure 27 HWCET scaling = 5

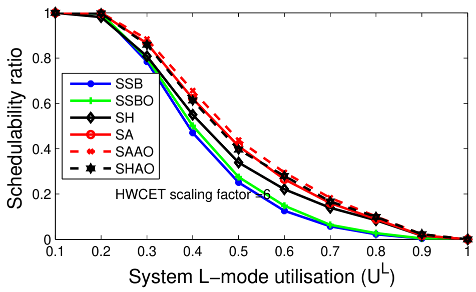

Figure 28 HWCET scaling = 6

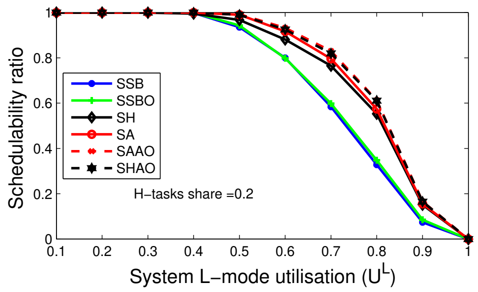

Figure 29 H-tasks share = 20%

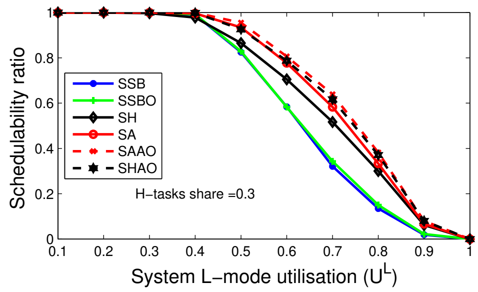

Figure 30 H-tasks share = 30%

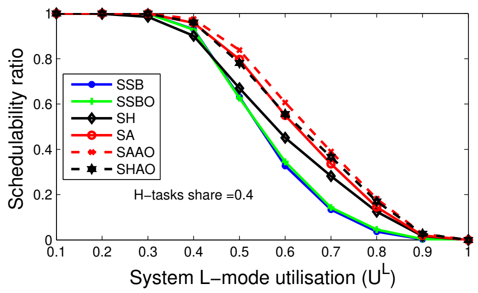

Figure 31 H-tasks share = 40%

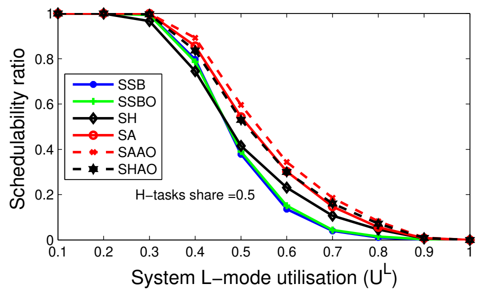

Figure 32 H-tasks share = 50%

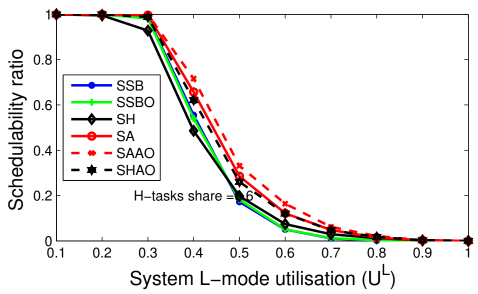

Figure 33 H-tasks share = 60%

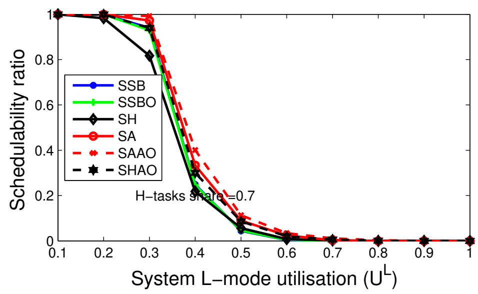

Figure 34 H-tasks share = 70%

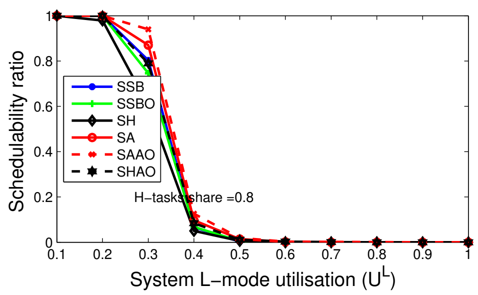

Figure 35 H-tasks share = 80%

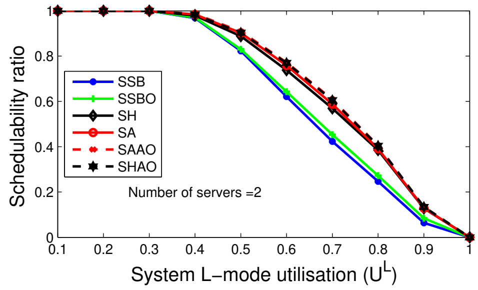

