Compensating Adaptive Mixed Criticality Scheduling

Robert I. Davis
rob.davis@york.ac.uk
Department of Computer Science, University of York
York, UK

Alan Burns
alan.burns@york.ac.uk
Department of Computer Science, University of York
York, UK

Iain Bate
iain.bate@york.ac.uk
Department of Computer Science, University of York
York, UK

ABSTRACT

The majority of prior academic research into mixed criticality systems assumes that if high-criticality tasks continue to execute beyond the execution time limits at which they would normally finish, then further workload due to low-criticality tasks may be dropped in order to ensure that the high-criticality tasks can still meet their deadlines. Industry, however, takes a different view of the importance of low-criticality tasks, with many practical systems unable to tolerate the abandonment of such tasks.

In this paper, we address the challenge of supporting genuinely graceful degradation in mixed criticality systems, thus avoiding the abandonment problem. We explore the Compensating Adaptive Mixed Criticality (C-AMC) scheduling scheme. C-AMC ensures that both high- and low-criticality tasks meet their deadlines in both normal and degraded modes. Under C-AMC, jobs of low-criticality tasks, released in degraded mode, execute imprecise versions that provide essential functionality and outputs of sufficient quality, while also reducing the overall workload. This compensates, at least in part, for the overload due to the abnormal behavior of high-criticality tasks. C-AMC is based on fixed-priority preemptive scheduling and hence provides a viable migration path along which industry can make an evolutionary transition from current practice.

CCS CONCEPTS

• **Computer systems organization** → **Real-time systems**; **Real-time systems**; • **Software and its engineering** → **Real-time schedulability**; **Real-time schedulability**.

KEYWORDS

Real-Time, Mixed Criticality, Fixed Priority, Schedulability Analysis

ACM Reference Format:

Robert I. Davis, Alan Burns, and Iain Bate. 2022. Compensating Adaptive Mixed Criticality Scheduling. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems (RTNS '22), June 7–8, 2022, Paris, France.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3534879.3534895

RTNS '22, June 7–8, 2022, Paris, France

ACM ISBN 978-1-4503-9650-9/22/06…$15.00
https://doi.org/10.1145/3534879.3534895

1 INTRODUCTION

There is a considerable body of research into mixed criticality systems stemming from the model presented by Vestal in 2007[57], see [21, 22] for a comprehensive survey and review. The majority of prior academic research in this area assumes that if high-criticality tasks continue to execute beyond the execution time limits at which they would normally finish, then further workload due to low-criticality tasks may be dropped in order to ensure that the high-criticality tasks can still meet their deadlines. Industry, however, takes a different view of the importance of low-criticality tasks, with practical systems unable to tolerate their abandonment. This disconnect has been discussed in a number of previous papers [34], [30], [49] and [29].

From an industry perspective, criticality relates to the functional safety of an application, see the IEC 61508, DO-178C, DO-254 and ISO 26262 standards. Typical names for criticality levels are ASILs (Automotive Safety and Integrity Levels), DALs (Design Assurance Levels or Development Assurance Levels) and SILs (Safety Integrity Levels). The criticality level of an application, or system function implemented via both hardware and software, is determined by a system safety assessment that involves Failure Modes and Effects Analysis. The criticality level typically depends on: (i) an evaluation of the consequences of a failure, (ii) the probability that the failure occurs, and (iii) the provision of means to mitigate or cope with the failure. Hence the criticality level of an application may not necessary reflect the severity or consequences of the failure. An example given by Esper et al. [30] and Ernst and Natale [29] comes from ISO 26262. If the probability of failure occurrence is very low, then the ASIL level assigned may be low, despite severe consequences if a failure actually happens. A different application with a high probability of failure may be assigned a higher ASIL despite having lower severity consequences in the event of failure. With this interpretation, the idea of dropping low-criticality functionality in favour of completing that of high-criticality does not hold; the consequences would be more severe. ISO 26262 also permits high-criticality applications to be composed from low-criticality components with diverse implementations; dropping one of these low-criticality components would remove the necessary diversity and undermine the safety argument for the high-criticality function. The message is that the criticality level is not the same as importance, and hence functionality that has low criticality cannot simply be dropped.

The notion of importance is explored further by Bletsas et al. [16], who draw a distinction between *criticality* as used for verification and *importance* as used to control graceful degradation. A task may have low criticality but high importance, or vice versa, though there is often a correlation between the two. Sundar and Easwaran [56] take this idea further, with a context aware approach to choosing which tasks to degrade based on the task or tasks that overrun.

In this paper, we take the view, prevalent in industry, that completely abandoning new releases of low-criticality tasks is *not* acceptable when system behavior diverges from what is normally expected. Rather, we consider a mixed-criticality system model where tasks of both high- and low-criticality must *always* meet their deadlines. Specifically, we propose the Compensating Adaptive Mixed Criticality (C-AMC) scheduling scheme that meets these stricter requirements. C-AMC ensures that both high- and low-criticality tasks meet their deadlines in both normal and degraded modes. However, once degraded mode is entered, *new* releases (jobs) of low-criticality tasks execute imprecise versions that provide essential functionality and outputs of sufficient quality, while reducing overall workload via their smaller execution time budgets. This adaptive behavior compensates, at least in part, for the longer execution times that may be exhibited by jobs of high-criticality tasks, for example executing error handling code that is not expected to execute during normal operation [42]. A similar model was previously suggested in a preliminary workshop paper at WMC 2013 [18]; however, the model and analysis provided there did not ensure that every job of a low-criticality task would meet its deadline, rather jobs that were active when degraded mode was entered were immediately only permitted a smaller execution time budget. Thus if such a job was part way through executing its primary version, then that job could end up being aborted due to an execution time overrun of the reduced budget, or alternatively miss its deadline. The imprecise mixed criticality model is also supported by the dynamic-priority EDF-VD [46, 47] scheme and by a scheme based on MC-fluid scheduling [10]. The research presented in this paper differs from those prior works by focussing on fixed-priority preemptive scheduling schemes, that can be adopted by industry via an evolutionary transition from current practice [43, 44].

The main contribution of the research reported in this paper is the Compensating Adaptive Mixed Criticality (C-AMC) scheme and its associated schedulability analysis. The C-AMC scheme:

- Ensures that both high- and low-criticality tasks meet their deadlines in both normal and degraded modes.
- Supports a form of degradation that is genuinely graceful, while reducing low-criticality workload to compensate for unexpected increases in high-criticality workload.
- Substantially improves schedulability compared to the single criticality approach that is common practice in industry.
- Provides a viable migration path for industry to make an evolutionary transition from current practice, based on fixed-priority preemptive scheduling [2, 3].
- Addresses one of the key open issues identified in the survey of research into mixed criticality systems [21]: Adding *"support for limited low-criticality functionality in higher criticality modes, avoiding the abandonment problem."*

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 introduces the system model, terminology, and notation used. Section 4 presents schedulability analysis for the C-AMC scheme, the performance of which is evaluated in Section 5. Section 6 concludes with a summary and directions for future research. Finally, the appendix considers task allocation on a multi-core processor under the C-AMC scheme.

2 RELATED WORK

In this section, we outline prior work on mixed criticality fixed-priority scheduling schemes for single-core processors.

Since Vestal's seminal work [57] in 2007, mixed criticality systems have become a hot topic of real-time systems research. Many of these papers focus on scheduling schemes that are based on fixed priorities, most notably Static Mixed Criticality (SMC) [8] and Adaptive Mixed Criticality (AMC) [9]. AMC is considered the most effective fixed-priority scheme [38], and has been extended to account for many additional aspects including: preemption thresholds [59, 60], multiple criticality levels [31], criticality-specific periods [11], changes in priority [7], communications [19], deferred preemption [20], weakly-hard timing constraints [33], probabilistic task models [48], design optimization [62], context switch costs [25], robustness and resilience [24], implementation overheads [44], and semi-clairvoyant timing behavior [23, 61]. An exact analysis has also been developed for periodic task sets [4, 50].

Various forms of degraded service have been proposed for low-criticality tasks when system behavior departs from what is normally expected. These include: abandoning all jobs; letting jobs that have already started complete execution, but abandoning newly released jobs [8]; extending periods and/or deadlines [54, 55]; dropping jobs from specific tasks [1, 32, 39]; and applying weakly-hard constraints, allowing some jobs to be skipped [33]. Alternative approaches seek to delay the time at which the system starts dropping new releases of low-criticality tasks, and also to reduce the time that the system spends doing so. Delaying the onset of degraded behavior can be achieved by using off-line sensitivity analysis [51] to increase all low-criticality execution time budgets while still retaining a schedulable system [18, 52, 53, 58]. Online accounting for budget under and overruns can also be used to delay switching to degraded mode [37]. Further, the time spent in degraded mode can be reduced via online budget accounting resulting in a faster bailout [13, 14] and recovery. Alternatively, the amount of time spent in degraded mode can be substantially reduced by triggering mode change transitions based on response times rather than execution times [15]. Also, by using a separate background priority queue, low-criticality jobs that would have been dropped in degraded mode can be run in what would otherwise have been idle time, providing a last chance to meet their deadlines [40]. Finally, we note that research into mixed criticality scheduling, including that described in this paper, differs from research into operational mode changes, due to the specific trigger conditions for the mode change and, as a consequence, the form of analysis required [17].

3 SYSTEM MODEL

In this paper, we assume a mixed criticality system executing on a single-core processor under fixed-priority preemptive scheduling. The model and subsequent analysis are also applicable to multi-core processors employing partitioned fixed-priority preemptive scheduling with full isolation between cores.

A mixed criticality system is assumed to have two criticality levels: *HI* and *LO*. Each task τ_i is characterised by its criticality level L_i, which is either *HI* or *LO*. Each task τ_i has two execution time budgets $C_i(LO)$ and $C_i(HI)$ that bound its execution time in normal and degraded mode respectively. For a *HI*-criticality task τ_k, $C_k(LO)$ and $C_k(HI)$ (with $C_k(LO) \le C_k(HI)$) are, respectively, the low assurance and the high assurance estimates of the WCET of its primary version, which is the only version that it executes. By contrast, for a *LO*-criticality task τ_j, $C_j(LO)$ and $C_j(HI)$ (with $C_k(LO) \ge C_k(HI)$) are low assurance estimates of the WCET of, respectively, its primary version and its imprecise version.

