
Proceedings Paper:
Bate, Iain John orcid.org/0000-0003-2415-8219, Burns, Alan orcid.org/0000-0001-5621-8816 and Davis, Robert Ian orcid.org/0000-0002-5772-0928 (2022) Analysis-Runtime Co-design for Adaptive Mixed Criticality Scheduling. In: Proceedings 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS). 2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS), 04-06 May 2022 , pp. 187-200.

Analysis-Runtime Co-design for Adaptive Mixed Criticality Scheduling

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

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

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

***Abstract*—In this paper, we use the term "Analysis-Runtime Co-design" to describe the technique of modifying the runtime protocol of a scheduling scheme to closely match the analysis derived for it. Carefully designed modifications to the runtime protocol make the schedulability analysis for the scheme less pessimistic, while the schedulability guarantee afforded to any given application remains intact. Such modifications to the runtime protocol can result in significant benefits with respect to other important metrics. An enhanced runtime protocol is designed for the Adaptive Mixed-Criticality (AMC) scheduling scheme. This protocol retains the same analysis, while ensuring that in the event of high-criticality behavior, the system degrades less often and remains degraded for a shorter time, resulting in far fewer low-criticality jobs that either miss their deadlines or are not executed.**

***Index Terms*—Real-Time, Mixed Criticality, Fixed Priority, Schedulability Analysis**

I. Introduction

The role of schedulability analysis is to provide a priori guarantees that a real-time application will meet its timing constraints at runtime. Schedulability analysis is traditionally derived after a runtime protocol has been chosen. So, for example, Response-Time Analysis [1], [2] was developed for fixed priority scheduling, and Processor Demand Analysis [3] was developed for EDF scheduling. Ideally the analysis derived is *exact*, in other words both *sufficient* (pass the test and satisfy all deadlines) and *necessary* (fail the test and miss some deadline). Unfortunately, as many runtime protocols give rise to computationally complex scheduling problems, tractable exact analysis is often unobtainable. Inexact but sufficient analysis must be used instead. Such forms of analysis are evaluated in terms of how pessimistic they are, i.e. how likely they are to deem an application unschedulable that will in fact meet all of its deadlines under the worst-case conditions.

With this traditional approach, the runtime protocol is typically a given. Often it has been defined, or even standardized, with little regard given to its timing behavior or its analysis. It is not surprising then that exact schedulability analysis is often intractable. In this paper we investigate how the runtime protocol itself can be modified so that a form of analysis that is inexact when applied to the original protocol becomes more precise when applied to the modified protocol. Even in situations where exact analysis is still not achievable, if the level of pessimism in the analysis can be reduced then there can still be an overall benefit.

When the offline analysis does not change but the runtime protocol is modified in compliance with it, then obviously the schedulability guarantees afforded to applications by the analysis are unaffected; however, the runtime behavior and characteristics can be significantly enhanced. For example, resilience (fault tolerance) can be improved and runtime overheads may be reduced. If the analysis cannot take advantage of a particular element of the original runtime protocol then that element can be removed with no impact on the system's guaranteed worst-case performance. We refer to this approach, of utilising feedback from inexact schedulability analysis in the design of the runtime protocol, as ***Analysis-Runtime Co-design***, and employ it in this paper to improve upon the Adaptive Mixed-Criticality (AMC) scheduling scheme [4].

The AMC scheduling scheme and its analysis are applicable to mixed criticality systems that are implemented using fixed priority scheduling on single-core processors. On detecting high-criticality behavior, the runtime protocol allows the system to switch from a mode in which all tasks release jobs, into a degraded mode in which only the high-criticality tasks are permitted to release jobs. Analysis of the AMC scheme makes pessimistic assumptions about the state of the system when these mode changes occur (see Section IV-B for details). By modifying the runtime protocol to closely reflect the analysis, we demonstrate that the system degrades less often and remains in the degraded mode for a shorter time, resulting in far fewer low-criticality jobs that either miss their deadlines or are not executed.

Research into real-time scheduling often seeks to narrow the gap between the guarantees provided by the schedulability analysis employed and the precise behavior of the runtime protocol (i.e. exact schedulability) by improving the analysis, often at the expense of making it intractable. By contrast, the *analysis-runtime co-design* approach proposed in this paper retains simple tractable analysis and the real-time guarantees that it provides, while modifying the runtime protocol in such a way that improves other important performance metrics. The tradeoff is that the gap between the guarantees provided by the analysis and the precise behavior of the modified

runtime protocol is narrowed via a reduction in schedulability according to a theoretical exact test. However, with no viable means of determining exact schedulability, this disadvantage may be hypothetical, while the advantages constitute practical improvements in performance.

The motivation for the specific research presented in this paper comes from mixed criticality systems in avionics. The avionics industry has a strong preference for simple scheduling policies and analyses, with those based on fixed priorities most commonly used [5], [6]. For mixed criticality systems, AMC schemes are being adopted [7], [8], underpinned by the simple yet effective AMC-rtb schedulability analysis [4], which is considered good enough for industrial use. In this research, we employ analysis-runtime co-design to modify the runtime protocol for AMC in such a way that it retains compatibility with the AMC-rtb schedulability test, while enabling substantial improvements in metrics related to low-criticality task performance. Using the modified runtime protocol reduces the number of times that degraded mode is entered, the total time in degraded mode, and most importantly the number of low-criticality jobs not executed or missing their deadlines to 16.8%, 1.7%, and 2.5% respectively of their values with the original AMC runtime protocol[1]. Ultimately, this is a compromise, since exact schedulability is reduced with the modified runtime protocol compared to the original, see Appendix A for details. From an industry perspective this compromise is worthwhile, since the advantages in improved runtime performance outweigh the marginal gains in schedulability that would, in any case, necessitate deployment of more complex schedulability tests.

The remainder of the paper is organized as follows: Section II discusses related work. Section III introduces the system model, terminology, and notation. Section IV presents the modified runtime protocol for the AMC scheme, leveraging analysis-runtime co-design to improve its effectiveness. A scenario-based evaluation of the performance of the modified protocol is given in Section V. Finally, Section VI concludes with a summary and directions for future research.

II. 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 [9] in 2007, mixed criticality systems have become a hot topic of real-time systems research, see [10], [11] for a comprehensive survey. Many of these papers focus on scheduling schemes that are based on fixed priorities, most notably Static Mixed Criticality (SMC) [12] and Adaptive Mixed Criticality (AMC) [4]. AMC is considered the most effective fixed priority scheme [13], and has been extended to account for many additional aspects including: preemption thresholds [14], [15], multiple criticality levels [16], criticality-specific periods [17], changes in priority [18], communications [19], deferred preemption [20], weakly-hard timing constraints [21], probabilistic task models [22], context switch costs [23], and robust [24] and semi-clairvoyant [25] timing behavior. An exact analysis has also been developed for periodic task sets with offsets [26], [27].