Figure 36 Fixed tasks per server, Servers = 2

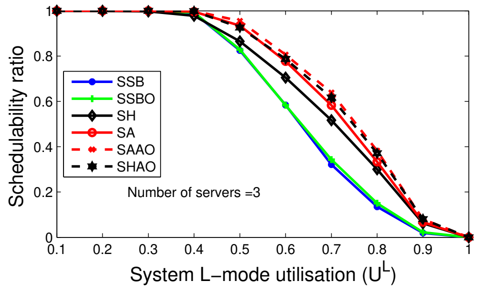

Figure 37 Fixed tasks per server, Servers = 3

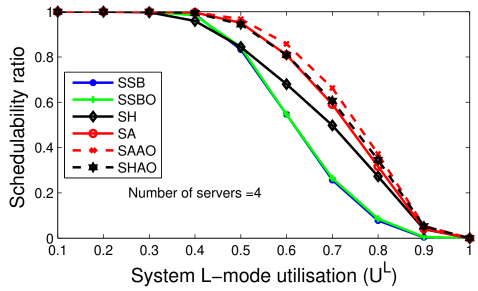

Figure 38 Fixed tasks per server, Servers = 4

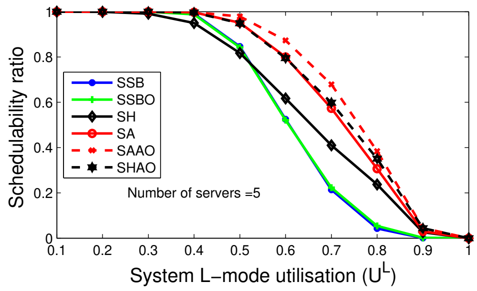

Figure 39 Fixed tasks per server, Servers = 5

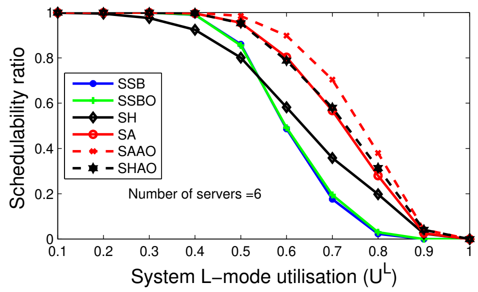

Figure 40 Fixed tasks per server, Servers = 6

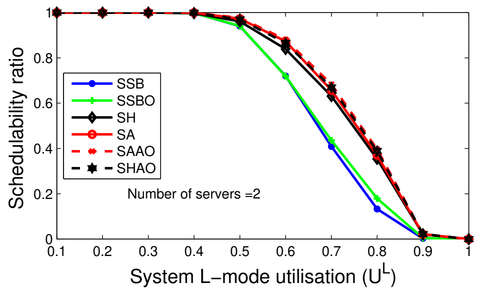

Figure 41 Fixed task-set size, servers = 2

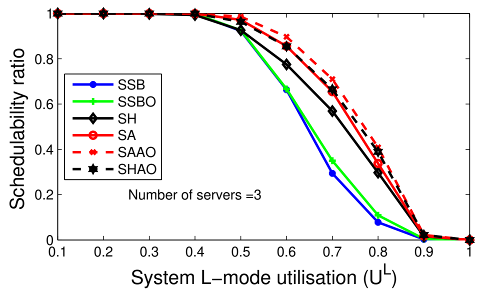

Figure 42 Fixed task-set size, servers = 3

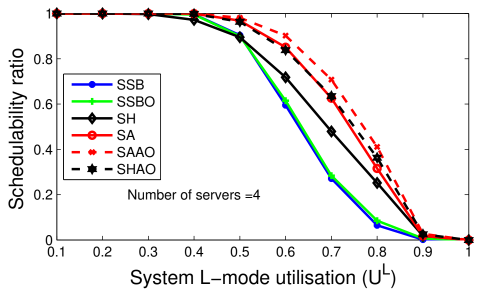

Figure 43 Fixed task-set size, servers = 4