Each task τ_i has a minimum inter-arrival time or period T_i between releases of its jobs, and a constrained relative deadline D_i, where $D_i \le T_i$. Each task τ_i is assumed to have a unique priority, with *hp*(i) (resp. *hep*(i)) used to denote the set of tasks with higher (resp. higher than or equal) priority to task τ_i. The priority assigned to each task is independent of its criticality level. All task parameters are assumed to take integer values, for example measured in processor clock cycles.

The Real-Time Operating System (RTOS) is required to provide execution time monitoring and budget enforcement. The RTOS is assumed to abort any job of a task that does not complete within its execution time budget. For a *LO*-criticality task τ_j, this budget is set to $C_j(LO)$ for jobs released in normal mode and to $C_j(HI)$ for jobs released in degraded mode. For a *HI*-criticality task τ_k, the budget is set to $C_k(HI)$ for jobs released in either mode.

The RTOS is also responsible for transitioning the system between normal and degraded modes. The system switches from normal mode to degraded mode when a *HI*-criticality task τ_k executes for $C_k(LO)$ without signaling completion, and returns to normal mode on an idle instant[1]. Jobs of a *LO*-criticality task τ_j that are released in normal mode execute their primary version and must complete within an execution time budget of $C_j(LO)$, whereas those jobs released in degraded mode execute their imprecise version and must complete within an execution time budget of $C_j(HI)$. By contrast, jobs of a *HI*-criticality task τ_k always execute their primary version and must complete within an execution time budget of $C_j(HI)$.

The decrease in workload due to *LO*-criticality jobs executing imprecise versions in degraded mode compensates, at least in part, for *HI*-criticality jobs that have overrun their low assurance WCET budget. We therefore refer to the mixed criticality scheduling scheme described above as Compensating Adaptive Mixed Criticality (C-AMC). Schedulability analysis for C-AMC, introduced in Section 4, provides the necessary guarantees that *all* jobs of *all* tasks will meet their deadlines under this scheme.

4 C-AMC SCHEME

In this section, we present schedulability analysis for the C-AMC scheme. This analysis builds on the existing analysis for AMC [9] and also on the analysis sketched in a preliminary workshop paper [18] for a similar model that did not provide schedulability guarantees for *all* jobs of *LO*-criticality tasks.

[1] An idle instant occurs when there are no jobs released prior to that time that have not completed.

In the original paper on AMC [9], two sufficient schedulability tests were developed. The first approach, called AMC-rtb, takes account of a response time bound on the duration over which higher priority *LO*-criticality tasks can be released. The second, more precise approach, called AMC-max, determines the worst-case response time by taking into account all possible times at which the transition from normal to degraded mode could occur. In the following subsections, we derive corresponding schedulability tests for C-AMC: (i) the C-AMC-rtb test based on a response time bound, and (ii) the C-AMC-max test based on a consideration of when the transition to degraded mode could occur.

4.1 C-AMC-rtb Schedulability Test

Considering the normal mode, where every task τ_i complies with its $C_i(LO)$ execution time budget, then schedulability can be determined using standard response time analysis for fixed-priority preemptive scheduling [6, 41]:

$$R_i(LO) \;=\; C_i(LO) \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{1}$$

Considering degraded mode, under C-AMC, all tasks are required to meet their deadlines. An upper bound on the worst-case response time $R_i(HI)$ for a task τ_i (of *HI*- or *LO*-criticality), accounting for the transition to degraded mode, can be derived as follows:

$$\begin{aligned} R_i(HI) \;=\; \max(C_i(LO), C_i(HI)) \;+\; \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \\ +\; \sum_{j \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil (C_j(LO) - C_j(HI)) \end{aligned} \tag{2}$$

where $\mathbf{hp}(i)$ is the set of tasks with priorities higher than that of task τ_i, and $\mathbf{hpL}(i)$ is the set of *LO*-criticality tasks with priorities higher than that of task τ_i.

The first term in (2) accounts for the larger of the two execution time budgets for both *HI*- and *LO*-criticality tasks. The second term assumes that jobs of each higher priority task τ_j may contribute interference equating to $C_j(HI)$ throughout the entire response time of task τ_i. Recall that for *HI*-criticality tasks this is the larger value, since $C_j(HI) \ge C_j(LO)$, while for *LO*-criticality tasks, it is the smaller value, since in that case $C_j(HI) \le C_j(LO)$. The third term adjusts for the fact that jobs of each higher priority *LO*-criticality task τ_j released in normal mode, which extends for at most $R_i(LO)$, can contribute an extra $C_j(LO) - C_j(HI)$ over and above the interference already accounted for from these jobs in the second term. In other words, jobs released in normal mode (within $R_i(LO)$) contribute at most $C_j(LO)$ since they execute primary versions, while the remaining jobs released at or after $R_i(LO)$ contribute at most $C_j(HI)$ since they execute imprecise versions.

The analysis embodied in (1) and (2) is referred to as the C-AMC-rtb test. Observe that the C-AMC-rtb test reduces to the AMC-rtb test if $C_j(HI) = 0$ for every *LO*-criticality task τ_j, i.e. jobs of *LO*-criticality tasks are not released in degraded mode, and the C-AMC-rtb test is modified to *not* check the schedulability of *LO*-criticality tasks in degraded mode, i.e. $R_i(HI)$ is not computed for *LO*-criticality tasks. Thus, the AMC-rtb test dominates the C-AMC-rtb test; however, this dominance comes at a cost of not providing any guarantees that jobs of LO-criticality tasks will meet their deadlines in degraded mode. In contrast to the AMC-rtb test, the C-AMC-rtb test guarantees the schedulability of *all* jobs of LO-criticality tasks including those that are active in degraded mode. Hence, if $C_j(HI) = 0$ for every LO-criticality task τ_j, i.e. jobs of LO-criticality tasks are not released in degraded mode, then the C-AMC-rtb test guarantees schedulability of all of the jobs of LO-criticality tasks that are released in normal mode, including those that are active across the mode change transition and hence complete in degraded mode. Ensuring schedulability of LO-criticality jobs that complete in degraded mode is a key difference with respect to the AMC-rtb test. For task sets schedulable according to the C-AMC-rtb test, no job of any task misses its deadline.

4.2 C-AMC-max Schedulability Test

Considering the normal mode, where every task τ_i complies with its $C_i(LO)$ execution time budget, then schedulability is again determined using the standard approach given by (1).

Considering degraded mode, under C-AMC, all tasks are required to meet their deadlines. An upper bound on the worst-case response time $R_i(HI)$ for a task τ_i (of HI- or LO-criticality), accounting for the transition to degraded mode, can be derived by computing the worst-case response time $R_i^s(HI)$ of task τ_i, assuming a transition to degraded mode at time s, and then taking the maximum of these values over all possible values of s. The formula for $R_i^s(HI)$ is constructed from the different forms of interference that task τ_i can experience:

$$R_i^s(HI) = \max(C_i(HI), C_i(LO)) + I_L(i, s, R_i^s(HI)) + I_H(i, s, R_i^s(HI)) \tag{3}$$

where $I_L(i, s, t)$ and $I_H(i, s, t)$ represent an upper bound on the interference from higher priority LO-criticality and higher priority HI-criticality tasks respectively, over a priority level-i busy period of length t, with a transition to degraded mode at a time s, as measured from the start of the busy period.

$I_L(i, s, t)$ is defined by considering the number of jobs of each higher priority LO-criticality task τ_j that can execute in a priority level-i busy period of length t, with the mode change taking place at time s, with $s < t$. Jobs of a LO-criticality task τ_j that are released before the mode change at time s execute their primary versions and so contribute $C_j(LO)$, while those jobs released at or after the mode change execute their imprecise versions and so contribute $C_j(HI)$. The total worst-case interference from higher priority LO-criticality tasks is therefore upper bounded by:

$$I_L(i, s, t) = \sum_{j \in \mathbf{hpL}(i)} \left(\left\lceil \frac{t}{T_j} \right\rceil C_j(HI) + \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) (C_j(LO) - C_j(HI)) \right) \tag{4}$$

where $\mathbf{hpL(i)}$ is the set of LO-criticality tasks with higher priority than τ_i.

The first term in (4) accounts for the fact that every job of task τ_j released in the busy period contributes at least $C_j(HI)$, while the second term corrects for the fact that those jobs released by time s contribute a larger amount $C_j(LO)$.

Following the analysis derived for the AMC-max test [9], a $\left\lfloor \frac{s}{T_j} \right\rfloor +$ 1 formulation is used for the second term in (4). This ensures that $I_L(i, s, t)$ increases with increasing values of s with steps at values of s corresponding to multiples of the periods of the higher priority LO-criticality tasks. This property is used later to limit the number of values of s that need to be checked. The use of $\left\lfloor \frac{s}{T_j} \right\rfloor + 1$ is preferred to $\left\lceil \frac{s}{T_j} \right\rceil$, since the former provides a valid upper bound for $I_L(i, s, t)$, while also retaining compatibility with, and reduction to, the original AMC-max test [9].

$I_H(i, s, t)$ is defined in the same way as in the analysis of AMC-max [9], by considering the number of jobs of each higher priority HI-criticality task τ_k that can execute in a priority level-i busy period of length t, with the mode change taking place at time s, with $s < t$. Those jobs of a HI-criticality task τ_k that have some part of their execution after time s can contribute interference of $C_k(HI)$, with the remainder contributing the smaller value $C_k(LO)$.

The maximum number of jobs of τ_k, with $D_k \le T_k$, that can be released in a busy period of length t and have some part of their execution in an interval of length $t - s$ is upper bounded by:

$$\min \left\{ \left\lceil \frac{t - s + D_k}{T_k} \right\rceil, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{5}$$

The first term in (5) follows from the fact that the latest a job of task τ_k can execute is at its deadline, while the earliest that subsequent jobs can execute is at their release times. For small values of s, the first term can be pessimistic; including more jobs than can actually be released in an interval of length t. This is taken into account by the second term, which limits the total number of jobs to the maximum that could be released in an interval of length t. The total worst-case interference from higher priority HI-criticality tasks is therefore upper bounded by:

$$I_H(i, s, t) = \sum_{k \in \mathbf{hpH}(i)} \left\lceil \frac{t}{T_k} \right\rceil C_k(LO) + \sum_{k \in \mathbf{hpH}(i)} \min \left\{ \left\lceil \frac{t - s + D_k}{T_k} \right\rceil, \left\lceil \frac{t}{T_k} \right\rceil \right\} (C_k(HI) - C_k(LO)) \tag{6}$$

where $\mathbf{hpH(i)}$ is the set of HI-criticality tasks with higher priority than τ_i.

