CISTER
Research Centre in
Real-Time & Embedded
Computing Systems

Conference Paper

Semi-partitioned mixed-criticality scheduling

Muhammad Ali Awan*
Konstantinos Bletsas*
Pedro Souto*
Eduardo Tovar*

CISTER_TR_161102

2017/04/03

Semi-partitioned mixed-criticality scheduling

Muhammad Ali Awan*, Konstantinos Bletsas*, Pedro Souto*, Eduardo Tovar*

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail: muaan@isep.ipp.pt, ksbs@isep.ipp.pt, pfs@fe.up.pt, emt@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Scheduling isolation in mixed-criticality systems is challenging without sacrificing performance. In response, we propose a scheduling approach that combines server-based semi-partitioning and deadline scaling. Semi-partitioning (whereby only some tasks migrate, in a carefully managed manner), hitherto used in single criticality systems, offers good performance with low overheads. Deadline-scaling selectivelyprioritizes high-criticality tasks in parts of the schedule to ensure theirdeadlines are met even in rares case of execution time overrun. Ournew algorithm NPS-F-MC brings semi-partitioning to mixed-criticalityscheduling and uses Ekberg and Yi 19s state-of-the-art deadline scaling approach. It ensures scheduling isolation among different-criticality tasksand only allows low-criticality task migration. We also explore variantsthat disallow migration entirely or relax the isolation between differentcriticalities (SP-EKB) in order to evaluate the performance tradeoffs associated with more flexible or rigid safety and isolation requirements.

Semi-partitioned mixed-criticality scheduling

Muhammad Ali Awan*+, Konstantinos Bletsas*+, Pedro F. Souto‡* and Eduardo Tovar*+

*CISTER/INESC-TEC Research Centre, Porto, Portugal
+ISEP/IPP, Porto ‡Faculty of Engineering, University of Porto

Abstract. Scheduling isolation in mixed-criticality systems is challenging without sacrificing performance. In response, we propose a scheduling approach that combines server-based semi-partitioning and deadline-scaling. Semi-partitioning (whereby only some tasks migrate, in a carefully managed manner), hitherto used in single criticality systems, offers good performance with low overheads. Deadline-scaling selectively prioritise high-criticality tasks in parts of the schedule to ensure their deadlines are met even in rares case of execution time overrun. Our new algorithm NPS-F-MC brings semi-partitioning to mixed-criticality scheduling and uses Ekberg and Yi's state-of-the-art deadline scaling approach. It ensures scheduling isolation among different-criticality tasks and only allows low-criticality task migration. We also explore variants that disallow migration entirely or relax the isolation between different criticalities (SP-EKB) in order to evaluate the performance tradeoffs associated with more flexible or rigid safety and isolation requirements.

1 Introduction

Many real-time embedded systems (automotive, avionics, aerospace) host functions of different *criticalities*. A deadline miss by a high-criticality function can be disastrous, but losing a low-criticality function only moderately affects the quality of service. Scalability and cost concerns favor *mixed-criticality* (MC) systems, whereby tasks of different criticalities are scheduled on the same processors but this brings challenges: Lower-criticality tasks interfering unpredictably with higher-criticality tasks can be catastrophic. Conversely, rigidly prioritisation by criticality leads to inefficient processor usage. Therefore, we seek (i) efficient use of processing capacity and (ii) schedulability guarantees for all tasks under typical conditions subject to (iii) ensured schedulability of high-criticality tasks in all cases. Most related works [1] use Vestal's model [2], which views the system operation as different modes (low- and high-criticality) and associates different worst-case task execution times (WCETs) in each mode with a corresponding degree of confidence. This is because the cost of provably safe WCET estimation (and the associated pessimism) is justified only for high-criticality tasks. Other tasks have less rigorous WCET estimates, which *might* be exceeded, very rarely.

We also adopt Vestal's model, with two criticality levels. Our main contribution is **NPS-F-MC**, an extension of the semi-partitioned scheduling algorithm

NPS-F [3] to mixed criticalities. NPS-F is *server-based*, which helps provide both fairness to low-criticality tasks and strict *temporal isolation* between high- and low-criticality tasks. The new algorithm employs the *per-task deadline scaling* scheduling technique by Ekberg and Yi [4], an extension of EDF-VD [5]. NPS-F-MC allows migration among processors only for servers for low-criticality tasks, with less severe safety considerations. However, given the conservative stance of certification authorities [6] towards task migrations, we formulate as another contribution a fully partitioned variant (NPS-F-IMA) and explore the performance gap from disallowing migrations entirely. As third contribution we explore the performance penalty from the strict temporal isolation by NPS-F-MC by formulating new partitioned (P-EKB) and semi-partitioned (SP-EKB) extensions of the (uniprocessor) algorithm of Ekberg and Yi, and comparing with those.

2 Overview

Task model [2] The system can be in either low- or high-criticality mode (L-mode or H-mode). A task is of either low or high criticality (L-task or H-task). Each H-task has two different WCET estimates: the one for the L-mode of operation (L-WCET), is *deemed* safe but lacks proof, whereas the one for the H-mode (H-WCET) is provably safe but usually much greater. Each L-task only has an L-WCET. The default system mode is L, but if *any* task exceeds its L-WCET, the system immediately switches into H-mode: all L-tasks are abandoned and only H-tasks remain. In H-mode, all H-tasks (incl. instances present at the time of the mode switch) are pessimistically assumed to execute for up to their H-WCET. Even so, it must be provable that no H-task deadlines can be missed.