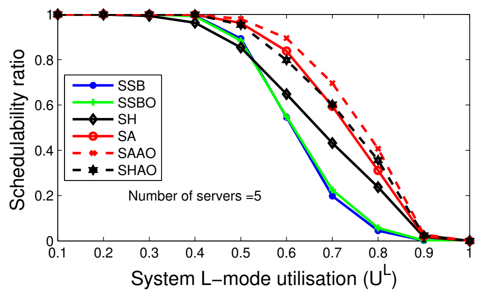

Figure 44 Fixed task-set size, servers = 5

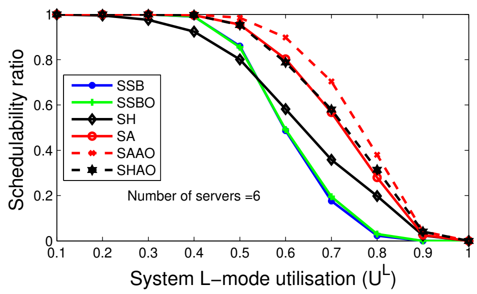

Figure 45 Fixed task-set size, servers = 6

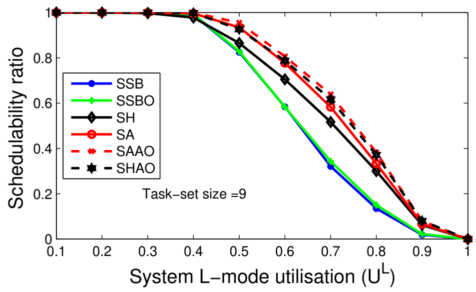

Figure 46 Task-set size = 9

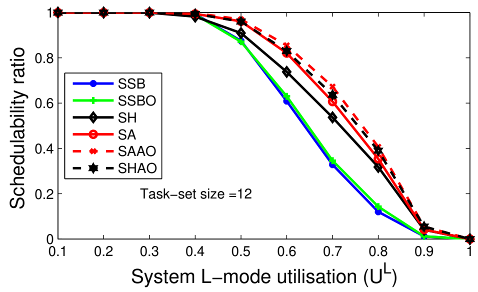

Figure 47 Task-set size = 12

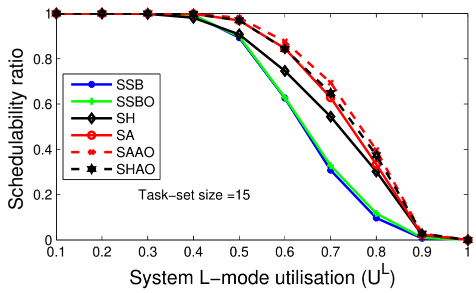

Figure 48 Task-set size = 15

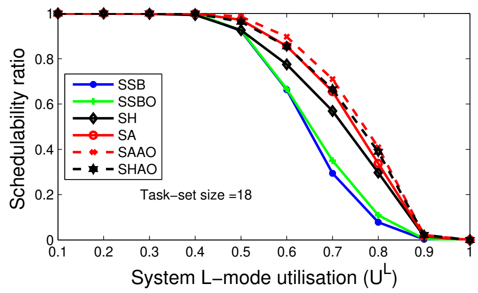

Figure 49 Task-set size = 18

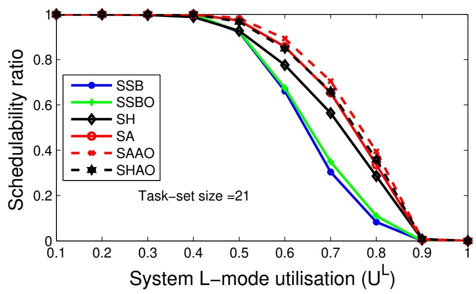

Figure 50 Task-set size = 21

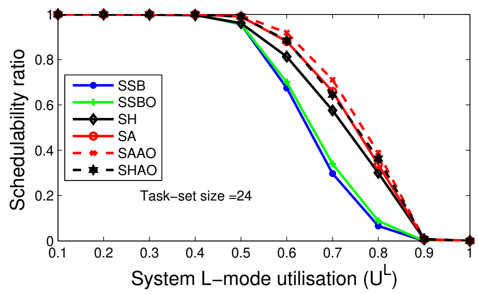

Figure 51 Task-set size = 24

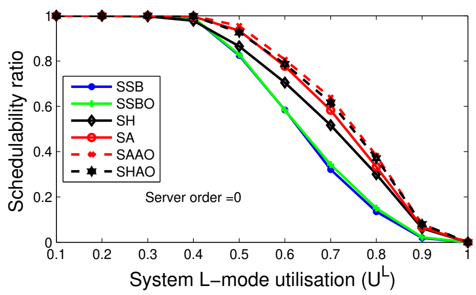