Hence the worst-case response time of task τ_i, occurring in degraded mode, with a mode change at time s is upper bounded by:

$$R_i^s(HI) = \max(C_i(HI), C_i(LO)) + \sum_{j \in \mathbf{hpL}(i)} \left(\left\lceil \frac{R_i^s(HI)}{T_j} \right\rceil C_j(HI) + \left(\left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) (C_j(LO) - C_j(HI)) \right) + \sum_{k \in \mathbf{hpH}(i)} \left\lceil \frac{R_i^s(HI)}{T_k} \right\rceil C_k(LO) + \sum_{k \in \mathbf{hpH}(i)} \min \left\{ \left\lceil \frac{R_i^s(HI) - s + D_k}{T_k} \right\rceil, \left\lceil \frac{R_i^s(HI)}{T_k} \right\rceil \right\} (C_k(HI) - C_k(LO)) \tag{7}$$

An upper bound on the worst-case response time of task τ_i is then given by the maximum over all possible values of s:

$$R_i(HI) = \max_{\forall s, s < R_i(LO)} \left\{ R_i^s(HI) \right\} \tag{8}$$

Note that the terms in (5), (6) and (7) have been simplified or re-arranged with respect to how they appear in the corresponding analysis for AMC-max [9].

Finally, it is necessary to limit the number of values of s that are considered from the range of all possible values. In (7), the first summation term, i.e. $I_L(i, s, t)$, increases as a step function with increasing values of s, while the final summation term, from $I_H(i, s, t)$, decreases with increasing values of s. The other terms do not vary with s. It follows that $R_i^s(HI)$ can only increase at values of s corresponding to multiples of the periods of higher priority LO-criticality tasks, hence these are the only values of s that need to be considered. Further, the mode change must occur by $R_i(LO)$, otherwise either task τ_i completes or, in the case that τ_i is a HI-criticality task, τ_i may itself be responsible for causing the mode change at that time. Hence s is restricted in (8) to the interval $[0, R_i(LO))$[2]. Finally, in the degenerate case where there are no LO-criticality tasks, $s = 0$ is checked, which reduces (7) to the standard analysis for fixed priority preemptive scheduling.

Note that the C-AMC-max analysis derived above does not assume a synchronous arrival sequence for all tasks, as that would not necessarily result in the worst-case response time. Rather, the analysis accounts independently for the maximum interference that can occur in two time windows, the first of length s representing normal mode, and the second of length $t - s$ representing degraded mode.

Observe that the C-AMC-max test reduces to the AMC-max test if $C_j(HI) = 0$ for every LO-criticality task τ_j, i.e. jobs of LO-criticality tasks are not released in degraded mode, and the C-AMC-max test is modified to *not* check the schedulability of LO-criticality tasks in degraded mode, i.e. $R_i(HI)$ is not computed for LO-criticality tasks. Thus, the AMC-max test dominates the C-AMC-max test; however, this dominance comes at a cost of not providing any guarantees that jobs of LO-criticality tasks will meet their deadlines in degraded mode. In contrast to the AMC-max test, the C-AMC-max test guarantees the schedulability of *all* jobs of LO-criticality tasks including those that are active in degraded mode. Hence, if $C_j(HI) = 0$ for every LO-criticality task τ_j, i.e. jobs of LO-criticality tasks are not released in degraded mode, then the C-AMC-max test guarantees schedulability of all jobs of LO-criticality tasks that are released in normal mode, including those that are active across the mode change transition and hence complete in degraded mode. Ensuring schedulability of LO-criticality jobs that complete in degraded mode is a key difference with respect to the AMC-max test. For task sets schedulable according to the C-AMC-max test, no job of any task misses its deadline.

Comparing the analysis for C-AMC-max and C-AMC-rtb, the following hold. First, with the C-AMC-max analysis (7), the largest possible contribution from higher priority LO-criticality tasks occurs when s takes its largest value, in which case the overall contribution from those tasks equates to that assumed by the C-AMC-rtb analysis in (2). This can be seen by considering that for each higher priority LO-criticality task τ_j, the largest value of s considered must be in the range $\left[\left\lfloor \frac{R_i(LO)-1}{T_j} \right\rfloor T_j, R_i(LO) - 1\right]$, since the largest value of s corresponds to a multiple of the period of some higher priority LO-criticality task such as τ_j and $s < R_i(LO)$. Further for any value of s in that range $\left\lfloor \frac{s}{T_j} \right\rfloor + 1 = \left\lceil \frac{R_i(LO)}{T_j} \right\rceil$. Substituting $\left\lceil \frac{R_i(LO)}{T_j} \right\rceil$ for $\left\lfloor \frac{s}{T_j} \right\rfloor + 1$ in (7) results in the same contribution from higher priority LO-criticality tasks as in (2). Second, with the C-AMC-max analysis (7), the largest possible contribution from higher priority HI-criticality tasks occurs when s takes its smallest value $s = 0$, in which case the overall contribution from those tasks equates to that assumed by the C-AMC-rtb analysis in (2). Since s cannot take both its smallest and largest possible values simultaneously, it follows that the C-AMC-max analysis dominates the C-AMC-rtb analysis.

[2]It is not required to check $s = R_i(LO)$, since any release of a LO-criticality task at that time would have a budget of $C_j(LO)$.

4.3 Priority Assignment

To maximize schedulability it is necessary to assign task priorities in an optimal way [27]. For constrained-deadline mixed-criticality task sets scheduled under AMC and analysed using AMC-max or AMC-rtb, it is known [9] that Deadline Monotonic priority ordering [45] is not optimal, but that an optimal priority ordering can be obtained via Audsey's Optimal Priority Assignment (OPA) algorithm [5].

It was proved in [26] that it is both sufficient and necessary to show that a schedulability test meets three simple conditions in order for Audlsey's OPA algorithm to be applicable. These three conditions require that schedulability of a task according to the test is: (i) independent of the relative priority order of higher priority tasks, (ii) independent of the relative priority order of lower priority tasks, and (iii) cannot get worse if the task is moved up one place in the priority order (i.e. its priority is swapped with that of the task immediately above it in the priority order).

We observe that these three conditions hold for the C-AMC-rtb and C-AMC-max analyses derived in Section 4, and thus Audsley's OPA algorithm is applicable and optimal with respect to these schedulability tests.

5 EVALUATION

In this section, we present an evaluation of the C-AMC schedulability tests introduced in Section 4.

5.1 Task Set Parameter Generation

The task set parameters used in the experiments were generated using a similar approach to that previously taken for mixed criticality systems, with the Dirichlet-Rescale (DRS) algorithm [36] (open source Python software [35]) used to provide an unbiased distribution of utilization values that sum to the target utilization required subject to a set of individual constraints.

The number of tasks per task set was fixed, default $N = 20$. The number of HI-criticality tasks N_{HI} was set to $N \cdot CP$ where CP is the Criticality Proportion (default $CP = 0.5$), with the remaining $N_{LO} = N - N_{HI}$ tasks designated LO-criticality.

Task utilizations were generated using the DRS algorithm. First, LO-criticality utilization values, $U_i(LO)$, were generated for the N_{HI} HI-criticality tasks, such that the total LO-criticality utilization, U_{HI}^{LO}, of those tasks summed to $CP \cdot U$, where U is the overall target utilization required. Similarly, LO-criticality utilization values, $U_i(LO)$, were generated for the N_{LO} LO-criticality tasks, such that the total LO-criticality utilization, U_{LO}^{LO}, of those tasks summed to $(1 - CP) \cdot U$. In both cases, the task utilization values were constrained to be in the range $[0, 1.0]$. Second, HI-criticality utilization values, $U_i(HI)$, were generated for the N_{HI} *HI*-criticality tasks, such that the total *HI*-criticality utilization, U_{HI}^{HI}, of those tasks summed to $CF \cdot CP \cdot U$, where CF is the Criticality Factor (default $CF = 2.0$) characterizing the ratio between the total *HI*-criticality and total *LO*-criticality utilization of the *HI*-criticality tasks ($CF = U_{HI}^{HI}/U_{HI}^{LO}$). Similarly, *HI*-criticality utilization values, $U_i(HI)$, were generated for the N_{LO} *LO*-criticality tasks, such that the total *HI*-criticality utilization, U_{LO}^{HI}, of those tasks summed to $XF \cdot (1 - CP) \cdot U$, where XF is the Compensating Factor (default $XF = 0.5$) characterizing the ratio between the total *HI*-criticality and total *LO*-criticality utilization of the *LO*-criticality tasks ($XF = U_{LO}^{HI}/U_{LO}^{LO}$). For *HI*-criticality tasks, the $U_i(HI)$ values were constrained to be in the range $[U_i(LO), 1.0]$, and for *LO*-criticality tasks, the $U_i(HI)$ values were constrained to be in the range $[0.0, U_i(LO)]$. Note that the total utilization in normal mode is always equal to the target utilization value, $U_{HI}^{LO} + U_{LO}^{LO} = U$. Further, the total utilization in degraded mode is $U_{HI}^{HI} + U_{LO}^{HI} = U \cdot CP \cdot CF + U \cdot (1 - CP) \cdot XF$. Hence, if $CP \cdot (CF - 1) = (1 - CP) \cdot (1 - XF)$, then the overall utilization in degraded mode also equates to U. In that case, the increase in the utilization of *HI*-criticality tasks in degraded mode is compensated for by an equivalent decrease in the utilization of *LO*-criticality tasks.

Task periods T_i were generated according to a log-uniform distribution [28] with a factor of 100 difference between the minimum and maximum possible period. This represents a spread of task periods from 10ms to 1 second, as found in many real-time systems. Task deadlines D_i were set equal to their periods T_i. The *LO*- and *HI*-criticality execution times of all tasks were given by $C_i(LO) = U_i(LO) \cdot T_i$ and $C_i(HI) = U_i(HI) \cdot T_i$ respectively.