MC scheduling with scaled deadlines Deadline-scaling for mixed-criticality systems originates with EDF-VD ("Earliest Deadline First - with Virtual Deadlines") [5]. EDF-VD uses standard EDF scheduling rules but, instead of reporting the real deadlines to the EDF scheduler for scheduling decisions, it reports shorter deadlines (if needed) for H-tasks during L-mode operation. This helps with the schedulability of H-tasks in the case of a switch to H-mode, because it prioritises H-tasks more than conventional EDF would, over parts of the schedule. This allows them to be sufficiently "ahead of schedule" and catch up with their true deadlines if any task overruns its L-WCET. While in H-mode, the true H-task deadlines are used for scheduling and L-tasks are dropped. EDF-VD proportionately shortens the H-task deadlines according to a single common scale factor and its schedulability test considers the task utilisations in both modes.

Ekberg and Yi [4] improved upon EDF-VD by enabling and calculating distinct scale factors for different H-tasks and using a more precise *demand bound function* (dbf) based schedulability test [7]. This improves performance. The calculation of the scale factors is an iterative task-by-task process. For details, see [4][8]. Recently, Masrur et al. [9] proposed using just two scale factors, to balance scheduling performance and computational complexity. A higher scale factor is used for tasks with an H-WCET/L-WCET ratio above some threshold.

Meanwhile, Easwaran developed a test [10] that dominates [4]. In this work we innovatively combine the deadline-scaling technique by Ekberg and Yi [4] with the existing NPS-F semi-partitioned scheduling algorithm.

Semi-partitioning and NPS-F Under semi-partitioned scheduling, most tasks are partitioned; the rest may migrate, in a carefully managed manner. This allows for efficiently utilising a multicore, without many preemptions, migrations and overheads, under strong schedulability guarantees. Semi-partitioned mixed-criticality scheduling was first proposed in [11]. Here, we conclude that work and adapt the NPS-F algorithm [3] (originally, for single-criticality systems).

Classic NPS-F assigns tasks via bin-packing, not directly to processors but to periodic fixed-budget servers using EDF as their internal scheduling policy. The servers are mapped to the available processors in a form of cyclic executive with a periodicity of S – the "timeslot length". Each server is either implemented as either one periodic "reserve" (fixed-length contiguous time window) on one processor or as multiple periodic reserves on different processors. A given server's tasks can only execute within its reserves, which in turn are exclusively used by those tasks. A server (its reserves) is appropriately sized, to ensure that its tasks meet all their deadlines at run-time. Additionally, the reserves of a server mapped to multiple processors must never overlap in time.

3 System model

We assume a set of n sporadic tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i has a minimum inter-arrival time T_i, a relative deadline $D_i \leq T_i$, a criticality level $\kappa_i \in \{L, H\}$ (low or high, respectively) and two WCET estimates, C_i^L and C_i^H, one for each mode. The subsets of L-tasks and H-tasks in τ are $\tau(L) \stackrel{\text{def}}{=} \{\tau_i \in \tau | \kappa_i \in L\}$ and $\tau(H) \stackrel{\text{def}}{=} \{\tau_i \in \tau | \kappa_i \in H\}$. It is assumed that $\forall \tau_i \in \tau(H)$, $C_i^L \leq C_i^H$ and $\forall \tau_i \in \tau(L)$, $C_i^H = 0$. Tasks in $\tau(H)$ are not allowed to migrate among processors. The utilisation of τ_i is $U_i^L \stackrel{\text{def}}{=} \frac{C_i^L}{T_i}$ and $U_i^H \stackrel{\text{def}}{=} \frac{C_i^H}{T_i}$ respectively in each mode. The system utilisation in each mode is $U^H \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau(H)} U_i^H$ and $U^L \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i^L$. Our platform $P \stackrel{\text{def}}{=} \{P_1, P_2, \ldots, P_m\}$ has m identical processors. We assume a set $\widetilde{P}$ of m'' servers, indexed $\widetilde{P}_1$ to $\widetilde{P}_{m''}$, with m'' not *a priori* defined.

During task assignment, the processing capacity of each server is equivalent to that of a physical processor. The set of tasks assigned to server $\widetilde{P}_k$ is denoted as $\tau(\widetilde{P}_k)$. Each server is only assigned tasks of the same criticality as each other. A server that contains only H-tasks is termed a H-server. Similarly, an L-server contains only L-tasks. The budget of a $\widetilde{P}_k$ is denoted by $X_{\widetilde{P}_k}$.

4 Task assignment, scheduling model and timing analysis

Overview NPS-F-MC partitions the set of H-tasks ($\tau(H)$) over $m_H^{''}$ non-migrating H-servers. Each H-server will be assigned to a different corresponding processor, so it must hold that $m_H^{''} \leq m$ or the algorithm will declare failure.

The set of L-tasks ($\tau(L)$) is partitioned over a separate set of L-servers. The "leftover" parts of the timeslots, that remain on the m processors, after the assignment and sizing of the non-migrating H-servers, are reclaimed from the processors for the mapping of the L-servers.

During L-mode operation, all tasks are scheduled within the respective servers under EDF. But if a task τ_i overruns its C_i^L (which triggers a mode switch), then all L-tasks are immediately dropped along with the server arrangement altogether, and the system switches to pure partitioned EDF scheduling of the H-tasks. This raises the question of how to specify the server budgets:

A **naive approach** would (i) partition the H-tasks to the H-servers using a uniprocessor EDF schedulability test that considers the overly conservative estimates C_i^H, to meet deadlines in H-mode, and (ii) assign budgets to the respective servers so that the H-tasks provably meet their deadlines in L-mode, as long as they all execute for up to their respective C_i^L. However, this may lead to missed deadlines during the mode transition.