Figure 52 Server order = $U^H(\tilde{P}_i) - U^L(\tilde{P}_i)$

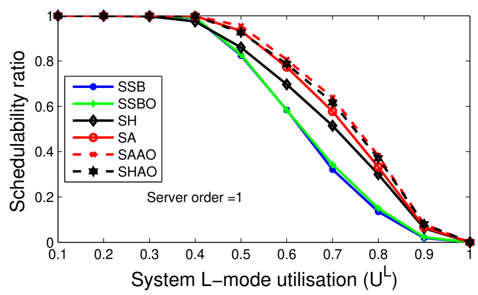

Figure 53 Server order = $U^H(\tilde{P}_i)/U^L(\tilde{P}_i)$

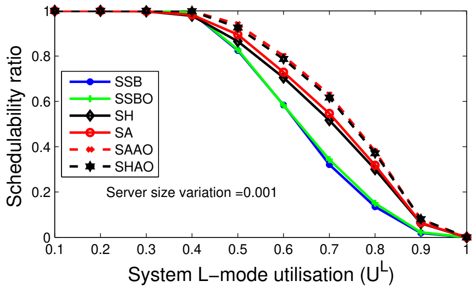

Figure 54 Server size variation $\beta = 0.001$

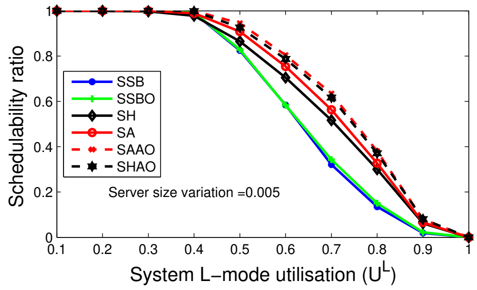

Figure 55 Server size variation $\beta = 0.005$

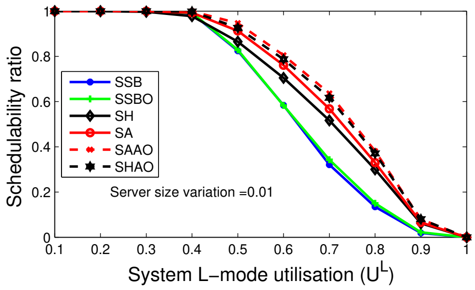

Figure 56 Server size variation $\beta = 0.01$

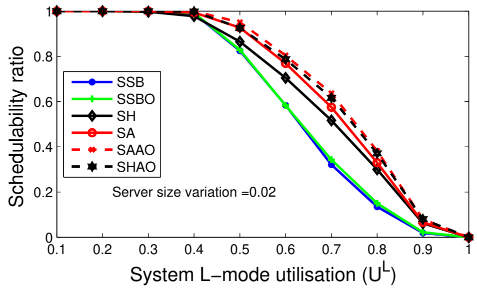

Figure 57 Server size variation $\beta = 0.02$

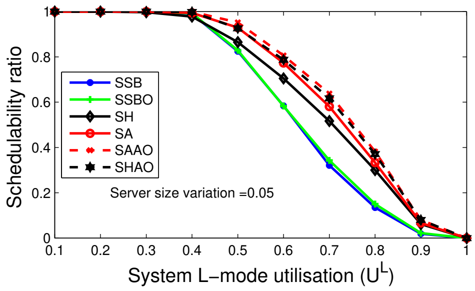

Figure 58 Server size variation $\beta = 0.05$