5.2 Experiments

The experiments considered systems with target utilization U varied from 0.025 to 0.975 in steps of 0.025. For each target utilization value examined, 1000 task sets were generated (100 in the case of experiments using the weighted schedulability measure [12]). The experiments investigated the performance of the following schedulability tests and necessary conditions:

(1) **AMC-valid**: This is a necessary feasibility condition given the basic requirements of the AMC scheme [9]. This upper bound checks that the total *LO*-criticality utilization of all tasks is feasible, i.e. $U_{HI}^{LO} + U_{LO}^{LO} \leq 1$, and that the total *HI*-criticality utilization of *HI*-criticality tasks is feasible, i.e. $U_{HI}^{HI} \leq 1$.

(2) **AMC-ubhl**: This is a necessary condition for schedulability given the basic requirements of the AMC scheme [9], assuming fixed-priority preemptive scheduling. This upper bound uses standard response time analysis for fixed-priority preemptive scheduling [6, 41] to check: (i) if all of the tasks are schedulable in normal mode, and (ii) if all of the *HI*-criticality tasks are schedulable in degraded mode, assuming that no releases of *LO*-criticality jobs take place in that mode. It ignores the impact of the mode change transition.

(3) **AMC-max**: Uses the AMC-max test [9] to determine task set schedulability under the AMC scheme.

(4) **AMC-rtb**: Uses the AMC-rtb test [9] to determine task set schedulability under the AMC scheme.

(5) **C-AMC-valid**: This is a necessary feasibility condition given the basic requirements of the C-AMC scheme. This upper bound checks that the total *LO*-criticality utilization of all tasks is feasible, i.e. $U_{HI}^{LO} + U_{LO}^{LO} \leq 1$, and that the total *HI*-criticality utilization of all tasks is feasible, i.e. $U_{LO}^{HI} + U_{HI}^{HI} \leq 1$.

(6) **C-AMC-ubhl**: This is a necessary condition for schedulability given the basic requirements of the C-AMC scheme, assuming fixed-priority preemptive scheduling. This upper bound uses standard response time analysis for fixed-priority preemptive scheduling [6, 41] to check: (i) if all of the tasks are schedulable in normal mode (i.e. assuming $C_i(LO)$ values), and (ii) if all of the tasks are schedulable in degraded mode (i.e. assuming $C_i(HI)$ values). It ignores the impact of the mode change transition.

(7) **C-AMC-max**: Uses the C-AMC-max test, see Section 4.2, to determine task set schedulability under the C-AMC scheme.

(8) **C-AMC-rtb**: Uses the C-AMC-rtb test, see Section 4.1, to determine task set schedulability under the C-AMC scheme.

(9) **FPPS**: Uses standard response time analysis for fixed-priority preemptive scheduling [6, 41] to determine if all of the tasks are schedulable assuming that the execution time of each task τ_i is given by $\max(C_i(LO), C_i(HI))$; in other words assuming the worst-case single criticality behavior.

In each case, Audsley's Optimal Priority Assignment algorithm [5] was used to assign priorities, ensuring an optimal priority assignment with respect to each schedulability test.

Observe that the following dominance relationships exist between the schedulability tests, as discussed in Section 4, and trivially extended to the upper bounds and FPPS: **AMC-test** $\rightarrow$ **C-AMC-test**, where $S \rightarrow Z$ indicates that test S dominates test Z, and **test** is one of **valid**, **ubhl**, **max**, or **rtb**. Further, **SCHED-valid** $\rightarrow$ **SCHED-ubhl** $\rightarrow$ **SCHED-max** $\rightarrow$ **SCHED-rtb** $\rightarrow$ **FPPS**, where **SCHED** is the scheduling scheme, either **AMC** or **C-AMC**.

5.3 Results

The figures illustrating the results are best viewed in color.

In the first experiment, we compared the performance of the various schedulability tests using the default parameters given in Section 5.1. The *Success Ratio*, i.e. the percentage of task sets generated that were deemed schedulable, is shown for each of the schedulability tests in Figure 1. The relative performance of the various tests follows the dominance relations set out in the previous section. Considering the C-AMC scheme, the C-AMC-max analysis shows a small but useful advantage over C-AMC-rtb, while both substantially outperform the single criticality approach to ensuring that all deadlines are met, i.e. FPPS. Observe that under the C-AMC scheme, with the default parameters ($CP = 0.5, CF = 2.0, XF = 0.5$), the upper bound on task set feasibility (validity) occurs at $U = 0.8$, compared to $U = 1.0$ for the AMC scheme. This is because under C-AMC, the utilization in degraded mode includes contributions from both *HI*- and *LO*-criticality tasks, i.e. $U_{HI}^{HI} + U_{LO}^{HI} = U \cdot CP \cdot CF + U \cdot (1 - CP) \cdot XF = 1.25U$.

In the second set of experiments, we used the weighted schedulability measure [12] to assess schedulability test performance while varying an additional parameter. In these experiments, the other parameters were set to the default values given in Section 5.1. In all of the weighted schedulability experiments the relative performance of the different tests follows the pattern illustrated in the first experiment, as dictated by the dominance relationships.

The results of varying the Criticality Proportion CP, from 0.0 to 1.0 in steps of 0.05, are shown in Figure 2. Recall that the Criticality Proportion determines the proportion of tasks that are HI-criticality. Observe that with a smaller proportion of HI-criticality tasks, in the range $[0.1, 0.4]$ the C-AMC-max and C-AMC-rtb tests are able to provide significant gains over the single criticality approach (FPPS). This is a result of the substantial reduction in workload due to executing imprecise versions of LO-criticality tasks in degraded mode. Further, when $CP = 0$, i.e. there are no HI-criticality tasks, or when $CP = 1$, i.e. there are no LO-criticality tasks, then the C-AMC-ubhl, C-AMC-max, C-AMC=rtb, AMC-ubhl, AMC-max, and AMC=rtb tests all reduce to the standard response time test for FPPS. Notice also that the limit on all systems being feasible (valid) is lower under C-AMC ($CP = 0.333$) than under AMC ($CP = 0.5$) due to the increased utilization that is supported in degraded mode.

The results of varying the Criticality Factor CF, from 1.0 to 3.0 in steps of 0.1, are shown in Figure 3. Recall that the Criticality Factor characterizes the ratio of total HI-criticality task utilization in degraded mode to that in normal mode, i.e. $CF = U_{HI}^{HI}/U_{LO}^{HI}$. The form of this graph is similar to that for CP shown in Figure 2. Akin to having a smaller proportion of HI-criticality tasks, having a smaller Criticality Factor, in the range $[1.1, 1.8]$ instead of 2.0 (the default), results in a smaller workload from HI-criticality tasks in degraded mode and ensures that the reduction in workload from LO-criticality tasks compensates sufficiently to provide substantially better schedulability than assuming a single criticality model, i.e. FPPS. Notice that when $CF = 1.0$, the workload from HI-criticality tasks is no higher in degraded mode, in fact that mode is never actually entered, and the C-AMC-ubhl, C-AMC-max, C-AMC-rtb, AMC-ubhl, AMC-max, and AMC-rtb tests all reduce to the standard response time test for FPPS. Notice also that the limit on all systems being feasible (valid) is lower under C-AMC ($CF = 1.5$) than under AMC ($CF = 2.0$) due to the increased total utilization supported in degraded mode.

The results of varying the Compensation Factor XF, from 0.0 to 1.0 in steps of 0.05, are shown in Figure 4. Recall that the Compensation Factor characterizes the ratio of total LO-criticality task utilization in degraded mode to that in normal mode, i.e. $XF = U_{LO}^{HI}/U_{LO}^{LO}$. Since the tests for the AMC model do not consider the execution of LO-criticality tasks in degraded mode, they are unaffected by the values of XF, hence the horizontal lines on the graph. By contrast, with the C-AMC model, smaller numeric values for XF correspond to a smaller workload due to LO-criticality tasks in degraded mode and hence better schedulability. Values for the Compensation Factor in the range $[0.0, 0.5]$ equate to a 2-fold or more reduction in workload, which provides substantial gains in schedulability compared to assuming a single criticality model, i.e. FPPS.

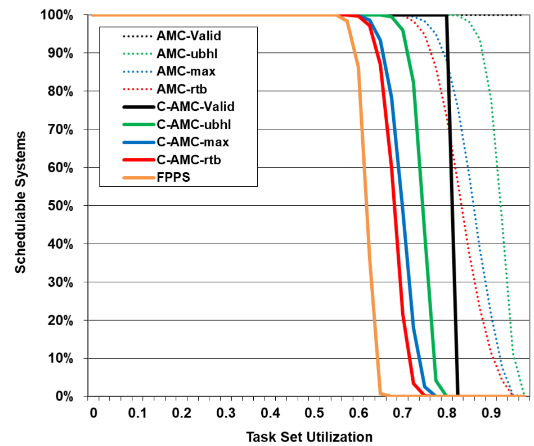

Figure 1: Success Ratio: Varying task set utilization.

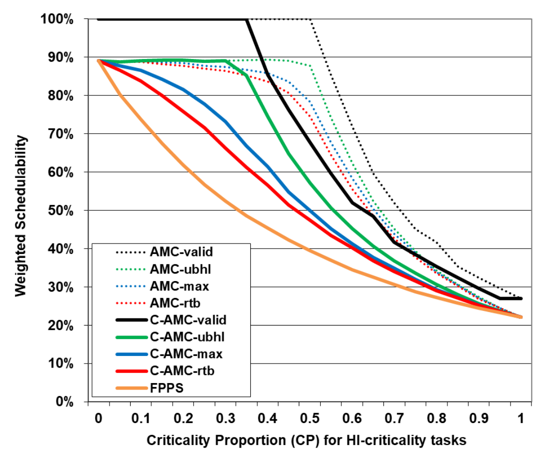

Figure 2: Weighted Schedulability: Varying CP.