A **conservative approach** would instead consider the C_i^H estimates, when sizing the H-servers for operation in L-mode. However, this approach decreases the processing capacity available for L-tasks and is inefficient.

Ideally, one should therefore set the server budgets to the optimal intermediate value that minimises the processing capacity used for H-servers (i.e., maximises the capacity available for L-servers) in L-mode without jeopardising the schedulability of the H-tasks at any point in time (even when a mode transition occurs). As part of this work, we identify how to compute these optimal H-server budgets, using the analysis of Ekberg and Yi [4].

In summary, a processor P_p with an H-server assigned to it is equivalently modelled as a separate uniprocessor system, whereupon a transformed task subset runs under EDF with deadline scaling. This consists of all H-tasks assigned to the single H-server $\widetilde{P}_p$ mapped to P_p plus a single "fake" L-task with parameters $(C_{\text{fake}}, D_{\text{fake}}, T_{\text{fake}}) \stackrel{\text{def}}{=} (S - X_{\widetilde{P}_p}, S - X_{\widetilde{P}_p}, S)$, where $X_{\widetilde{P}_p}$ is the budget of $\widetilde{P}_p$. This zero-laxity fake task equivalently represents the periodic unavailability of the processor, for the tasks of $\widetilde{P}_p$ to execute on. The budget $X_{\widetilde{P}_p}$ is then set to the minimum value for which the transformed MC task subset is schedulable on a uniprocessor, using the deadline-scaling by Ekberg and Yi.

In detail. The proposed approach (outlined in pseudocode as Algorithm 1), is divided into three offline stages, (i) task-to-server assignment, (ii) sizing servers ("inflating", in NPS-F jargon) and (iii) mapping servers to processors.

The first stage assigns H-tasks to servers via First-Fit (FF) bin-packing, subject to an exact uniprocessor EDF schedulability test, from classic (criticality-

oblivious) EDF theory [7] that uses the respective H-WCETs as input. The L-tasks are assigned to a different set of servers via First-Fit, using the same test, but using their L-WCETs as inputs.

The "inflated utilisation" $U^{infl.} \stackrel{\text{def}}{=} \frac{X_{\widetilde{P}_p}}{S}$ of each server is computed in the second stage. The sum of inflated utilisations of all servers corresponds to the total processing capacity (informally, the number of processors) required for successfully scheduling the given task set under the proposed approach. Finally, in the third stage, servers are mapped to physical processors and their periodic reserves are arranged to avoid time-overlaps of reserves belonging to same server.

(i) Task-to-server mapping: Initially (Algorithm 1, lines 1-5), the H-tasks are assigned to servers (as many as needed) using First-Fit, assuming their C_i^H WCET estimate and according to an exact uniprocessor (single-criticality) EDF schedulability test. The $m_H^{''}$ servers $\widetilde{P}_1$ to $\widetilde{P}_{m_H^{''}}$ thus formed, are all H-servers.

Algorithm 2 presents the First-Fit bin-packing routine that assigns tasks to servers. The exact uniprocessor EDF schedulability test employed therein makes use of the *demand bound function* [7]. It is an abstraction of the computational requirements of the tasks. The demand of an arbitrary-deadline task τ_i over any possible time interval of length t, denoted by $\text{DBF}(\tau_i, t, \kappa)$, is a tight (i.e., exact and least) upper bound on the maximum cumulative execution requirement of jobs by τ_i over a time interval of length t; the additional argument $\kappa \in L, H$ denotes whether the WCET assumed for those jobs is C_i^L or C_i^H (see Equation 1). The DBF for a set of tasks and the corresponding schedulability condition are given by Equations 2 and 3, respectively.

$$\forall t \geq 0, \quad \text{DBF}(\tau_i, t, \kappa) \stackrel{\text{def}}{=} \max\left(0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i^{\kappa} \tag{1}$$

$$\text{DBF}(\tau, t, \kappa) = \sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t, \kappa) \tag{2}$$

$$\forall t \geq 0, \quad \text{DBF}(\tau, t, \kappa) \leq t \tag{3}$$

The schedulability of a server is tested with the following expression: $\forall t > 0, \quad \text{DBF}(\tau(\widetilde{P}_k), t, \kappa) \leq t$. If the test succeeds, a provisional assignment is made

Algorithm 1 Semi-partitioned MC algorithm (NPS-F-MC)

Input: τ, P ▷ assuming $U^L \leq 1$ and $U^H \leq 1$
Output: Schedulability status

1: **for** $(i := 1; i \leq n; i++)$ **do**
2: **if** $(\kappa_i == H)$ **then**
3: Assign_To_Server_FF (τ_i, H)
4: **end if**
5: **end for**
6: $m_H^{''} :=$ Index_Of_Highest_Populated_Server()
▷ $m_H^{''}$ is the number of H-servers
7: **if** $(m_H^{''} > m)$ **then return** FAILURE
8: **end if**
9: **for** $(i := 1; i \leq n; i{+}{+})$ **do**
10: **if** $(\kappa_i == L)$ **then**
11: Assign_To_Server_FF (τ_i, L)
12: **end if**
13: **end for**
14: $m^{''} :=$ Index_Of_Highest_Populated_Server()
▷ $m^{''}$ is the total number of servers
15: $U_{tot}^{infl} := 0$
16: **for** $(q := 1; q \leq m^{''}; q{+}{+})$ **do**
17: $U_{\widetilde{P}_q}^{infl} :=$ Mixed_Criticality_Inflate$(\widetilde{P}_q)$
18: $U_{tot}^{infl} := U_{tot}^{infl} + U_{\widetilde{P}_q}^{infl}$
19: **if** $(U_{tot}^{infl} > m)$ **then return** FAILURE
20: **end if**
21: **end for**
22: Do_Server_To_Processor_Mapping(Inflated Servers)
23: **return** SUCCESS