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 [12]; extending periods and/or deadlines [28], [29]; reducing execution times by switching to simpler versions [30]; dropping jobs from specific tasks [31]–[33]; and applying weakly-hard constraints, allowing some jobs to be skipped [21]. 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 [34] to increase all low-criticality execution time budgets while still retaining a schedulable system [30], [35]–[37].

Online accounting for budget under and overruns can also be used to delay switching to degraded mode [38]. Further, the time spent in the degraded mode can be reduced via online budget accounting resulting in a faster bailout [39], [40] and recovery. Finally, 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 [41]. This mechanism is orthogonal to the fixed priority scheduling scheme used and can be applied to both AMC [4] and the Bailout Protocol [39], [40].

Other work considers task-level [31] rather than system-level mode changes, which selectively restricting releases of some low-criticality tasks rather than all of them.

There are disparate views within the real-time systems community as to the timing requirements for mixed criticality systems, while most works assume that low-criticality tasks do not have to meet their deadlines and new releases can potentially be dropped as part of graceful degradation, others do not [42], [43]. In this paper, we assume that abandoning new releases of low-criticality tasks is acceptable when system behavior diverges from what is normally expected.

III. System Model

In this paper, we assume a mixed criticality system executing on a single-core processor under various schemes based on fixed priority preemptive scheduling.

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 LO-criticality task τ_j has a single estimate $C_j(LO)$ of its Worst-Case Execution Time (WCET). By contrast, each HI-criticality task τ_k has two estimates $C_k(LO)$ and $C_k(HI)$ of its WCET, where $C_k(HI) \geq C_k(LO)$. Each task τ_i has a minimum inter-arrival time or period T_i between releases of its

[1]Results based on a comparison of the mean values obtained in experiments on 500 task sets with semi-harmonic periods, representative of those found in automotive and avionics systems, comparisons made between the original runtime protocol (AMC+) and the modified runtime protocol (AMC-RH), see Section V-E, Figures 1–3 for further details.

jobs, and a constrained relative deadline D_i, where $D_i \leq T_i$. 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 or equal) priority than task τ_i.

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 τ_i that does not complete within its execution time budget. This budget is set to $C_i(LO)$ for a LO-criticality task and $C_i(HI)$ for a HI-criticality task.

HI- and LO-criticality tasks have different requirements in terms of the level of assurance required for their timing guarantees. In any interval of time when the processor is busy executing tasks:

R1 If all jobs of the tasks comply with their LO-criticality WCET estimates $C_i(LO)$, then all jobs must be guaranteed to meet their deadlines.

R2 If a job of a HI-criticality task τ_i executes for its LO-criticality WCET estimate $C_i(LO)$ without signaling completion, then only HI-criticality tasks are required to meet their deadlines.

These requirements give rise to the concept of a *normal* mode during which all tasks must meet their deadlines, and an *abnormal* mode during which only HI-criticality tasks need meet their deadlines. Abnormal mode extends from the time at which a job of a HI-criticality task τ_i has executed for $C_i(LO)$ without completing until the next idle instant. All other time intervals equate to normal mode. Schedulability analysis of mixed criticality schemes provides the necessary guarantees that these requirements are met.

In this paper, we use the term *degraded* mode to describe an interval of time during which a mixed criticality scheduling scheme does not release new jobs of LO-criticality tasks. To comply with the above timing requirements, degraded mode must be contained within abnormal mode (defined above); however, there need not be a one to one correspondence between the two. Entry to and exit from degraded mode takes place under the control of the RTOS and represents a well-defined runtime behavior of the system. With the original runtime protocol for AMC [4] degraded mode is entered when a HI-criticality task executes for its LO-criticality execution time estimate $C_i(LO)$ without signalling completion. By contrast, with SMC [12] the concept of abnormal mode remains, but there is no degraded mode as such, since releases of LO-criticality tasks are never abandoned. Finally, with the modified runtime protocol for AMC proposed in this paper, degraded mode is entered when a job of a HI-criticality task τ_i has not signalled completion by a time equal to its LO-criticality response time $R_i(LO)$, as measured from the start of the priority level-i busy period during which it executes.

The concept of a *priority level-i busy period* [44] is defined as follows:

(i) It starts at a time $s[i]$ when a job of a task of priority i or higher (i.e. in $hep(i)$) is released and there are no jobs of tasks in $hep(i)$ with execution pending that were released before time $s[i]$.

(ii) It is a contiguous interval of time during which jobs of tasks in $hep(i)$ execute.

(iii) It ends at the earliest time $t[i]$ after $s[i]$ when there are no jobs of tasks in $hep(i)$ that have execution pending that were released strictly before $t[i]$.

In mathematical terms, busy periods can be viewed as right half open intervals $[s[i], t[i])$. Thus the end of one priority level-i busy period may correspond to the start of the next priority level-i busy period; the two are however distinct and are not amalgamated.

IV. Adaptive Mixed Criticality Schemes

In this section, we recap on the Adaptive Mixed Criticality (AMC) scheme [4] and show how the runtime protocol can be modified to improve the service provided to LO-criticality tasks by delaying the transition to degraded mode and expediting the return from it.

A. Schedulability Analysis of AMC

The Adaptive Mixed Criticality (AMC) scheme [4] is based on fixed priority preemptive scheduling. Under the original runtime protocol for AMC, the system enters degraded mode when a HI-criticality task τ_i executes for its LO-criticality execution time budget $C_i(LO)$ without signaling completion, and returns from degraded mode when an idle instant occurs[2]. During degraded mode, new releases of LO-criticality tasks are abandoned.

In normal mode, when all tasks comply with their LO-criticality execution time budgets, then all tasks must be schedulable. Hence schedulability can be determined using standard response time analysis for fixed priority preemptive scheduling [1], [2], evaluated via fixed point iteration:

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

In abnormal mode, only HI-criticality tasks are required to meet their deadlines. The worst-case response time $R_i(HI)$ for a HI-criticality task τ_i, accounting for the transition to abnormal mode, is given as follows [4]:

$$R_i(HI) \;=\; C_i(HI) \;+\; \sum_{j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \;+\; \sum_{k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (2)$$

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

The analysis embodied in (1) and (2) is referred to as the AMC-rtb test [4], where rtb stands for *response time bound*.

[2]An idle instant occurs at time t when there are no jobs released prior to that time that have execution remaining.