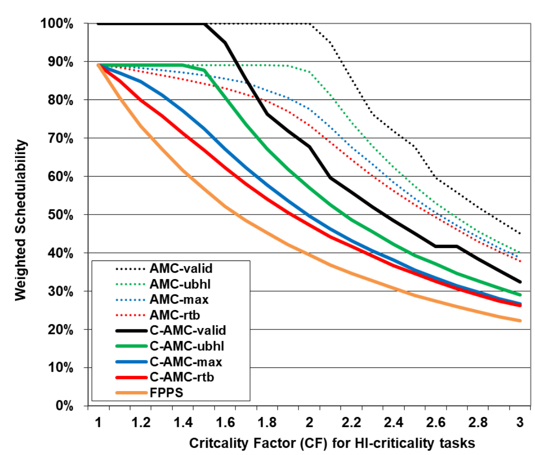

Figure 3: Weighted Schedulability: Varying CF.

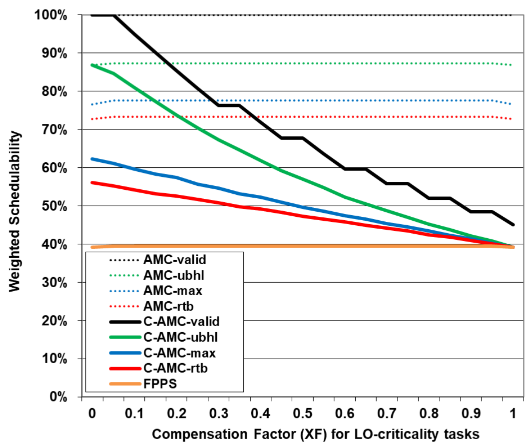

Figure 4: Weighted Schedulability: Varying XF.

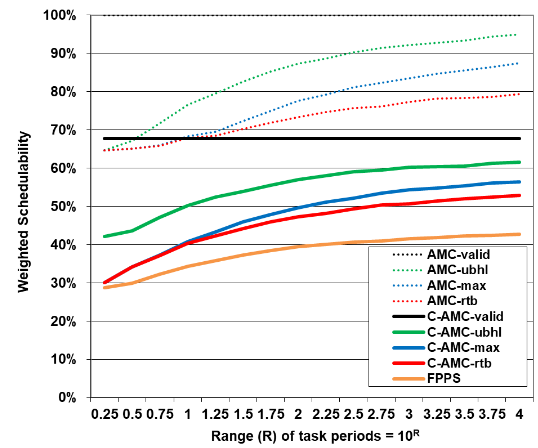

Figure 7: Weighted Schedulability: Varying period range.

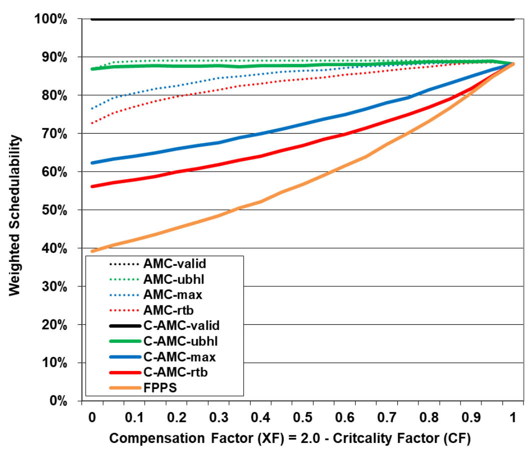

Figure 5: Weighted Schedulability: Varying both XF and CF.

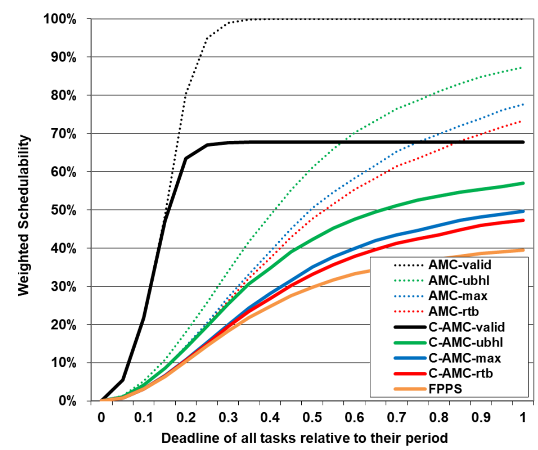

Figure 8: Weighted Schedulability: Varying deadlines.

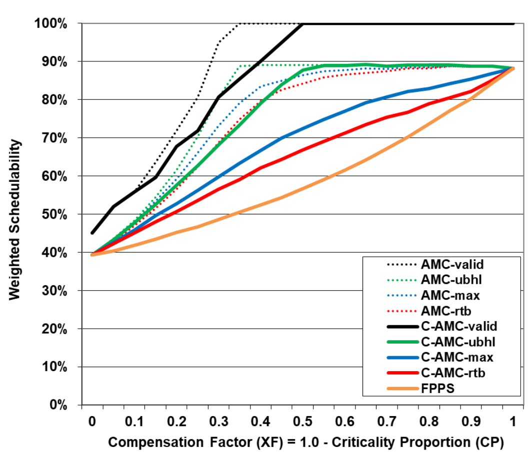

Figure 6: Weighted Schedulability: Varying both XF and CP.

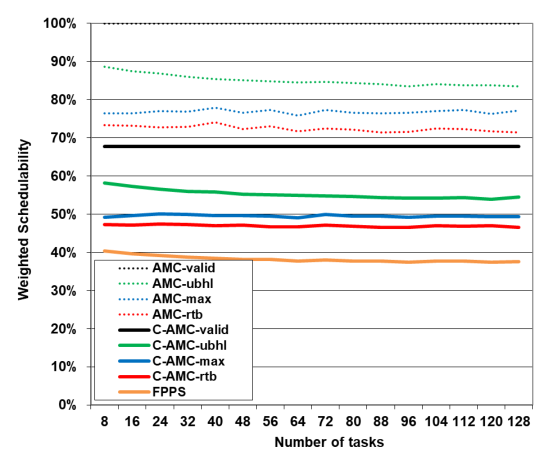

Figure 9: Weighted Schedulability: Varying number of tasks.

In Figure 4, observe that when $XF = 0$, meaning that LO-criticality jobs are not released in degraded mode, schedulability according to the C-AMC-max test remains below that for the AMC-max test, and similarly schedulability according to the C-AMC-rtb test remains below that for the AMC-rtb test. This is because under the C-AMC model, LO-criticality tasks released in normal mode, but completing in degraded mode are afforded schedulability guarantees, whereas under the AMC model they are not. Note, the steps in the line for feasibility (validity) under the C-AMC scheme are due to the precise quantization of the utilization of the generated tasks sets[3].

The results of varying both the Compensation Factor XF, from 0.0 to 1.0 in steps of 0.05, and simultaneous varying the Criticality Factor CF in opposition to it such that $CF = 2.0 - XF$, are shown in Figure 5. The idea being to examine how schedulability changes when the total utilization in degraded mode, i.e. $U^{HI}_{HI} + U^{HI}_{LO}$, is held constant at U, but the workload due to HI-criticality tasks (controlled by CF) is decreased from U to zero, while the workload from LO-criticality tasks (controlled by XF) is increased from zero to U. As expected, schedulability is maximized when $CF = XF = 1$ and the behavior reduces to that of a single criticality system with no change in execution times between normal and degraded mode. Observe that for decreasing values of XF and hence increasing values of CF, schedulability degrades; however, the C-AMC-max and C-AMC-rtb tests are still able to provide substantially improved performance compared to a single criticality system, i.e. FPPS. Notice that the upper bounds C-AMC-ubhl and AMC-ubhl are almost horizontal lines in Figure 5, this is because those upper bounds consider schedulability in normal and degraded mode separately. Hence, they are unaffected by the increased difficulty in ensuring schedulability across the mode change transition when there are large changes in the execution times of tasks between the two modes.

In Figure 5, as in Figure 4, observe that when $XF = 0$, meaning that LO-criticality jobs are not released in degraded mode, schedulability according to the C-AMC-max test remains below that for the AMC-max test, and similarly for the C-AMC-rtb and AMC-rtb tests. This is because under the C-AMC model, LO-criticality tasks released in normal mode, but completing in degraded mode are afforded schedulability guarantees, whereas under the AMC model, they are not.

The results of varying the Compensation Factor XF, from 0.0 to 1.0 in steps of 0.05, and simultaneous varying the Criticality Proportion CP in opposition to it, such that $CP = 1.0 - XF$, are shown in Figure 6. Note that for this specific experiment, the Criticality Factor CF was set to 1.5. The idea being to examine how schedulability changes when the total utilization in degraded mode, i.e. $U^{HI}_{HI} + U^{HI}_{LO}$, is held constant at U, but the workload due to HI-criticality tasks (controlled by CP) is decreased from U to zero, while the workload due to LO-criticality tasks (controlled by XF) is increased from zero to U. As expected, schedulability is maximized when $CP = 0$ and $XF = 1$ and the behavior reduces to that of a single criticality system with no change in execution times between normal and degraded mode. As also expected schedulability improves for increasing values of XF and hence decreasing values of CP, corresponding to smaller differences between the behavior in normal and degraded mode. At either extreme, the system reduces to one of single criticality (either all HI-criticality tasks or all LO-criticality tasks), hence C-AMC-ubhl, C-AMC-max, and C-AMC-rtb, all reduce to the same performance as FPPS. Nevertheless, for intermediate values of XF and CP, corresponding to mixed criticality systems, the C-AMC-max and C-AMC-rtb tests are able to provide substantially improved performance compared to a single criticality system, i.e. FPPS.

The results of varying the range of task periods, 10^R, for R from 0.25 to 4 in steps of 0.25, are shown in Figure 7. This equates to the range of task periods varying from 1.78 to 10,000. As expected with scheduling policies based on fixed priorities, the general trend for all of the schedulability tests is gradually increasing performance with an increasing range of task periods. Observe that for the smallest period ranges, e.g. $R = 0.25$, where the maximum and minimum periods differ only by a factor of 1.78, the performance of both C-AMC-max and C-AMC-rtb tends to that for FPPS. The reason for this is that when all tasks have essentially the same periods, then both the C-AMC-max and C-AMC-rtb analyses include interference due to one job of each higher priority task at its larger execution value, i.e. $C_j(HI)$ for a HI-criticality task and $C_j(LO)$ for a LO-criticality task, thus schedulability is effectively the same as for FPPS. (Note, in the case of C-AMC-max, this can be seen by considering $s = 0$ as the mode change time in the analysis).