Algorithm 2 Assign_To_Server_FF(τ_i, κ_i)

Input: τ_i, κ_i **Output:** Assignment status

1: **if** ($\kappa_i == H$) **then**
2: $q := 1$
3: **else**
4: $q := m''_H + 1$
5: **end if**
6: **while** (true) **do**
7: **if** (DBF($\tau(\widetilde{P}_q) \cup \{\tau_i\}, t, \kappa_i) \leq t,\ \forall t > 0$)
8: **then** $\tau(\widetilde{P}_q) := \tau(\widetilde{P}_q) \cup \{\tau_i\}$
9: BREAK
10: **else**
11: $q := q + 1$
12: **end if**
13: **end while**
14: **return** SUCCESS

Algorithm 3 Mixed_Criticality_Inflate($\widetilde{P}_k$)

Input: $\widetilde{P}_k$ **Output:** $U^{infl}_{\widetilde{P}_k}$

1: $X_{min} := 0$; $X_{max} := S = \frac{DT_{min}}{\delta}$;
2: **while** ($X_{max} - X_{min} > \Delta$) **do**
3: $X := \frac{X_{min} + X_{max}}{2}$
4: Create $\tau_{\text{fake}} = \langle T_{\text{fake}} := S, D_{\text{fake}} := S - X,$
$C_{\text{fake}} := S - X, L \rangle$
5: $\tau(\widetilde{P}_k) := \tau(\widetilde{P}_k) \cup \{\tau_{\text{fake}}\}$
6: **if** ($\widetilde{P}_k$ == H-server &&
Ekberg_Analysis($\tau(\widetilde{P}_k)$) == SUCCESS)
7: **then** $X_{max} := \frac{X_{min} + X_{max}}{2}$
8: **else if** ($\widetilde{P}_k$ == L-server &&
($\forall t > 0,\ \text{DBF}(\tau(\widetilde{P}_k), t, L) \leq t$))
9: **then** $X_{max} := \frac{X_{min} + X_{max}}{2}$
10: **else**
11: $X_{min} := \frac{X_{min} + X_{max}}{2}$
12: **end if**
13: $\tau(\widetilde{P}_k) := \tau(\widetilde{P}_k) \setminus \{\tau_{\text{fake}}\}$
14: **end while**
15: $X_{\widetilde{P}_k} := X_{max}$
16: **return** $U^{infl}_{\widetilde{P}_k} := \frac{X_{\widetilde{P}_k}}{S}$

permanent. The task-to-server assignment procedure *per se* always succeeds, because we are not *a priori* bounded to any particular number of servers; we can create/populate as many servers as needed (and, at worst, a task will be the first task assigned to a newly populated server). As mentioned, when assigning H-tasks, we assume $\kappa = H$. To speed up the computation, we use the improved Quick Processor Demand analysis (QPA*) by Zhang and Burns [12].

After all H-tasks are assigned to H-servers, if the number of H-servers (m''_H) exceeds the number of processors (m), then NPS-F-MC declares failure (see lines 6-8 in Algorithm 1). This reflects the real-world requirement that each H-server be mapped to only one processor and not allowed to migrate at run-time, because those tasks are critical and their scheduling should be as predictable as possible.

Afterwards, the L-tasks are assigned to the L-servers, indexed $\widetilde{P}_{m''_H+1}$ and upwards, by the same bin-packing procedure, but using their C^L_i WCET estimates for the schedulability test guiding the assignments (Algorithm 1, lines 9-13). Once this is done, m'' servers have been created and populated with tasks: of these, servers $\widetilde{P}_1$ to $\widetilde{P}_{m''_H}$ are H-servers and $\widetilde{P}_{m''_H+1}$ to $\widetilde{P}_{m''}$ are L-servers.

(ii) Sizing servers: The second step of the offline phase performs the server sizing (see Algorithm 1, lines 14-21). The timeslot length S (i.e., the period of all servers) is defined as $S \stackrel{\text{def}}{=} DT_{min}/\delta$ where DT_{min} is the shortest inter-arrival

time or relative deadline of all tasks and δ is a positive integer (usually $\delta = 1$)[1]. Let $X_{\widetilde{P}_k}$ denote the fixed time budget of server $\widetilde{P}_k$. Then, the system utilisation consumed by the server (i.e., its "inflated utilisation", in NPS-F jargon) is:

$$U_{\widetilde{P}_k}^{infl} \stackrel{\text{def}}{=} \frac{X_{\widetilde{P}_k}}{S} \tag{4}$$

The value of $X_{\widetilde{P}_k}$ is computed for each server by the function presented in Algorithm 3. Assume that a contiguous time window $X \leq S$ denotes the time that server $\widetilde{P}_k$ is active within a given timeslot of length S. The remaining fraction of the timeslot, consisting of a time window of length $S - X$ wherein $\widetilde{P}_k$ is inactive, is modelled as an interfering periodic zero-laxity *fake L-task* with the following attributes: $\tau_{\text{fake}} \stackrel{\text{def}}{=} \langle T_{\text{fake}} = S, D_{\text{fake}} = S - X, C_{\text{fake}} = S - X, L \rangle$. This standard task set transformation technique, for analytical convenience, was first used (for single-criticality workloads) for NSP-F server sizing by Souza et al. and is explained in [13], p. 702. This conceptual fake task along with the real tasks mapped to server $\widetilde{P}_k$, i.e., $\tau(\widetilde{P}_k) \cup \{\tau_{\text{fake}}\}$ are tested with the mixed-criticality schedulability uniprocessor analysis of Ekberg and Yi [4]. This analysis scales the deadlines of H-tasks (if needed) to make the task set schedulable in both H- and L-mode. If the analysis succeeds, the scaled H-task deadlines are output.