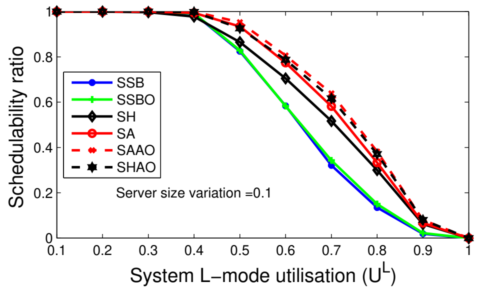

Figure 59 Server size variation $\beta = 0.1$

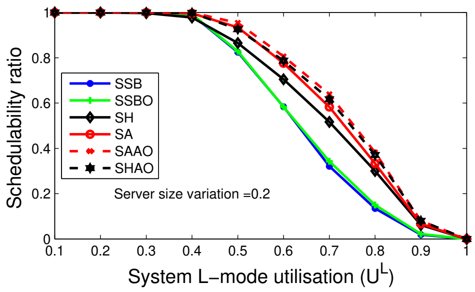

Figure 60 Server size variation $\beta = 0.2$

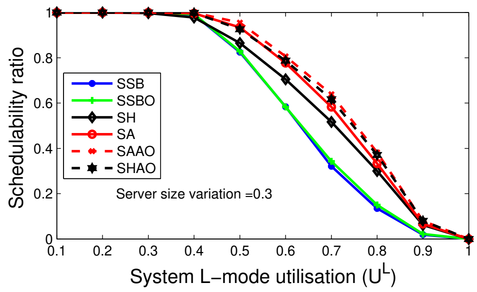

Figure 61 Server size variation $\beta = 0.3$

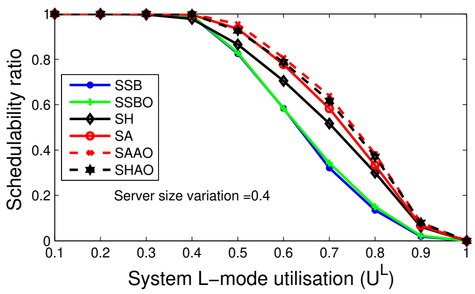

Figure 62 Server size variation $\beta = 0.4$

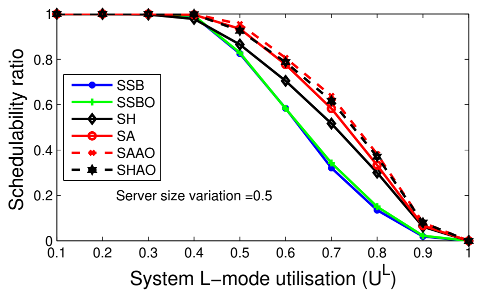

Figure 63 Server size variation $\beta = 0.5$

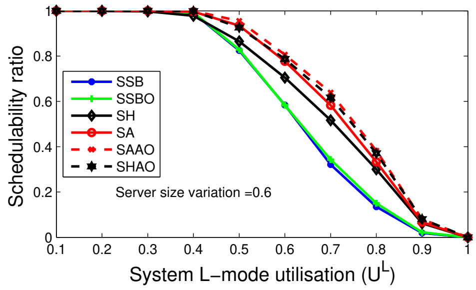

Figure 64 Server size variation $\beta = 0.6$

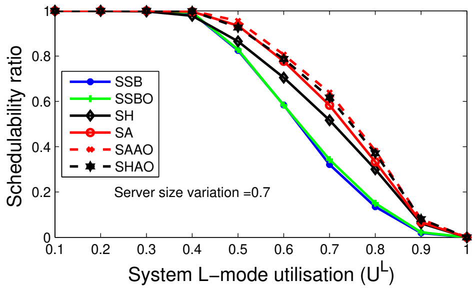

Figure 65 Server size variation $\beta = 0.7$

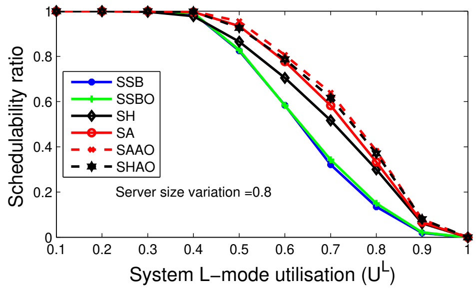