The results of varying the task deadlines as a fixed proportion of their periods from 0 to 1.0 in steps of 0.05 are shown in Figure 8. As expected, all of the schedulability tests show gradually increasing performance with increasing deadlines, with the best performance obtained in the implicit deadline case, i.e. when $D_i = T_i$. The two necessary feasibility (validity tests) exhibit different behavior, since they check that the utilization does not exceed 1 in either normal or degraded mode, and also that the execution time of each task does not exceed its deadline, which may happen for very small deadlines.

Finally, we also investigated varying the task set cardinality from 8 to 128 in steps of 8. The results of this experiment are shown in Figure 9. Observe that all of the schedulability tests exhibit performance that is largely independent of the number of tasks.

6 CONCLUSIONS

Academic research into mixed criticality systems often assumes that if high-criticality tasks continue to execute beyond the execution time limits at which they would normally finish, then further workload due to low-criticality tasks should be dropped in order to ensure that the high-criticality tasks can still meet their deadlines. Industry, however, takes a different view of the importance of low-criticality tasks, with many practical systems unable to tolerate the complete abandonment of such tasks.

The research presented in this paper focuses on the above issue by introducing the Compensating Adaptive Mixed Criticality scheduling scheme. The C-AMC scheme ensures that both high- and low-criticality tasks meet their deadlines in both normal and degraded modes. Under C-AMC, jobs of low-criticality tasks, released in degraded mode, execute imprecise versions that are able to provide outputs of sufficient quality, while also reducing

[3] Re-drawing this line for larger numbers of task sets makes no difference.

the overall workload. This compensates, at least in part, for the overload due to high-criticality tasks, which while always executing their primary versions, may also run error handling code that is not expected to execute during normal operation.

Two schedulability tests, C-AMC-max and C-AMC-rtb, were derived for the C-AMC scheme and shown, via extensive evaluation across a wide range of different parameter settings, to provide substantially improvements in schedulability compared to a single criticality baseline that reflects current industry practice. Since C-AMC is based on fixed-priority preemptive scheduling, it provides a viable migration path along which industry can make an evolutionary transition to adaptive mixed criticality systems.

In future, we intend to build on earlier work with industry that examined the application of research into mixed-criticality systems to a DAL-A aircraft engine Full Authority Digital Engine Controller (FADEC) [43]. While the standard AMC scheme was initially prototyped as a solution, the new C-AMC scheme provides significant advantages, not least the ability to provide genuine graceful degradation by continuing to execute imprecise versions of low-criticality tasks and ensuring that their deadlines are met. In short, the C-AMC scheme provides engineers with significant additional flexibility in the design of mixed criticality systems.

APPENDIX: TASK ALLOCATION

In this appendix we explore the improvements in schedulability that can be achieved, for mixed criticality multi-core systems that make use of partitioned C-AMC or partitioned FPPS scheduling, by allocating tasks using Simulated Annealing. Note, here we make the simplifying assumption that the multi-core hardware platform provides full isolation between the different cores, and thus that there is no cross-core contention or interference, hence schedulability on each core depends only on the tasks allocated to that core. The system model assumed is thus effectively the same as that described in Section 3; however, instead of a single-core processor, there is multi-core processor, with m homogeneous cores, each of which independently executes the set of mixed-criticality tasks assigned to it. The task allocation problem considers how best to assign tasks to cores such that the tasks allocated to each core are schedulable according to independent (i.e. partitioned) C-AMC or FPPS scheduling on that core.

6.1 Simulated Annealing

Simulated Annealing relies on two key functions, a `Cost_Function` that determines the quality of each possible solution, and a `Modify_Function` that makes a randomly chosen, but valid modification to the current solution, in order to create a new solution that is close to it.

For Simulated Annealing to be effective, it is important that the `Cost_Function` provides a smooth and continuous metric, indicative of solution quality, that can drive the search towards an optimal solution. In the context of task allocation, we use the processor speed scaling factor F [51]. For a given allocation of tasks to cores, the `Cost_Function` determines the smallest value of F such that the execution times of all tasks can be scaled by a factor of $1/F$ (alternatively, the periods and deadlines can be scaled by a factor of F) and the system remains schedulable. This metric optimizes both schedulability and robustness, since F takes its smallest value for the task allocation that can tolerate the processor running at the lowest possible speed.

The processor speed scaling factor provides a continuous metric, that is at or below 1.0 for schedulable task allocations, and above that value for unschedulable allocations. The value of F is calculated via a binary search, in conjunction with an appropriate schedulability test. As a starting point, the binary search requires minimum and maximum bounds. These can be determined as follows: (i) the minimum bound is such that the scaled deadline for one of the tasks is reduced to its execution time, (ii) the maximum bound is such that the execution times of all tasks fit within the smallest scaled deadline of any task. Any value of F smaller than the minimum bound is guaranteed to result in an unschedulable system, whereas a value of F equal to the maximum bound is guaranteed to result in a schedulable system, given that the deadlines are constrained ($D_i \leq T_i$).

It is essential that the `Modify_Function` is able to span the search space, otherwise the algorithm may be unable to ever find the optimal solution. In the case of the task allocation problem, it must be possible, via repeated application of the `Modify_Function` to move from any valid task allocation to any other one. Our implementation of the `Modify_Function` makes one of two possible changes to an existing allocation: (i) it selects a task at random and changes its allocated core to a randomly selected different core, (ii) it selects two different tasks at random that are allocated to different cores, and swaps their allocation around[4]. The single task modification is randomly selected 20% of the time, with swapping selected the remaining 80% of the time.

The Simulated Annealing algorithm operates via two nested loops. The outer loop represents a series of reducing temperatures, used in the choices that the algorithm makes. In the experiments, the initial `temperature` was set to 1.0, and the final `min_temperature` to 0.01. Further, the `cooling_factor` was set to 0.95499, which results in 100 iterations of the outer loop. The inner loop iterates 50 times at each temperature. Thus the algorithm explores 5000 allocations in all, starting from an initial allocation of tasks to cores. In the experiments, the initial allocation was taken directly from the system generation, with an equal number of tasks, with equal total utilization, assigned to each core.

Simulated Annealing explores the search space by making modifications to an existing allocation via the `Modify_Function`, and then determining the quality of the new allocation formed via the `Cost_Function`. If the new allocation is an improvement on the best allocation seen so far then it is saved. If the new allocation is an improvement on the current one, then it becomes the current allocation, which the algorithm will continue searching from. If the new allocation does not represent an improvement, then there is still a chance that it will be accepted, and hence built upon. The probability of acceptance depends on how much worse the allocation has become, and the current temperature. Initially, when the temperature is high, new allocations can be accepted that are substantially worst than the current allocation. This helps to avoid the search becoming stuck in a local optimum. As the temperature

[4]In the unlikely event that all tasks are allocated to the same core, then a null swap is performed that does not modify the allocation.

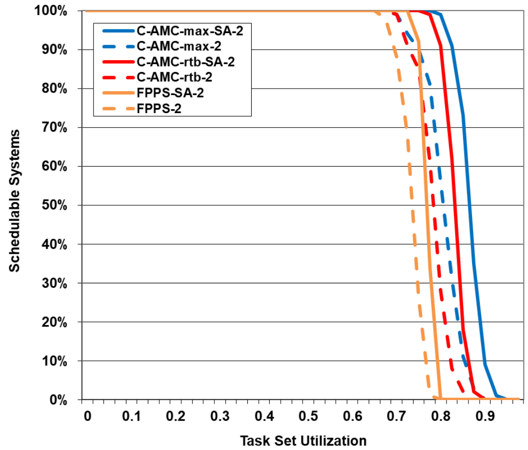

Figure 10: Success Ratio: Simulated Annealing for 2 cores.

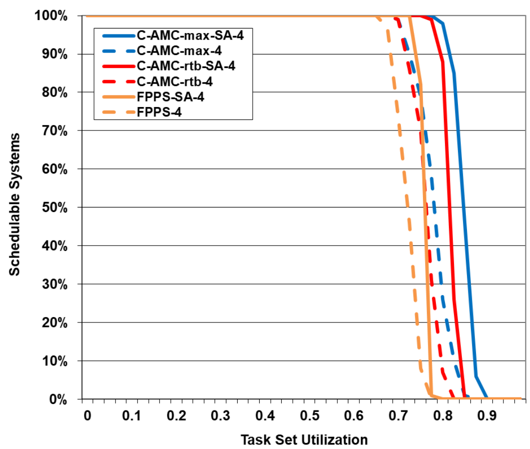

Figure 11: Success Ratio: Simulated Annealing for 4 cores.

decreases, only smaller negative steps are likely to be accepted, until at very low temperatures, the algorithm effectively behaves like a hill-climbing search, only accepting improved allocations.

6.2 Task Allocation Experiments

We compared the performance of the default initial assignment of tasks to cores, which has task sets with the exact same target utilization U assigned to each core, with that obtained by Simulated Annealing starting from the initial assignment. Since Simulated Annealing involves many trial allocations, we reduced the number of systems generated per utilization level from 1000 to 100. This was done to ensure that the overall runtime remained manageable[5].

Each system comprised NM tasks, with a different set of N tasks, with total utilization U, initially allocated to each of the M cores. By default $N = 10$, hence each 2 core system had 20 tasks in total, and each 4 core system 40 tasks in total. Task sets were generated with the following parameter settings: (i) $CP = 0.5$, so half of the tasks were HI-criticality and half were LO-criticality, and $CF = 1.5$ and $XF = 0.5$ so that the increase in workload in degraded mode due to HI-criticality tasks was balanced by the reduction in workload due to LO-criticality tasks. Other task parameters were set as described in Section 5.1. Audsey's Optimal Priority Assignment (OPA) algorithm [5] was used with each of the schedulability tests, since this was shown, in Section 4.3, to be optimal in each case.

The Simulated Annealing algorithm started from the initial allocation and was able to re-allocate tasks to different cores in order to improve overall system schedulability. For a system to be schedulable, the task sets on each of its cores had to be schedulable. While the initial allocation comprised task sets of equal utilization on each core, this was not necessarily the case with the final allocation obtained via Simulated Annealing.

We compared the effectiveness of the task allocations generated by Simulated Annealing for three schedulability tests: **C-AMC-max**, **C-AMC-rtb**, and **FPPS** as described in Section 5.2.