B. Applying Analysis-Runtime Co-design

We now apply the technique of *analysis-runtime co-design* to modify the runtime protocol for the AMC scheme to match the worst-case behavior that is accounted for by the above analysis. The original runtime protocol for the AMC scheme enters degraded mode when a job of a HI-criticality task τ_i has executed for $C_i(LO)$ without completing. By contrast, the analysis (given by (2)) assumes that jobs of LO-criticality tasks continue to be released up to a time $R_i(LO)$ from the start of the priority level-i busy period in which the job of task τ_i was released. We therefore modify the runtime protocol to reflect the behavior assumed by the analysis. The modified runtime protocol for AMC is specified as follows:

S1: The system starts in normal mode. In normal mode both LO- and HI-criticality tasks release jobs for execution.

S2: The system enters degraded mode when an active, i.e. released but unfinished, job of some HI-criticality task τ_i reaches a time $R_i(LO)$ after the start of the priority level-i busy period in which it was released. Jobs of LO-criticality tasks released in degraded mode are dropped, i.e. not executed.

S3: The system exits degraded mode, i.e. returns to normal mode, when a job of some HI-criticality task τ_j completes and there is no active job of another HI-criticality task τ_k that has reached a time $R_k(LO)$ after the start of the priority level-k busy period in which it was released.

S4: At all times, fixed priority preemptive scheduling is used to determine which of the active jobs to run.

We refer to the variant of the AMC scheme using the modified runtime protocol as **AMC-RH**, since it relies on monitoring the response times of HI-criticality tasks relative to the start of the busy period in which they were released.

We now prove that the AMC-rtb analysis given by (1) and (2) is sufficient to ensure schedulability under AMC-RH for both LO- and HI-criticality tasks according to requirements R1 and R2, given in Section III.

First, we note two properties of the analysis that apply to schedulable systems of constrained-deadline mixed criticality tasks, i.e. when $R_i(LO) \leq R_i(HI) \leq D_i \leq T_i$, and follow directly from standard response time analysis for fixed priority preemptive scheduling [1], [2].

Property P1: $R_i(LO)$, given by (1), corresponds to the longest priority level-i busy period within which a job of a HI-criticality task τ_i can execute, assuming that it and all higher priority tasks execute for no more than their LO-criticality execution times.

Property P2: $R_i(HI)$, given by (2), corresponds to the longest priority level-i busy period within which a job of a HI-criticality task τ_i can execute, assuming that: (i) it and all higher priority HI-criticality tasks execute for no more than their HI-criticality execution times, and (ii) all higher priority LO-criticality tasks release jobs only during the first part of the priority level-i busy period, up to $R_i(LO)$ from when that busy period started.

Theorem 1. *The AMC-rtb analysis given by (1) and (2) is a sufficient schedulability test for AMC-RH, ensuring that requirements R1 and R2 are met.*

Proof. We separate the proof into two cases corresponding to requirements R1 and R2.

Case 1: Considers intervals where the processor is busy executing tasks and no job of any task exceeds its LO-criticality WCET estimate. In this case, requirement R1 applies and jobs of both LO- and HI-criticality tasks must be guaranteed to meet their deadlines. Since no job exceeds its LO-criticality WCET estimate, then (1) bounds the LO-criticality response time for each task under AMC-RH. Further, it follows from Property P1 and the runtime protocol employed by AMC-RH that degraded mode cannot be entered in this case, and so no LO-criticality jobs are dropped.

Case 2: Considers intervals where the processor is busy executing tasks and one or more jobs of one or more HI-criticality tasks execute for their LO-criticality WCET estimates without signaling completion. In this case, requirement R2 applies and only jobs of HI-criticality tasks need be guaranteed to meet their deadlines. In the context of this case, consider an arbitrary priority level-i busy period in which a single[3] job J of a HI-criticality task τ_i executes. Note, the end of this priority level-i busy period corresponds to the completion of job J. Since fixed priority preemptive scheduling is employed, there can be no interference on job J due to jobs of lower priority tasks. Further, by definition of a priority level-i busy period, there can be no interference from any jobs of higher priority tasks that were released prior to the start of the busy period. There are two sub-cases to consider.

Case 2a: The length of the priority level-i busy period is no greater than $R_i(LO)$, in which case $R_i(HI)$ given by (2) trivially upper bounds the response time of job J, since $R_i(HI) \geq R_i(LO)$.

Case 2b: The length of the priority level-i busy period is greater than $R_i(LO)$. Given the runtime protocol employed by AMC-RH, the system must necessarily be in degraded mode from time $R_i(LO)$ after the start of the busy period until job J completes. From Property P2, it follows that $R_i(HI)$ given by (2) upper bounds the response time of job J □

Note that in *Case 2b* above, there could be other intervals of degraded mode prior to $R_i(LO)$ as a consequence of the behavior of the modified runtime protocol with respect to jobs of other HI-criticality tasks; however, these additional degraded mode intervals could only serve to decrease the interference on job J by preventing the release of jobs of higher priority LO-criticality tasks.

The AMC-rtb analysis given by (1) and (2) is *less precise* for the original runtime protocol for AMC. This imprecision occurs because releases of jobs with HI-criticality behavior are not possible prior to a time $R_i(LO)$, as measured from the start of the busy period, without also potentially substantially restricting the number of LO-criticality jobs that can be released and therefore contribute interference. The analysis

[3]Since deadlines are constrained, then for a schedulable system there can be at most one job of task τ_i in each priority level-i busy period.

is *more precise* for the modified runtime protocol of AMC-RH, since releases of LO-criticality jobs are still permitted up to $R_i(LO)$ even when every job of a HI-criticality task in the priority level-i busy period executes for its HI-criticality execution time. Nevertheless, the analysis is still not exact for AMC-RH, as shown by an example in Appendix A that also serves to illustrate the differences in precision.

The original runtime protocol for AMC [4] assumes that degraded mode is entered when a HI-criticality task τ_i executes for $C_i(LO)$ without signalling completion, and is exited once an idle instant occurs. However, considering the schedulability analysis embodied in (2), it is easy to see that these criteria are pessimistic. By comparison, the modified runtime protocol for AMC-RH delays entry into degraded mode for as long as permitted by the offline analysis and similarly exits degraded mode as early as permitted.

Procrastinating until $R_i(LO)$ from the start of the busy period before switching to degraded mode means that the transition occurs as late as it possibly could do when $C_i(LO)$ is used as a trigger. Procrastinating enables overruns by some HI-criticality jobs to be mitigated by underruns of other jobs meaning that a mode change may not be necessary at all. Further, for periodic tasks, when the pattern of job releases does not follow the worst-case (i.e. the initial synchronous arrival sequence) then any dynamic slack available is automatically captured by procrastinating before making the mode change. Similarly, slack created by jobs of sporadic tasks arriving at less that their maximum rate is also automatically captured. Reducing the number of times that degraded mode is entered and the time spent in that mode has significant advantages in terms of reducing the number of LO-criticality jobs that miss their deadlines or do not execute because they are abandoned.