Computing $X_{\widetilde{P}_k}$ is an iterative process whose objective is to minimise the value of X (duration of periodic reserve allocated to $\widetilde{P}_k$). This minimum value of X that works corresponds to the optimal value for $X_{\widetilde{P}_k}$. To obtain it we iteratively sample the interval $X \in [0, S]$ using binary search and applying the test of Ekberg and Yi at each iteration, until the desired level of precision. Note that for each feasible value of X, Ekberg and Yi's algorithm could output different task deadline scale factors, in the general case.

Similarly, we compute the server budget $X_{\widetilde{P}_k}$ for each L-server $\widetilde{P}_k$. This is a simpler procedure because L-servers are only active in L-mode. So, there is no need to use the mixed-criticality schedulability test of Ekberg an Yi; the standard (single-criticality) optimal server sizing method for NPS-F is used [13] instead. Again, the attributes of a fake task are computed in a similar way, i.e., $\tau_{\text{fake}} \stackrel{\text{def}}{=} \langle T_{\text{fake}} = S, D_{\text{fake}} = S - X, C_{\text{fake}} = S - X, L \rangle$. The total demand of an L-server $\widetilde{P}_k$ along with the fake task τ_{fake} is given as follows:

$\text{DBF}(\tau(\widetilde{P}_k) \cup \{\tau_{\text{fake}}\}, t, L) = \text{DBF}(\tau(\widetilde{P}_k), t, L) + \text{DBF}(\tau_{\text{fake}}, t, L)$.

If $\forall t > 0, \;\; \text{DBF}(\tau(\widetilde{P}_k) \cup \{\tau_{\text{fake}}\}, t, L) \leq t$, then this server is schedulable with a budget of X. As in the case of H-servers discussed previously, Algorithm 3 minimises the value of X. The process of computing $\forall t > 0, \;\; \text{DBF}(\tau(\widetilde{P}_k) \cup \{\tau_{\text{fake}}\}, t, L) \leq t$ can be sped up with the QPA* algorithm.

If the L-servers are allowed to migrate among different processors then the task set is schedulable if the sum of inflated utilisations of all servers does not exceed m, the number of processors in the platform.

[1] Setting S to an integral fraction of DT_{min} was handy for proving a utilisation bound for NPS-F in [3], but in fact the DBF-based server-sizing by Sousa et al. [13] allows for dropping this constraint. In this paper, we just stick to tradition.

(iii) Server-to-processor mapping: We employ the so-called "semi-partitioned" mapping from the original NPS-F. This ensures that at least m servers never migrate; in our case, it is the H-servers that do not migrate and there can be at most m of those. Figure 1 is an example of this mapping arrangement.

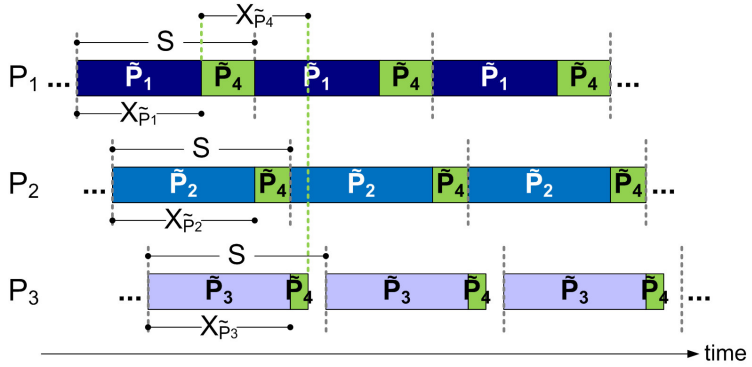

Fig. 1: Mapping of m''=4 servers to m=3 processors. Three servers ($\widetilde{P}_1$ to $\widetilde{P}_3$) never migrate and the remaining timeslot portions on each processor are re-used for mapping $\widetilde{P}_4$. The timeslot boundaries on different processors are shifted accordingly, such that the reserves of the migrating server never overlap in time. NPS-F-MC assigns H-tasks to non-migrating servers.

5 Other derivative approaches

We now formulate other MC scheduling algorithm variants drawing from NPS-F and the scheduling with deadline-scaling by Ekberg and Yi: NPS-F-IMA, a strictly partitioned variant of NPS-F-MC, and SP-EKB which differs from NPS-F-MC mainly in that tasks of mixed criticalities can be scheduled together in the same server (potentially leading to migration of H-tasks). For comparison, we also formulate cNPS-F, which foregoes deadline scaling but instead sizes H-servers only considering the H-WCETs (i.e., the "conservative approach" of Section 4). Finally P-EKB is the partitioned multiprocessor version of Ekberg and Yi's algorithm. Studying these variants helps understand the performance tradeoffs of different scheduling arrangements and safety requirements.