Figures 10 and 11 illustrate the effectiveness of the allocations produced by Simulated Annealing for 2 cores and for 4 cores respectively. The results for Simulated Annealing are labelled **C-AMC-max-SA-*m***, **C-AMC-rtb-SA-*m***, and **FPPS-SA-*m*** respectively, where m denotes the number of cores, either 2 or 4, and are compared to the results for the baseline allocation, labelled **C-AMC-max-*m***, **C-AMC-rtb-*m***, and **FPPS-*m*** respectively.

Figures 10 and 11 show that Simulated Annealing is able to improve schedulability, compared to the baseline, for each of the schedulability tests and numbers of cores considered. Observe that the improvement obtained is substantially larger for the C-AMC-max and C-AMC-rtb tests than it is for FPPS. This is because when used in conjunction with the C-AMC scheme, Simulated Annealing is able to find allocations that minimize the additional interference encountered across the mode change transition, hence improving schedulability.

Table 1: Number of additional schedulable systems found using Simulated Annealing for task allocation.

	Test	Extra with SA	
2 cores	C-AMC-max	243	6.1%
	C-AMC-rtb	198	5.0%
	FPPS	145	3.6%
4 cores	C-AMC-max	272	6.8%
	C-AMC-rtb	220	5.5%
	FPPS	158	4.0%

The number of additional systems that were found schedulable using the allocations determined by Simulated Annealing are listed in Table 1, both as a number out of 4000 systems in total, and as a percentage. Observe that the gains obtained by using Simulated Annealing are slightly larger with 4 cores than with 2 cores. This is because the larger systems present more opportunities for task allocations that improve schedulability.

[5]The Simulated Annealing algorithm was configured to iterate 5000 times. On each iteration the schedulability test was run approximately 10 times to determine the processor speed scaling factor via binary search. Hence, to analyse 100 systems requires approximately 5,000,000 schedulability tests.

ACKNOWLEDGMENTS

This research was funded in part by Innovate UK HICLASS project (113213) EPSRC Research Data Management: No new primary data was created during this study.

REFERENCES

[1] Yasmina Abdeddaïm. 2020. Accurate Strategy for Mixed Criticality Scheduling. In *Verification and Evaluation of Computer and Communication Systems - 14th International Conference, VECoS 2020, Xi'an, China, October 26-27, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12519)*, Belgacem Ben Hedia, Yu-Fang Chen, Gaiyun Liu, and Zhenhua Yu (Eds.). Springer, 131–146. https://doi.org/10.1007/978-3-030-65955-4_10

[2] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2020. An Empirical Survey-based Study into Industry Practice in Real-time Systems. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*. IEEE, 3–11. https://doi.org/10.1109/RTSS49844.2020.00012

[3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2021. A comprehensive survey of industry practice in real-time systems. *Real-Time Systems* (11 Nov 2021), 41 pages. https://doi.org/10.1007/s11241-021-09376-1

[4] Sedigheh Asyaban and Mehdi Kargahi. 2018. An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems. *Real Time Syst.* 54, 1 (2018), 32–90. https://doi.org/10.1007/s11241-017-9287-2

[5] Neil C. Audsley. 2001. On priority assignment in fixed priority scheduling. *Inf. Process. Lett.* 79, 1 (2001), 39–44. https://doi.org/10.1016/S0020-0190(00)00165-4

[6] Neil C. Audsley, Alan Burns, Michael Richardson, Kenneth W. Tindell, and Andrew J. Wellings. 1993. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal* 8 (September 1993), 284–292(8). Issue 5. https://digital-library.theiet.org/content/journals/10.1049/sej.1993.0034

[7] Sanjoy Baruah, Alan Burns, and Robert I. Davis. 2013. An Extended Fixed Priority Scheme for Mixed Criticality Systems. In *Workshop on Real-Time Mixed Criticality Systems (ReTiMics) 2013, 21st August, Taipei, Taiwan*, Guiseppe Lipari Laurent George (Ed.). 18–24. https://www-users.cs.york.ac.uk/~robdavis/papers/jitterRTCSA.pdf

[8] Sanjoy K. Baruah and Alan Burns. 2011. Implementing Mixed Criticality Systems in Ada. In *Reliable Software Technologies - Ada-Europe 2011 - 16th Ada-Europe International Conference on Reliable Software Technologies, Edinburgh, UK, June 20-24, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6652)*, Alexander B. Romanovsky and Tullio Vardanega (Eds.). Springer, 174–188. https://doi.org/10.1007/978-3-642-21338-0_13

[9] Sanjoy K. Baruah, Alan Burns, and Robert I. Davis. 2011. Response-Time Analysis for Mixed Criticality Systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. IEEE Computer Society, 34–43. https://doi.org/10.1109/RTSS.2011.12

[10] Sanjoy K. Baruah, Alan Burns, and Zhishan Guo. 2016. Scheduling Mixed-Criticality Systems to Guarantee Some Service under All Non-erroneous Behaviors. In *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*. IEEE Computer Society, 131–138. https://doi.org/10.1109/ECRTS.2016.12

[11] Sanjoy K. Baruah and Bipasa Chattopadhyay. 2013. Response-time analysis of mixed criticality systems with pessimistic frequency specification. In *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2013, Taipei, Taiwan, August 19-21, 2013*. IEEE Computer Society, 237–246. https://doi.org/10.1109/RTCSA.2013.6732224

[12] Andrea Bastoni, Bjorn B. Brandenburg, and James H. Anderson. 2010. Cache-Related Preemption and Migration Delays: Empirical Approximation and Impact on Schedulability. In *International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*. 33–44.

[13] Iain Bate, Alan Burns, and Robert I. Davis. 2015. A Bailout Protocol for Mixed Criticality Systems. In *27th Euromicro Conference on Real-Time Systems, ECRTS 2015, Lund, Sweden, July 8-10, 2015*. IEEE Computer Society, 259–268. https://doi.org/10.1109/ECRTS.2015.30

[14] Iain Bate, Alan Burns, and Robert I. Davis. 2017. An Enhanced Bailout Protocol for Mixed Criticality Embedded Software. *IEEE Trans. Software Eng.* 43, 4 (2017), 298–320. https://doi.org/10.1109/TSE.2016.2592907

[15] Iain Bate, Alan Burns, and Robert I. Davis. 2022. Analysis-Runtime Co-design for Adaptive Mixed Criticality Scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2022, 4-6 May 2022, Milano, Italy*, Heechul Yun (Ed.). IEEE Computer Society, 14 pages.

[16] Konstantinos Bletsas, Muhammad Ali Awan, Pedro Souto, Benny Akesson, Alan Burns, and Eduardo Tovar. 2018. Decoupling Criticality and Importance in Mixed-Criticality Scheduling. In *Workshop on Mixed Criticality Systems (WMC)*. 25–30. https://drive.google.com/file/d/14sLpczS6wpQN99dPET08sZP3qjvEr5S2/view

[17] Alan Burns. 2014. System Mode Changes - General and Criticality-Based. In *2nd International Workshop on Mixed Criticality Systems (WMC), at the Real Time Systems Symposium (RTSS)*. 3–8. https://www-users.cs.york.ac.uk/~robdavis/wmc2014/1.pdf

[18] Alan Burns and Sanjoy Baruah. 2013. Towards A More Practical Model for Mixed Criticality Systems. In *Workshop on Mixed Criticality Systems (WMC)*. 1–6. https://www-users.cs.york.ac.uk/~robdavis/wmc2013/paper3.pdf

[19] Alan Burns and Robert I. Davis. 2013. Mixed Criticality on Controller Area Network. In *25th Euromicro Conference on Real-Time Systems, ECRTS 2013, Paris, France, July 9-12, 2013*. IEEE Computer Society, 125–134. https://doi.org/10.1109/ECRTS.2013.23

[20] Alan Burns and Robert I. Davis. 2014. Adaptive Mixed Criticality Scheduling with Deferred Preemption. In *Proceedings of the IEEE 35th IEEE Real-Time Systems Symposium, RTSS 2014, Rome, Italy, December 2-5, 2014*. IEEE Computer Society, 21–30. https://doi.org/10.1109/RTSS.2014.12

[21] Alan Burns and Robert I. Davis. 2018. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.* 50, 6 (2018), 82:1–82:37. https://doi.org/10.1145/3131347

[22] Alan Burns and Robert I. Davis. 2019. *Mixed Criticality Systems: A Review (12th Edition)*. Technical Report MCC-1(M), available at https://www-users.cs.york.ac.uk/~burns/review.pdf. Department of Computer Science, University of York.

[23] Alan Burns and Robert I. Davis. 2020. Schedulability Analysis for Adaptive Mixed Criticality Systems with Arbitrary Deadlines and Semi-Clairvoyance. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*. IEEE, 12–24. https://doi.org/10.1109/RTSS49844.2020.00013

[24] Alan Burns, Robert I. Davis, Sanjoy K. Baruah, and Iain Bate. 2018. Robust Mixed-Criticality Systems. *IEEE Trans. Computers* 67, 10 (2018), 1478–1491. https://doi.org/10.1109/TC.2018.2831227

[25] Robert I. Davis, Sebastian Altmeyer, and Alan Burns. 2018. Mixed Criticality Systems with Varying Context Switch Costs. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*, Rodolfo Pellizzoni (Ed.). IEEE Computer Society, 140–151. https://doi.org/10.1109/RTAS.2018.00024

[26] Robert I. Davis and Alan Burns. 2011. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real Time Syst.* 47, 1 (2011), 1–40. https://doi.org/10.1007/s11241-010-9106-5

[27] Robert I. Davis, Liliana Cucu-Grosjean, Marko Bertogna, and Alan Burns. 2016. A review of priority assignment in real-time systems. *J. Syst. Archit.* 65 (2016), 64–82. https://doi.org/10.1016/j.sysarc.2016.04.002

[28] Paul Emberson, Roger Stafford, and Robert I. Davis. 2010. Techniques For The Synthesis Of Multiprocessor Tasksets. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 6–11. http://retis.sssup.it/waters2010/waters2010.pdf

[29] Rolf Ernst and Marco Di Natale. 2016. Mixed Criticality Systems - A History of Misconceptions? *IEEE Des. Test* 33, 5 (2016), 65–74. https://doi.org/10.1109/MDAT.2016.2594790

[30] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. 2015. How realistic is the mixed-criticality real-time system model?. In *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, Julien Forget (Ed.). ACM, 139–148. https://doi.org/10.1145/2834848.2834869

[31] Tom Fleming and Alan Burns. 2013. Extending Mixed Criticality Scheduling. In *Workshop on Mixed Criticality Systems (WMC)*. 7–12.