Conversely, monitoring response times instead of execution times to initiate the transition to degraded mode has a disadvantage in terms of more precise schedulability tests. There are some systems that would be schedulable under AMC according to exact analysis of the original runtime protocol that are not schedulable according to exact analysis of the modified protocol, see Appendix A for an example.

A further variant of the AMC scheme, explored in the evaluation in Section V, is referred to as **AMC-RA**. Similar to AMC-RH, AMC-RA is defined by specifications S1, S2, and S4; however, specification S3 is replaced by S5 below.

S5: Under AMC-RA, the system exits degraded mode when a job of some task τ_j completes and there are no other active jobs of any task, i.e. there is an idle instant.

Theorem 2. *The AMC-rtb analysis given by (1) and (2) is a sufficient schedulability test for AMC-RA, ensuring that requirements R1 and R2 are met.*

Proof. Proof is identical to that for AMC-RH given in Theorem 1, with the word "AMC-RH" replaced by "AMC-RA" □

Appendix B provides a brief summary of how the RTOS can manage entry to and exit from degraded mode, implementing the modified runtime protocol of AMC-RH and AMC-RA.

C. Increasing Execution Time Budgets

The performance of AMC-RH and AMC-RA can be further improved by using sensitivity analysis [34] to make use of offline slack [45] to increase the LO-criticality execution time budgets of HI-criticality tasks as far as possible while still retaining a schedulable system [30], [35].

The specific method used here is as proposed by Bate et al. in [39], [40]. First, the execution time budgets of *all* HI-criticality tasks are increased as much as possible while ensuring that the system remains schedulable according to the AMC analysis (i.e. (1) and (2)). This is achieved by forming a binary search for the largest value of α such that the system remains schedulable when the $C_i(LO)$ value for each HI-criticality task τ_i is replaced by $C_i(BU) = \min(C_i(HI), \alpha C_i(LO))$. Note, we use $C_i(BU)$ rather than $C_i(LO)$ to emphasize that these are no longer the LO-criticality execution time estimates associated with those HI-criticality tasks, but rather *execution time budgets* that will be used to determine larger $R_i(BU)$ values that replace the $R_i(LO)$ values, and are used to trigger the transition to degraded mode at runtime. The initial lower value of α used for the binary search is 1, since the system is assumed to be schedulable under AMC to begin with, and the initial upper value is given by the largest ratio $C_i(HI)/C_i(LO)$ for any HI-criticality task τ_i. At each step of the binary search, Audsley's Optimal Priority Assignment algorithm [46] is used along with the single task schedulability test (i.e. (1) and (2)) to determine if the system is schedulable for that value of α. By reapplying Audsley's algorithm on each step, this ensures that the final assignment also has the most robust priority ordering [47]. Second, a similar process is used to further increase, if possible, the $C_i(BU)$ value for each individual task in turn, since after the first step, some but not all of the $C_i(BU)$ values may still be increased without making the system unschedulable. (This is done for all HI-criticality tasks in order of increasing deadlines).

We refer to the more sophisticated schemes that use increased execution time budgets as **AMC-RHS** and **AMC-RAS** respectively. The behavior of these schemes is effectively the same as AMC-RH and AMC-RA; however, all occurrences of $C_i(LO)$ for HI-criticality tasks are replaced by the larger $C_i(BU)$ values, leading to larger $R_i(BU)$ values that replace the $R_i(LO)$ values and are monitored in order to trigger transitions to degraded mode. For systems that are schedulable under classical fixed priority preemptive scheduling (i.e. assuming that all jobs may take an execution time that corresponds to their own criticality level i.e. $C_i(HI)$ for HI-criticality tasks, and $C_i(LO)$ for LO-criticality tasks), AMC-RHS and AMC-RAS have the useful property that no LO-criticality jobs miss their deadlines. This is the case, because for such systems the first step described above results in $C_i(BU) = C_i(HI)$ for all HI-criticality tasks. In practice, some of the statically available slack in the system could also be used to provide LO-criticality tasks with additional headroom for longer than expected execution, i.e. execution budgets larger than $C_i(LO)$. This is not explored further in this paper.

D. Lazy execution of Jobs

The performance of AMC-RHS and AMC-RAS can be further improved by running LO-criticality jobs that would otherwise be abandoned in degraded mode at background priorities to give them a *last chance* to complete by their deadlines. This approach was proposed by Iacovelli and Kirner [41] as a way of augmenting the Bailout Protocol [39], [40], but can be applied to any scheme based on fixed priorities that abandons jobs in degraded mode.

The basic idea is one of *lazy* execution. A separate background run-queue is used for all jobs of LO-criticality tasks that would otherwise be abandoned in degraded mode. Instead of being abandoned, these jobs are added to the background run-queue and executed according to fixed priority preemptive scheduling whenever the normal run-queue is empty, i.e. when the processor would otherwise be idle. If a job from the background run-queue reaches its deadline before it is completed, then the job is discarded. Since we consider only constrained-deadline tasks, only one job of each task can be active at any given time, and hence jobs of the same task execute in strict order of arrival regardless of whether they are placed in the background run-queue or the normal run-queue. We refer to the schemes that make use of both lazy execution and increased execution time budgets as **AMC-RHSL** and **AMC-RASL** respectively.

V. Scenario-Based Evaluation

In this section, we present a simulation and hence scenario-based evaluation of the performance of the variants of the AMC scheme introduced in this paper and compare them to prior work in this area, specifically to variants of the Bailout Protocol [39], [40] and the original AMC scheme [4], using an experimental framework with configurations and metrics similar to those that have previously been used to evaluate the Bailout Protocol [39], [40] and Lazy Execution [41].

Scenario-based evaluation [48]–[50] complements the evaluation of schedulability analysis as the latter only illustrates under what conditions timing guarantees are met. Rather, we are interested in how well the different schemes perform in terms of minimizing the number of jobs of LO-criticality tasks that are abandoned without executing or miss their deadlines.

A. Evaluation Metrics

The following metrics were used in the evaluation.

(i) *Number of HI-criticality Deadline Misses (HDM)*: Such deadline misses should not be experienced with any of the schemes. This metric is used to check that is the case.

(ii) *Jobs Not Executed (JNE)*: The number of LO-criticality jobs that were abandoned in degraded mode.

(iii) *LO-criticality Deadline Misses (LDM)*: The number of LO-criticality jobs that were executed, but missed their deadlines.