IMA-mindful variant (NPS-F-IMA) The Integrated Modular Avionics (IMA) standard ARINC 653 enforces spatial and temporal partitioning to ensure safety aspects and enable incremental development and certification. However, NPS-F-MC allows L-servers to migrate among different processors. For a scheduling ar-

Algorithm 4 IMA-mindful (NPS-F-IMA)

Additional code to add between lines 8 and 9 in Algorithm 1

1: **for** $(q := m_H^{''} + 1; q \leq 2 \cdot m_H^{''}; q++)$ **do**
2: Create a server $\widetilde{P}_q$ ▷ Note that $S = DT_{min}/\delta$
3: $X_{\widetilde{P}_{q-m_H^{''}}}$:=Mixed_Criticality_Inflate($\widetilde{P}_{q-m_H^{''}}$)$\cdot S$
4: Create $\tau_{\text{fake}} = \langle T_{\text{fake}} := S, D_{\text{fake}} := C_{\text{fake}} := X_{\widetilde{P}_{q-m_H^{''}}}, L\rangle$
5: $\tau(\widetilde{P}_q) := \tau(\widetilde{P}_q) \cup \{\tau_{\text{fake}}\}$
6: **end for**

Additional code to add between lines 14 to 15 in Algorithm 1

7: **if** $(m^{''} - m_H^{''} > m)$ **then**
return FAILURE
8: **end if**
9: **for** $(q := m_H^{''} + 1; q \leq 2 \cdot m_H^{''}; q++)$ **do**
10: $\tau(\widetilde{P}_q) := \tau(\widetilde{P}_q) \setminus \{\tau_{\text{fake}} \in \tau(\widetilde{P}_q)\}$
11: **end for**

rangement more inline with IMA standards, we propose a variant (NPS-F-IMA) that disallows the migration of L-servers and sizes their budgets accordingly.

The pseudocode for NPS-F-IMA is derived by adding a few lines of pseudocode to Algorithm 1, as described in Algorithm 3. As we know, each H-server is mapped to one processor. The leftover portion of the timeslot on such a processor can be turned into an L-server. Assume that $m_H^{''}$ is the number of H-servers. Then up to $m_H^{''}$ L-servers each share a processor with an H-server. Assume that H-server $\widetilde{P}_q$ and L-server $\widetilde{P}_r$ share a processor P_m. Let $X_{\widetilde{P}_q}$ be the size of periodic reserve allocated to $\widetilde{P}_q$. Then $\widetilde{P}_r$ should be filled with L-tasks such that its periodic reserve size never exceeds $S - X_{\widetilde{P}_r}$. In order to ensure this requirement, we add to $\widetilde{P}_r$ a fake task $\tau_{\text{fake}} = \langle T_{\text{fake}} = S, D_{\text{fake}} = X_{\widetilde{P}_q}, C_{\text{fake}} = X_{\widetilde{P}_q}, L\rangle$, that corresponds to the workload of $\widetilde{P}_q$, before adding any real task into it. The method ensures that, after the task-to-server assignment completes, the size of the periodic reserve for $\widetilde{P}_r$ (which is computed based on the computational requirements of the *real* tasks assigned to it), never exceeds $S - X_{\widetilde{P}_q}$.

This procedure is repeated for all servers indexed $m_H^{''} + 1$ to $2 \cdot m_H^{''}$. The pseudocode of this additional code that adds fake tasks to L-servers is presented in Algorithm 4 (lines 1 to 6). An L-server that does not share the processor with an H-server is not subject to this and can therefore use the full timeslot if needed. The NPS-F-IMA algorithm declares failure, if the number of L-servers exceeds the number of processors (see lines 7 to 8 in Algorithm 4). Once, L-servers are instantiated, these "placeholder" fake tasks are removed (see line 9 to 11 in Algorithm 4). The inflated utilisation of all the servers is then computed. In the server-to-processor mapping phase, each H-server $\widetilde{P}_k$ is mapped to processor P_k with the same index. An L-server $\widetilde{P}_k$, whose index lies in the range $m_H^{''} + 1$ to $m^{''}$ is mapped to processor $P_{k-m_H^{''}}$.

Non-deadline-scaled cNPS-F To assess the benefits from deadline-scaling in semi-partitioned scheduling, we define cNPS-F, a variant not using deadline-scaling. It uses the same bin-packing but (i) bases scheduling decisions on the real deadlines also in L-mode and (ii) uses only the H-WCETs for H-server sizing.

P-EKB and SP-EKB Ekberg and Yi's approach, formulated for uniprocessors, can be used for multiprocessor scheduling with processor partitioning.

We call this approach P-EKB. Tasks are assigned to the m processors via bin-packing (we assume First-Fit). On each processor, to test the feasibility of each assignment, the deadline scaling algorithm is used, as a schedulability test. Each time that a new task is assigned, the deadline scale factors of already assigned tasks are computed anew. This arrangement is migration-free but L-tasks and H-tasks are scheduled together on each processor, without strict isolation.

Similarly, for a semi-partitioned approach that borrows from NPS-F but without the server-level isolation of H-tasks and L-tasks, one could perform this bin-packing over m'' bins (as many as needed; not necessarily bound to m) and then create mixed-criticality servers out of those, which are mapped to the m processors as in NPS-F. We call this arrangement SP-EKB. One thing to note is that, for the purpose of sizing servers under SP-EKB, the "fake task" modelling the periodic unavaibility of the processor to the server has to be modelled as an H-task – unlike what was the case for MC-NPS-F. The reason for this is that, in the general case, neighboring servers may have both H-tasks and L-tasks meaning that it would not be possible in the H-mode to drop the server arrangement and collapse to pure partitioning/use of an entire processor's full capacity for a server's H-tasks. This means that, all other things being equal, a server would have greater inflated utilisation under SP-EKB than under MC-NPS-F.