Figure 66 Server size variation $\beta = 0.8$

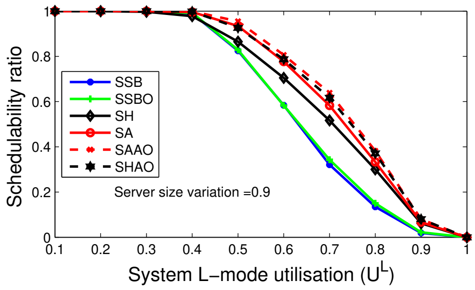

Figure 67 Server size variation $\beta = 0.9$

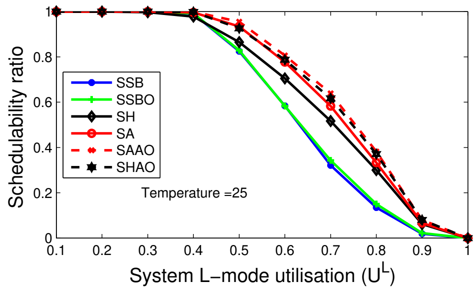

Figure 68 Temperature = 25

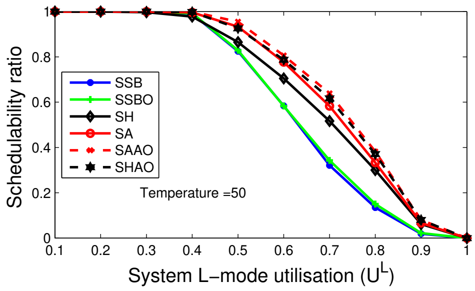

Figure 69 Temperature = 50

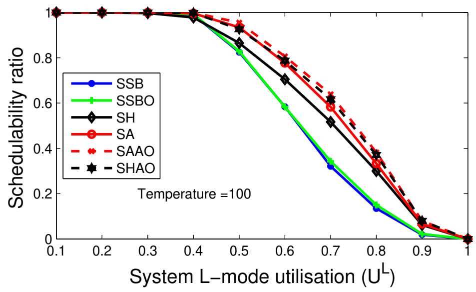

Figure 70 Temperature = 100

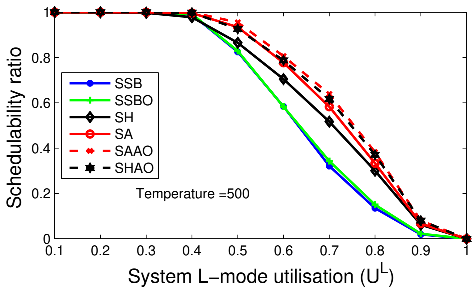

Figure 71 Temperature = 500

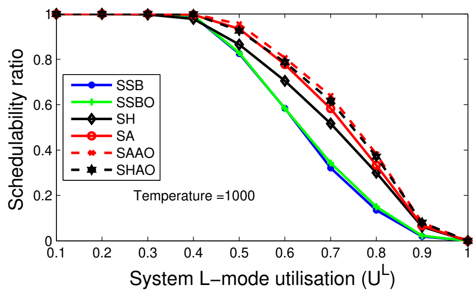

Figure 72 Temperature = 1000

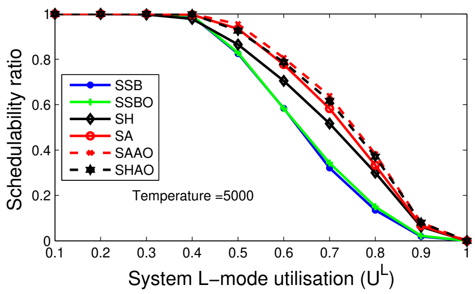

Figure 73 Temperature = 5000

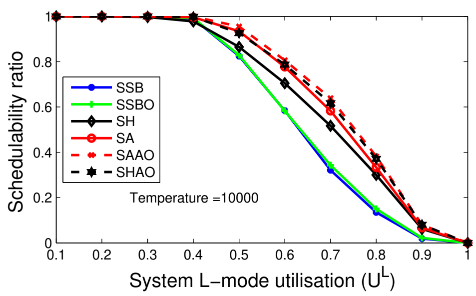

Figure 74 Temperature = 10000