(iv) *Time in Degraded mode (TiD)* - The amount of time spent in degraded mode. (For the variants of the Bailout Protocol that we compare against, degraded mode equates to the bailout and recovery modes).

(v) *Number of times in Degraded mode (NiD)* - How many times the system entered degraded mode.

Aside from the HDM metric, which should always be zero, the most important metric is the sum of JNE and LDM, which represents the total number of LO-criticality jobs that are not completed by their deadlines, including those that are abandoned upon release.

B. Scheduling Schemes

The following mixed criticality scheduling schemes were compared.

1. **AMC-RA** – The modified AMC scheme, as described in Section IV, with transitions to degraded mode when any HI-criticality task τ_i reaches, without completing, its LO-criticality response time $R_i(LO)$ since the start of the busy period in which it was released. Degraded mode is exited on an idle instant.
2. **AMC-RAS** – AMC-RA enhanced by off-line increases in execution time budgets, see Section IV-C.
3. **AMC-RASL** – AMC-RAS enhanced via lazy execution [41] of jobs that would otherwise be abandoned, see Section IV-D.
4. **AMC-RH** – The modified AMC scheme, as described in Section IV, with transitions to degraded mode when any HI-criticality task τ_j reaches, without completing, its LO-criticality response time $R_j(LO)$ since the start of the busy period in which it was released. Degraded mode is exited once there is no such active HI-criticality task in the run queue.
5. **AMC-RHS** – AMC-RH enhanced by off-line increases in execution time budgets, see Section IV-C.
6. **AMC-RHSL** – AMC-RHS enhanced via lazy execution [41] of jobs that would otherwise be abandoned, see Section IV-D.
7. **AMC+** – The original AMC scheme [4], with return to normal mode on an idle instant.
8. **AMC+S** – The AMC+ scheme, enhanced by off-line increases in execution time budgets, see Section IV-C.
9. **AMC+SG** – The AMC+ scheme enhanced by both off-line increases in execution time budgets, and runtime reclamation of gain time, see Section 5.2 of [40].
10. **AMC+SGL** – The AMC+SG scheme enhanced via lazy execution [41] of jobs that would otherwise be abandoned, see Section IV-D.
11. **BP** – The basic Bailout Protocol, see Section 4 of [40].
12. **BPS** – The Bailout Protocol enhanced by off-line increases in execution time budgets, see Section 5.1 of [40].
13. **BPSG** – The Bailout Protocol enhanced by both off-line increases execution time budgets, and runtime reclamation of gain time, see Section 5.2 of [40].
14. **BPSGL** – BPSG enhanced via lazy execution [41] of jobs that would otherwise be abandoned, see Section IV-D.

C. Task Set Generation

Task set generation was performed as follows:

- *Task Set Cardinality* - The number of tasks 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 tasks assigned LO-criticality.
- *Task Utilizations* - The Dirichlet-Rescale (DRS) algorithm [51] open source Python software [52] was used to provide an unbiased distribution of task utilization values that summed to the target utilization required, subject to a set of individual constraints. First, HI-criticality utilization values $U_i(HI)$ were generated for the $n(HI)$ HI-criticality tasks, such that the total HI-criticality utilization of those tasks summed to $U(HI) = CP \cdot CF \cdot U$, where CF is the Criticality Factor (default $CF = 2.0$) characterizing the multiplier between HI- and LO-criticality utilization, and U is the target utilization required (default $U = 0.8$). Second, LO-criticality utilization values $U_i(LO)$ were generated for all of the tasks, such that the total LO-criticality utilization of all tasks summed to $U(LO) = U$. For LO-criticality tasks, $U_i(LO)$ was constrained to be in the range $[0.0, 1.0]$, while for HI-criticality tasks, $U_i(LO)$ was constrained to be in the range $[0.0, U_i(HI)]$, hence ensuring that $U_i(LO) \leq U_i(HI)$.
- *Periods and Deadlines* - The period of each task was chosen in one of two ways. *Semi-harmonic periods* were chosen at random from a set of harmonics of two base frequencies (i.e. 25, 50, 100, 250, 500, 1000 and 20, 40, 80, 200, 400, 800ms) as typically found in automotive and avionics systems [53]. *Non-harmonic periods* were chosen at random according to a log-uniform distribution [54], from a range 10ms to 1 second (rounded to 0.1ms). Task deadlines were set equal to their periods.
- *Execution Times* - The LO-criticality execution times of all tasks were given by $C_i(LO) = U_i(LO) \cdot T_i$, and the HI-criticality execution times of HI-criticality tasks by $C_i(HI) = U_i(HI) \cdot T_i$. Finally, Best-Case Execution Times (BCET) were chosen at random between 80% and 100% of $C_i(LO)$. (This small variation is representative of code from Safety Critical Systems).
- *Failure Probability* (FP) - At runtime, jobs of HI-criticality tasks had a probability of FP (default $FP = 10^{-4} = 0.01\%$) of exceeding their $C_i(LO)$ execution time.

In all of the experiments, we required that the task sets chosen had at least one task that was unschedulable according to exact analysis of fixed priority preemptive scheduling [2] (i.e. ignoring criticality), but were nevertheless schedulable according to the AMC-rtb test [4].

D. Simulation

The experiments covered 500 task sets for each of the configurations considered. For each scheduling scheme, we simulated the runtime behavior of each task set, starting with a different random seed. The same random seeds were used for each of the schemes to ensure a precise like-for-like comparison. The duration of each simulation run was 10^{13} time units (of 0.1ms), hence this was sufficient for 10^6 jobs of the task with the longest period.

In the simulation on each release, an actual execution time was chosen for the job as follows. For jobs of LO-criticality tasks, the value was chosen at random from a uniform distribution in the range $[BCET, C_i(LO)]$. For jobs of HI-criticality tasks, a random boolean variable with a probability of FP (default 10^{-4}) of returning $true$ was used to determine if the job would exhibit HI-criticality behavior. If so, then its execution time was chosen at random from a uniform distribution in the range $[C_i(LO), C_i(HI)]$, otherwise the range was $[BCET, C_i(LO)]$. The probability FP used to determine if HI-criticality behavior would be exhibited was deliberately set to a relatively high value by default to stress the system behavior. In practice such a high value is perhaps unlikely, but possible, for example if the testing used to determine LO-criticality execution time estimates did not actually reveal the worst-case path.

For the schemes making use of statically available slack to increase execution time budgets, the $C_i(BU)$ parameters were computed via off-line sensitivity analysis, as described in Section IV-C, before running the simulator. The $C_i(BU)$ and corresponding $R_i(BU)$ values were then used by the simulator to determine when the system should transition to degraded mode, with the $C_i(LO)$ values used in the selection of job execution times, as explained above. The simulation did not include scheduling overheads, while these would have some impact in practice, all of the schemes compared have low overheads similar to those incurred by execution time monitoring and budget accounting.