Although SP-EKB dominates P-EKB (*if* the same task ordering is used for both algorithms), it allows the migration of high-criticality tasks, which may be undesirable for in practice. Table 1 summarises the different design aspects.

algorithm	scheduling class	deadline scaling	server-based	H-task/L-task isol.	H-task migration
NPS-F-MC	semi-part.	YES	YES	YES	NO
NPS-F-IMA	part.	YES	YES	YES	NO
cNPS-F	semi-part.	NO	YES	YES	NO
P-EKB	part.	YES	NO	NO	NO
SP-EKB	semi-part.	YES	NO	NO	YES

Table 1: Comparison of scheduling approaches

6 Evaluation

Experimental setup To evaluate the theoretical scheduling effectiveness of the approaches presented, we apply the respective offline schedulability tests to synthetic task sets, whose generation is controlled with the following parameters:

– L-mode utilisations (U_i^L): Generated using the UUnifast-discard algorithm [14] for unbiased distribution. C_i^L is derived as $U_i^L \cdot T_i$.

– Task period (T_i): Generated with a log-uniform distribution, in the range of 10ms to 100ms, i.e., $T_i = 10^x : x \in [log_{10}10, log_{10}100]$.

– Task deadline (D_i): The scheduling approaches discussed work for constrained deadlines ($D_i \leq T_i$) but this evaluation assumes implicit deadlines ($D_i = T_i$).

– Distribution of high and low criticality tasks: The fraction of H-tasks in the task set is configurable. (For an integer number of H-tasks, we round up.)

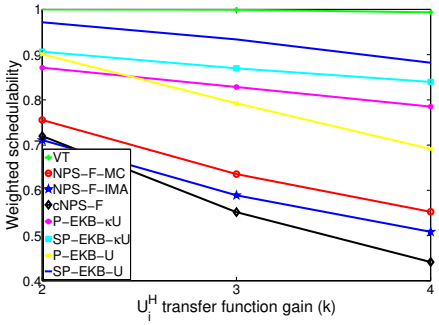

Fig. 2: n=12, m=4, 40% H-tasks

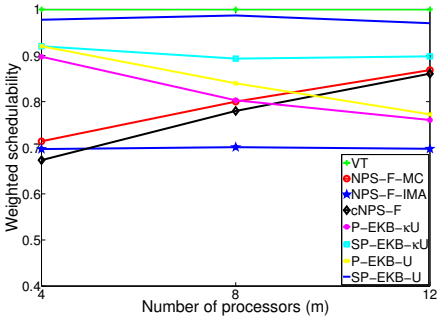

Fig. 3: n=16, 40% H-tasks, k=2

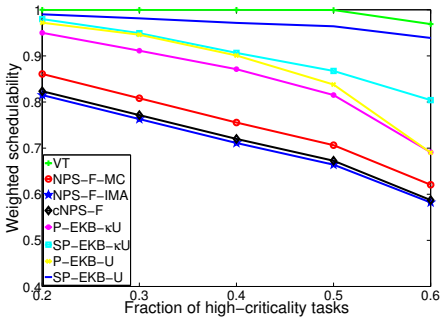

Fig. 4: n=12, m=4, k=2

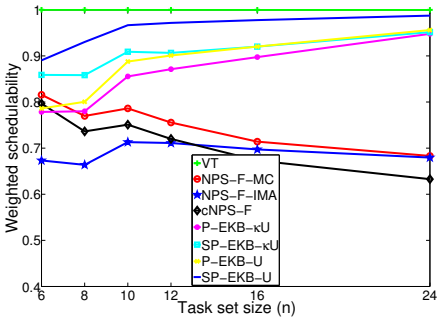

Fig. 5: m=4, 40% H-tasks, k=2

– H-mode utilisation (U_i^H): Derived from U_i^L via a transfer function $f(U_i^L, k)$ with a parameter k. For small values of U_i^L, $f(U_i^L, k) \approx k \cdot U_i^L$ but for greater values the gain is progressively smaller, so that $U_i^L \leq f(U_i^L, k) \leq 1, \ \forall \ U_i^L \in [0,\ 1]$. For details, see the Appendix of our TR [15]. C_i^H is computed as $T_i \cdot U_i^H$.

The resolution is microsecond. Each task set is generated for a given target utilisation $U^* = x * m : x \in (0, 1]$, where m is the number of processors. For each combination of input parameters explored we generate 1000 task sets.

We compare the scheduling approaches listed in Table 1. To keep the number of plots in check, in each experiment we vary one parameter with the others fixed. We also plot a "validity test" (VT), namely: $(U^L \leq m) \wedge (U^H \leq m)$. This test (a necessary but not sufficient condition for schedulability) rejects trivially infeasible tasks sets. Its curve over-approximates the feasible task sets.

Due to lack of space instead of providing plots comparing the algorithms 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*.[2] This performance metric is adopted

[2] The plots of (non-weighted) schedulability can still be found in the Appendix of our TR [15].

from [16] and allows condensing what would have been three-dimensional plots into two dimensions. It is a weighted average, in which more weight is given to task-sets with higher utilisation, i.e., task-sets that are supposedly harder to schedule. Specifically, using the notation from [17]:

Let $S_y(\tau, p)$ represent the binary 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 some schedulability test y as a function of parameter p, is:

$$W_y(p) = \frac{\sum_{\forall \tau} \left(\bar{U}^L(\tau) \cdot S_y(\tau, p) \right)}{\sum_{\forall \tau} \bar{U}^L(\tau)} \tag{5}$$

In the above equation (adapted from [17]), $\bar{U}^L(\tau) \stackrel{\text{def}}{=} \frac{U^L(\tau)}{m}$ is the system utilisation in L-mode, normalised by the number of processors m.

Results For P-EKB and SP-EKB, we used two different configurations: "-κU" means that tasks are indexed with H-tasks preceding L-tasks and in order of non-increasing U_i^L, for same-criticality tasks. "-U" means that tasks are simply indexed by non-increasing U_i^L. The corresponding variants with ordering by D_i, instead of U_i^L, almost always performed worse, so we don't include them.