[32] Tom Fleming and Alan Burns. 2014. Incorporating The Notion of Importance into Mixed Criticality Systems. In *Workshop on Mixed Criticality Systems (WMC)*. 33–38. https://www-users.cs.york.ac.uk/~robdavis/wmc2014/6.pdf

[33] Oliver Gettings, Sophie Quinton, and Robert I. Davis. 2015. Mixed criticality systems with weakly-hard constraints. In *Proceedings of the 23rd International Conference on Real Time Networks and Systems, RTNS 2015, Lille, France, November 4-6, 2015*, Julien Forget (Ed.). ACM, 237–246. https://doi.org/10.1145/2834848.2834850

[34] Paul Graydon and Iain Bate. 2013. Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. In *1st International Workshop on Mixed Criticality Systems (WMC), at the Real Time Systems Symposium (RTSS)*. 19–24. https://www-users.cs.york.ac.uk/~robdavis/wmc2013/paper14.pdf

[35] David Griffin, Iain Bate, and Robert I. Davis. 2020. *dgdguk/drs: v2.0.0*. https://doi.org/10.5281/zenodo.4264857 Also funded by Innovate UK HICLASS project (Ref. 113213) https://gtr.ukri.org/projects?ref=113213.

[36] David Griffin, Iain Bate, and Robert I. Davis. 2020. Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests. In *41st IEEE Real-Time Systems Symposium, RTSS 2020, Houston, TX, USA, December 1-4, 2020*. IEEE, 76–88. https://doi.org/10.1109/RTSS49844.2020.00018

[37] Biao Hu, Lothar Thiele, Pengcheng Huang, Kai Huang, Christoph Griesbeck, and Alois C. Knoll. 2019. FFOB: efficient online mode-switch procrastination in mixed-criticality systems. *Real Time Syst.* 55, 3 (2019), 471–513. https://doi.org/10.1007/s11241-018-9323-x

[38] Huang-Ming Huang, Christopher D. Gill, and Chenyang Lu. 2012. Implementation and Evaluation of Mixed-Criticality Scheduling Approaches for Periodic Tasks. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium, Beijing, China, April 16-19, 2012*, Marco Di Natale (Ed.). IEEE Computer Society, 23–32. https://doi.org/10.1109/RTAS.2012.16

[39] Pengcheng Huang, Pratyush Kumar, Nikolay Stoimenov, and Lothar Thiele. 2013. Interference Constraint Graph - A new specification for mixed-criticality systems. In *Proceedings of 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation, ETFA 2013, Cagliari, Italy, September 10-13, 2013*, Carla Seatzu (Ed.). IEEE, 1–8. https://doi.org/10.1109/ETFA.2013.6647967

[40] Saverio Iacovelli and Raimund Kirner. 2019. A Lazy Bailout Approach for Dual-Criticality Systems on Uniprocessor Platforms. *Designs* 3, 1 (2019), 1–26. https://doi.org/10.3390/designs3010010

[41] Mathai Joseph and Paritosh K. Pandya. 1986. Finding Response Times in a Real-Time System. *Comput. J.* 29, 5 (1986), 390–395. https://doi.org/10.1093/comjnl/29.5.390

[42] Stephen Law and Iain Bate. 2016. Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis. In *28th Euromicro Conference on Real-Time Systems, ECRTS 2016, Toulouse, France, July 5-8, 2016*. IEEE Computer Society, 189–199. https://doi.org/10.1109/ECRTS.2016.21

[43] Stephen Law, Iain Bate, and Benjamin Lesage. 2019. Industrial Application of a Partitioning Scheduler to Support Mixed Criticality Systems. In *31st Euromicro Conference on Real-Time Systems, ECRTS 2019, July 9-12, 2019, Stuttgart, Germany (LIPIcs, Vol. 133)*, Sophie Quinton (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:22. https://doi.org/10.4230/LIPIcs.ECRTS.2019.8

[44] Stephen Law, Iain Bate, and Benjamin Lesage. 2020. Justifying the Service Provided to Low Criticality Tasks in a Mixed Criticality System. In *28th International Conference on Real Time Networks and Systems, RTNS 2020, Paris, France, June 10, 2020*, Liliana Cucu-Grosjean, Roberto Medina, Sebastian Altmeyer, and Jean-Luc Scharbarg (Eds.). ACM, 100–110. https://doi.org/10.1145/3394810.3394814

[45] Joseph Y.-T. Leung and Jennifer Whitehead. 1982. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Evaluation* 2, 4 (1982), 237–250. https://doi.org/10.1016/0166-5316(82)90024-4

[46] Di Liu, Nan Guan, Jelena Spasic, Gang Chen, Songran Liu, Todor P. Stefanov, and Wang Yi. 2018. Scheduling Analysis of Imprecise Mixed-Criticality Real-Time Tasks. *IEEE Trans. Computers* 67, 7 (2018), 975–991. https://doi.org/10.1109/TC.2018.2789879

[47] Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor P. Stefanov, and Wang Yi. 2016. EDF-VD Scheduling of Mixed-Criticality Systems with Degraded Quality Guarantees. In *2016 IEEE Real-Time Systems Symposium, RTSS 2016, Porto, Portugal, November 29 - December 2, 2016*. IEEE Computer Society, 35–46. https://doi.org/10.1109/RTSS.2016.013

[48] Dorin Maxim, Robert I. Davis, Liliana Cucu-Grosjean, and Arvind Easwaran. 2017. Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS 2017, Grenoble, France, October 04 - 06, 2017*, Enrico Bini and Claire Pagetti (Eds.). ACM, 237–246. https://doi.org/10.1145/3139258.3139276

[49] Michael Paulitsch, Oscar Medina Duarte, Hassen Karray, Kevin Mueller, Daniel Münch, and Jan Nowotsch. 2015. Mixed-Criticality Embedded Systems - A Balance Ensuring Partitioning and Performance. In *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*. IEEE Computer Society, 453–461. https://doi.org/10.1109/DSD.2015.100

[50] Ivan Pavic and Hrvoje Dzapo. 2020. Commentary to: An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems. *Real Time Syst.* 56, 1 (2020), 112–119. https://doi.org/10.1007/s11241-020-09345-0

[51] Sasikumar Punnekkat, Robert I. Davis, and Alan Burns. 1997. Sensitivity Analysis of Real-Time Task Sets. In *Advances in Computing Science - ASIAN '97, Third Asian Computing Science Conference, Kathmandu, Nepal, December 9-11, 1997, Proceedings (Lecture Notes in Computer Science, Vol. 1345)*, R. K. Shyamasundar and Kazunori Ueda (Eds.). Springer, 72–82. https://doi.org/10.1007/3-540-63875-X_44

[52] Luca Santinelli and Zhishan Guo. 2018. A Sensitivity Analysis for Mixed Criticality: Trading Criticality with Computational Resource. In *23rd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2018, Torino, Italy, September 4-7, 2018*. IEEE, 313–320. https://doi.org/10.1109/ETFA.2018.8502493

[53] François Santy, Laurent George, Philippe Thierry, and Joël Goossens. 2012. Relaxing Mixed-Criticality Scheduling Strictness for Task Sets Scheduled with FP. In *24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11-13, 2012*, Robert Davis (Ed.). IEEE Computer Society, 155–165. https://doi.org/10.1109/ECRTS.2012.39

[54] Hang Su, Peng Deng, Dakai Zhu, and Qi Zhu. 2016. Fixed-Priority Dual-Rate Mixed-Criticality Systems: Schedulability Analysis and Performance Optimization. In *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016, Daegu, South Korea, August 17-19, 2016*. IEEE Computer Society, 59–68. https://doi.org/10.1109/RTCSA.2016.16

[55] Hang Su and Dakai Zhu. 2013. An elastic mixed-criticality task model and its scheduling algorithm. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, Enrico Macii (Ed.). EDA Consortium San Jose, CA, USA / ACM DL, 147–152. https://doi.org/10.7873/DATE.2013.043

[56] Vijaya Kumar Sundar and Arvind Easwaran. 2019. A Practical Degradation Model for Mixed-Criticality Systems. In *IEEE 22nd International Symposium on Real-Time Distributed Computing, ISORC 2019, Valencia, Spain, May 7-9, 2019*. IEEE, 171–180. https://doi.org/10.1109/ISORC.2019.00040

[57] Steve Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 239–243. https://doi.org/10.1109/RTSS.2007.47

[58] Marcus Völp, Michael Roitzsch, and Herman Härtig. 2015. Towards an Interpretation of Mixed Criticality for Optimistic Scheduling. In *Work-In-Progress Session 21st IEEE Real-Time and Embedded Technology and Applications Symposium*. 15–16.

[59] Qingling Zhao, Zonghua Gu, and Haibo Zeng. 2013. PT-AMC: integrating preemption thresholds into mixed-criticality scheduling. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, Enrico Macii (Ed.). EDA Consortium San Jose, CA, USA / ACM DL, 141–146. https://doi.org/10.7873/DATE.2013.042

[60] Qingling Zhao, Zonghua Gu, Haibo Zeng, and Nenggan Zheng. 2018. Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling. *J. Syst. Archit.* 83 (2018), 57–74. https://doi.org/10.1016/j.sysarc.2017.03.007

[61] Qingling Zhao, Mengfei Qu, Bo Huang, Zhe Jiang, and Haibo Zeng. 2022. Schedulability analysis and stack size minimization for adaptive mixed criticality scheduling with semi-Clairvoyance and preemption thresholds. *J. Syst. Archit.* 124 (2022), 102383. https://doi.org/10.1016/j.sysarc.2021.102383

[62] Yecheng Zhao and Haibo Zeng. 2017. An efficient schedulability analysis for optimizing systems with adaptive mixed-criticality scheduling. *Real Time Syst.* 53, 4 (2017), 467–525. https://doi.org/10.1007/s11241-017-9267-6