E. Evaluation Results

The evaluation results are shown using box and whisker plots. The box represents the range of values between quartiles (25 and 75 percentiles). The horizontal line in the middle of the box is the median (50 percentile). The two horizontal whiskers above and below the box show the 5 and 95 percentiles.

Four types of task sets were considered: (i) strictly periodic task sets with semi-harmonic periods, (ii) strictly periodic task sets with non-harmonic periods, (iii) sporadic task sets with semi-harmonic periods, and (iv) sporadic task sets with non-harmonic periods. In the case of sporadic task sets, all LO-criticality tasks were sporadic, while all HI criticality tasks remained strictly periodic. The schedulability analysis for sporadic tasks was exactly the same as for periodic tasks. At runtime, however, at each (periodic) arrival time for a sporadic task, the probability of releasing its job was 0.5. Thus the job release pattern mirrored that of strictly periodic behavior, with the exception that about 50% of the jobs of LO-criticality tasks were omitted.

The number of HI-criticality Deadline Misses(HDM) for all four types of task set and all 14 schemes was zero, hence these results are not shown in the graphs. The other results are organized by the type of task set, and then discussed according to the metric considered.

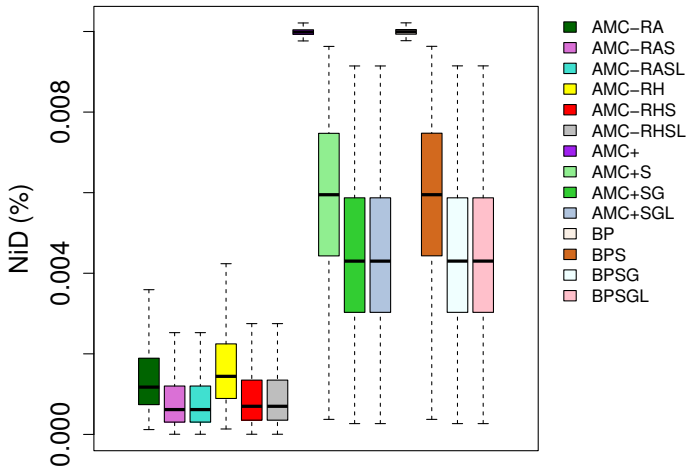

Fig. 1. Results for $NiD(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods.

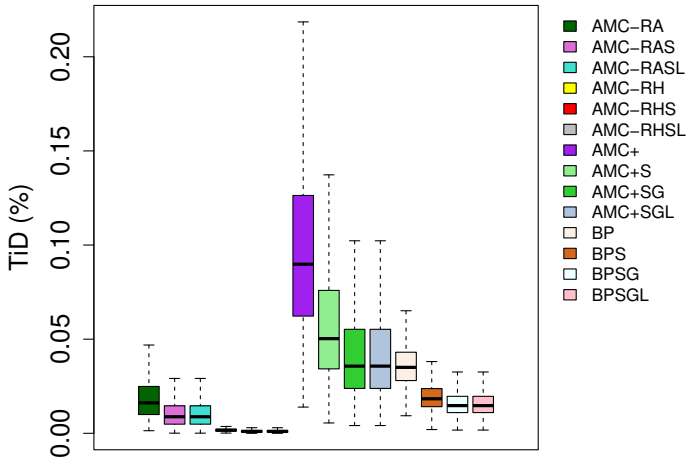

Fig. 2. Results for $TiD(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods.

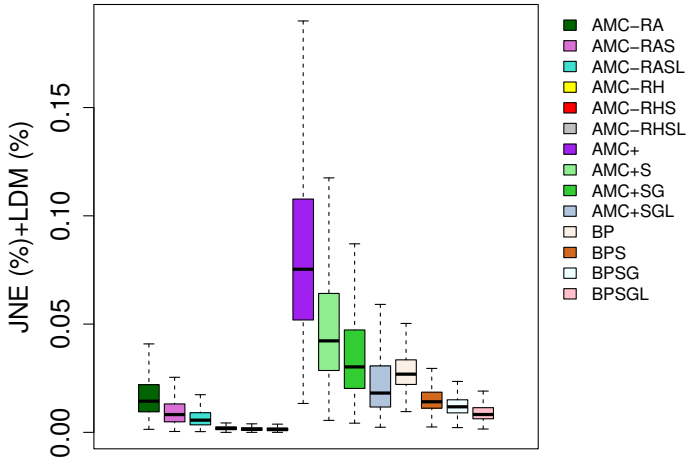

Fig. 3. Results for $JNE + LDM(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods.

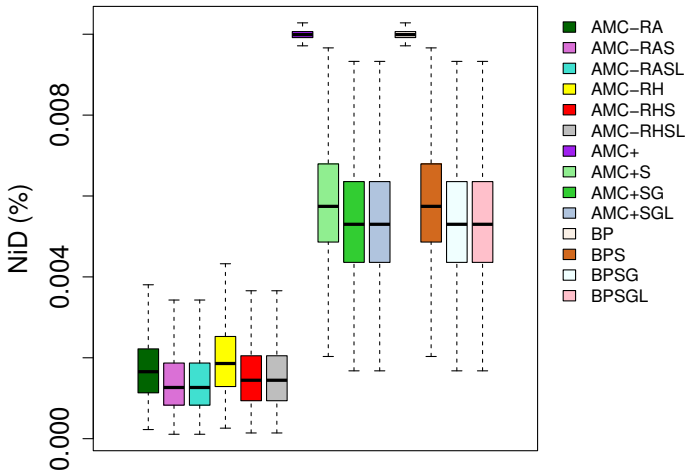

Fig. 4. Results for $NiD(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods.

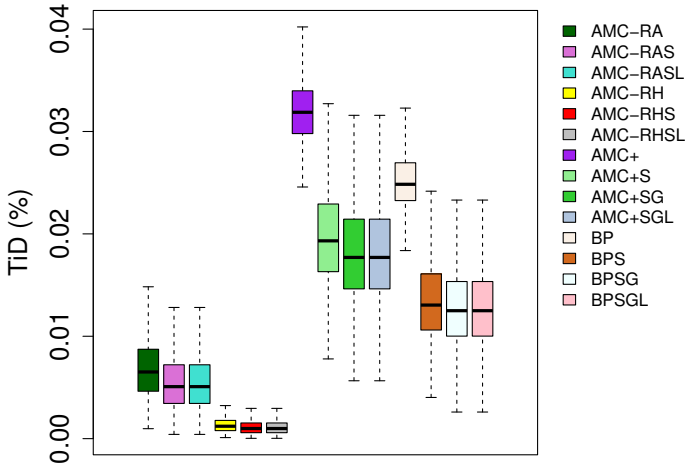

Fig. 5. Results for $TiD(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods.