For all four parameters varied (transfer function gain k, number of processors m, fraction of H-tasks, number of tasks n), most of the time[3] SP-EKB outperforms P-EKB. In turn, P-EKB usually outperforms NPS-F-MC and, by a larger margin, cNPS-F. Figure 3 is an exception, with P-EKB dropping in performance, as m rises, contrary to the other algorithms, and being overtaken by NPS-F-MC and NPS-F-IMA. This is because, when both the system utilisation (normalised by m) and the number of processors are kept the same during task generation but m increases, the average U_i^L also increases. This implies increased bin-packing fragmentation for non-server-based partitioned approaches.

Some conclusions drawn from these experiments:

–Semi-partitioning helps moderately but noticeably with performance. (Compare NPS-F-MC to NPS-F-IMA and SP-EKB to P-EKB).

–For SP-EKB and P-EKB, the choice of task ordering for the bin-packing matters a lot.

– The isolation of H-tasks from L-tasks, through separate servers for the two task categories, sharply penalises performance. (Compare NPS-F-MC to SP-EKB.) By comparison, the performance hit from disallowing L-server migration is smaller. (Compare NPS-F-MC with NPS-F-IMA.)

[3] Recall that, for the same configuration, SP-EKB strictly dominates P-EKB. However, some task sets schedulable by SP-EKB-κU are unschedulable by P-EKB-U (and vice versa) and some tasks schedulable by SP-EKB-U are unschedulable by P-EKB-κU (and vice versa).

Ultimately, the choice of scheme will depend on the kind of scheduling guarantees and isolation the particular application scenario requires, but our experiments explore the performance ceilings associated with each arrangement.

7 Conclusions and future work

This work brought together server-based semi-partitioning and deadline-scaling techniques for mixed-criticality scheduling. Our main contribution, the scheduling algorithm NPS-F-MC, offers isolation between tasks of different criticalities but allows low-criticality tasks to migration, for better system utilisation. Our experiments show that deadline scaling also works well in a semi-partitioned context. However, enforcing complete scheduling isolation, can be expensive. In practice, different application requirements might mean that any task migration is to be avoided or, conversely, that complete scheduling isolation between task of different criticalities is not a requirement, as long as schedulability is ensured even in the case of mode change. For these cases, we therefore formulate the related scheduling algorithms NPS-F-IMA and SP-EKB, respectively. Our experimental results of theoretical schedulability offer some preliminary exploration of the performance tradeoffs when considering different scheduling arrangements: partitioning vs semi-partitioning, scheduling isolation for tasks of different criticalities by use of separate servers vs mixed-criticality scheduling within the same server, use of deadline scaling in the context of a semi-partitioned approach.

As future work, we intend to also incorporate the effects of task contention over cache and memory into the schedulability tests.

Acknowledgments

We would like to thank Pontus Ekberg for clarifying to us some aspects of his algorithm.

This work was partially supported by National Funds through FCT/MEC (Portuguese Foundation for Science and Technology) and co-financed by ERDF (European Regional Development Fund) under the PT2020 Partnership, within the CISTER Research Unit (CEC/04234); also by by FCT/MEC and the EU ARTEMIS JU within project ARTEMIS/0001/2013- JU grant nr. 621429 (EMC2).

References

1. A. Burns and R. Davis, "Mixed criticality systems: A review," TR. Computer Science, U. of York, UK, 2013.
2. S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proc. RTSS*, 2007, pp. 239–243.
3. K. Bletsas and B. Andersson, "Preemption-light multiprocessor scheduling of sporadic tasks with high utilisation bound," in *Proc. RTSS*, 2009.
4. P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," in *Proc. ECRTS*, 2012, pp. 135–144.

5. S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in *Proc. ECRTS*, 2012, pp. 145–154.
6. Federal Aviation Authority, "CAST-32: Multi-core Processors," https://www.faa.gov/, 2014.
7. S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. RTSS*, 1990, pp. 182–190.
8. P. Ekberg and W. Yi, "Bounding and shaping the demand of generalized mixed-criticality sporadic task systems," *Real-Time Systems*, vol. 50, no. 1, pp. 48–86, 2014.
9. A. Masrur, D. Müller, and M. Werner, "Bi-level deadline scaling for admission control in mixed- criticality systems," in *RTCSA*, 2015, pp. 100–109.
10. A. Easwaran, "Demand-based scheduling of mixed-criticality sporadic tasks on one processor," in *Proc. RTSS*, 2013.
11. K. Bletsas and S. M. Petters, "Using NPS-F for Mixed-Criticality Multicore Systems," in *Proc. RTSS WiP*, 2012.
12. F. Zhang and A. Burns, "Improvement to Quick Processor-Demand Analysis for EDF-Scheduled Real-Time Systems," in *Proc. ECRTS*, 2009, pp. 76–86.
13. P. B. Sousa, K. Bletsas, E. Tovar, P. F. Souto, and B. Åkesson, "Unified overhead-aware schedulability analysis for slot-based task-splitting," *Real-Time Systems*, vol. 50, no. 5-6, pp. 680–735, 2014.
14. E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2009.
15. M. A. Awan, K. Bletsas, P. F. Souto, and E. Tovar, "Semi-partitioned mixed-criticality scheduling," TR. CISTER/ISEP. http://www.cister.isep.ipp.pt/docs/CISTER-TR-161102, 2016.
16. A. Bastoni, B. Brandenburg, and J. Anderson, "Cache-related preemption and migration delays: Empirical approximation and impact on schedulability," *Proc. OSPERT*, pp. 33–44, 2010.
17. A. Burns and R. Davis, "Adaptive mixed criticality scheduling with deferred preemption," in *Proc. RTSS*, 2014, pp. 21–30.