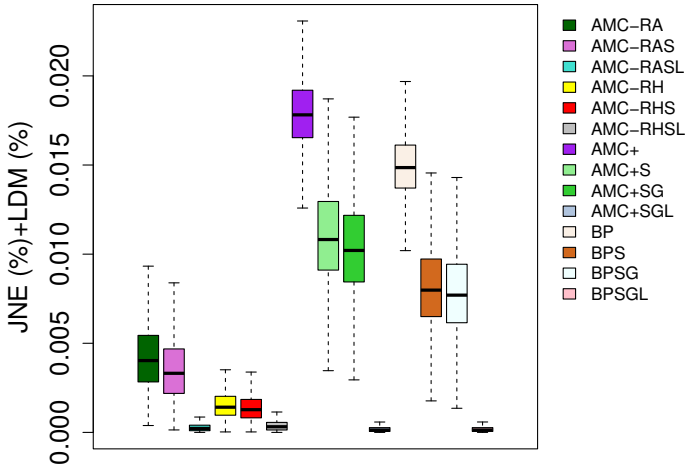

Fig. 6. Results for $JNE + LDM(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods.

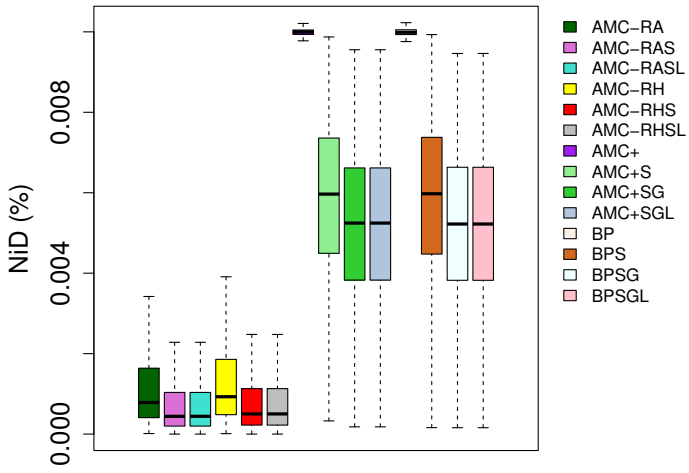

Fig. 7. Results for $NiD(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods and Sporadic.

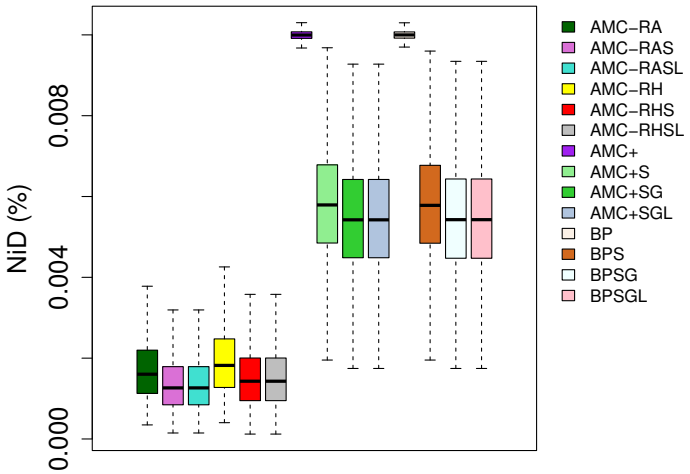

Fig. 10. Results for $NiD(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods and Sporadic.

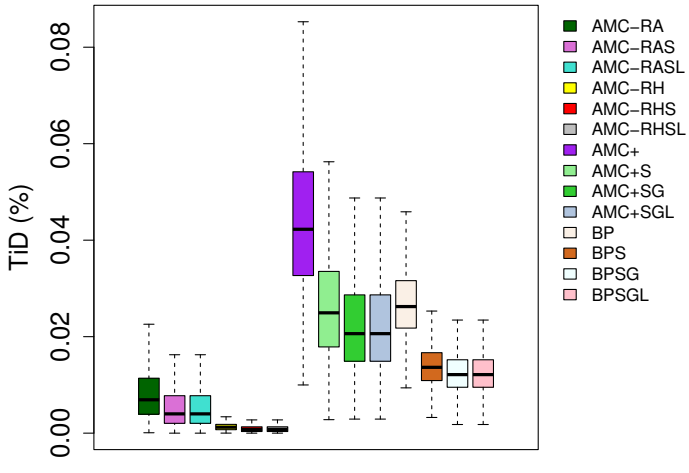

Fig. 8. Results for $TiD(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods and Sporadic.

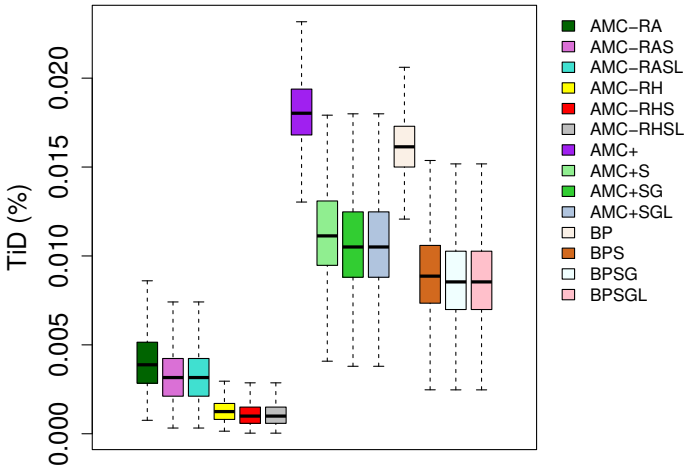

Fig. 11. Results for $TiD(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods and Sporadic.

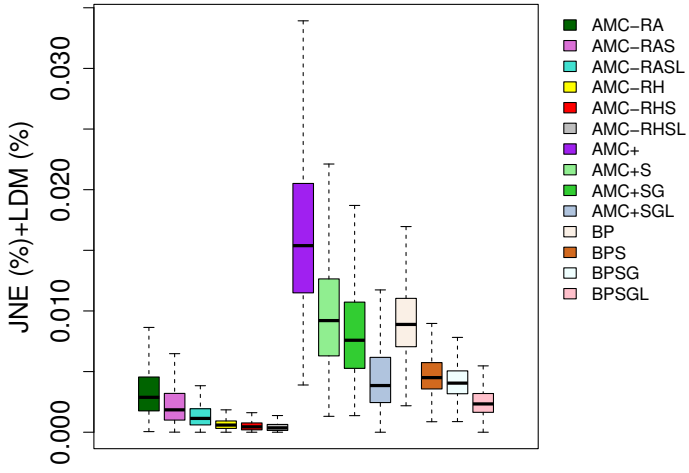

Fig. 9. Results for $JNE + LDM(\%)$ - 80% LO-criticality Utilization: Semi-harmonic Periods and Sporadic.

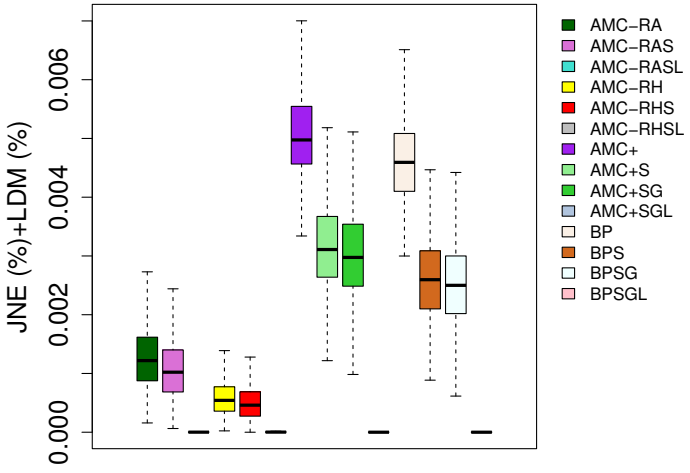

Fig. 12. Results for $JNE + LDM(\%)$ - 80% LO-criticality Utilization: Non-Harmonic Periods and Sporadic.

Figs. 1, 4, 7 and 10 illustrate the number of times ($NiD(\%)$) that degraded mode was entered as a percentage of the number of HI-criticality jobs (i.e. the maximum number of times that degraded mode could ever be entered), for the four types of task set. Note, the y-axis scale is identical on these four graphs.

$NiD(\%)$ has the same median value of 0.01 for AMC+ and BP for all task sets. These values simply reflect the configured Failure Probability FP (of a HI-criticality job executing for more that its LO-criticality execution time), by default $FP = 10^{-4} = 0.01\%$. Utilizing off-line slack to increase LO-criticality execution time budgets improves performance for the AMC+S and BPS schemes, with further improvements obtained by utilizing gain-time in AMC+SG and BPSG.

Observe that $NiD(\%)$ is almost exactly the same for each AMC+ scheme and the equivalent scheme based on the Bailout Protocol (i.e. for AMC+ and BP, for AMC+S and BPS, and for AMC+SG and BPSG). This is because the Bailout Policy only operates once degraded mode is entered, with the same criteria for transition to that mode as AMC+. Hence, the Bailout Policy does not act to reduce the number of times that the system enters degraded mode, only how long it stays in that mode.

The AMC-RA and AMC-RH schemes greatly reduce the number of times that degraded mode is entered compared to AMC+ and BP. This is because the AMC-RA and AMC-RH schemes wait until the LO-criticality response time of some HI-criticality task is reached from the start of the busy period in which it is released, rather than transitioning to degraded mode immediately a HI-criticality task reaches its LO-criticality execution time budget without completing. This advantage is further enhanced by utilizing off-line increases in execution time budgets to provide longer intervals (response times) before degraded mode is entered in the AMC-RAS and AMC-RHS schemes. The AMC-RH schemes typically enter degraded mode slightly more often than their AMC-RA counterparts. The reason being that the AMC-RA schemes wait for an idle instant to exit degraded mode. This can lead to what would otherwise be two separate intervals of degraded mode under the equivalent AMC-RH scheme being amalgamated into one interval of degraded mode under the AMC-RA scheme, hence lowering the number of times that degraded mode is entered, but increasing the time in that mode.

Figs. 2, 5, 8, and 11 illustrate the time spent in degraded mode ($TiD(\%)$) as a percentage of the total simulation time. Note, the different y-axis scale on these four graphs. With non-harmonic task sets, the workload is typically spread out more by the de-synchronized job releases, resulting in shorter busy periods and more frequent idle instants. This is the reason why the time in degraded mode for all schemes is much shorter for non-harmonic task sets than with semi-harmonic task sets. It also explains why the schemes based on the Bailout Protocol are able to substantially reduce the time in degraded mode compared to their AMC+ counterparts for semi-harmonic task sets, but not so much for non-harmonic task sets. In the former case, the bailout mechanism conveys an advantage, whereas in the latter case, both types of scheme often exit degraded mode via an idle instant.

Figs. 2, 5, 8, and 11 also show how the advantage that the AMC-RA and AMC-RH families of schemes have in entering degraded mode fewer times, combined in the case of AMC-RH with not having to wait for an idle instant to exit degraded mode, translates into a large reduction in the overall time spent in that mode compared to the BP and AMC+ schemes. Further, each AMC-RH scheme has a significant advantage over its AMC-RA counterpart due to its ability to exit degraded mode without having to wait for an idle instant.

Figs. 3, 6, 9 and 12 show the percentage of LO-criticality jobs that were not completed ($JNE(\%) + LDM(\%)$), either because they were not executed or because they failed to complete by their deadline. Note, the different y-axis scale on these four graphs. These figures show that the baseline AMC-RA and AMC-RH schemes are highly effective in reducing the percentage of LO-criticality jobs that are not completed compared to both the AMC+ scheme and the Bailout Protocol. Further, by utilizing off-line increases in execution time budgets to delay the transition to degraded mode, the AMC-RAS and AMC-RHS schemes outperform the existing AMC+S and BPS schemes that utilize off-line slack. As expected, the AMC-RH (resp. AMC-RHS) scheme outperforms the AMC-RA (resp. AMC-RAS) scheme, since given the exact same scenario of job releases, it transitions out of degraded mode no later.

Finally, observe that for all schemes, lazy execution is highly effective at reducing the percentage of LO-criticality jobs that are not completed, but has no bearing on the number of times that degraded mode is entered or on the time spent in that mode. Lazy execution is less effective for semi-harmonic task sets due to the longer busy periods and fewer idle intervals compared to non-harmonic task sets.

Overall, the results shown in Figs. 1 to 12 provide evidence that the modified runtime protocol for the AMC scheme introduced in this paper is highly effective in reducing: (i) the number of times that degraded mode is entered, (ii) the amount of time that is spent in degraded mode, and hence (iii) the number of LO-criticality jobs that are unable to complete execution by their deadlines. In particular, the AMC-RH schemes outperform the previously published AMC+ and BP schemes, providing substantially lower median values on all three performance metrics.

VI. Conclusions

In mixed criticality systems, schemes such as Adaptive Mixed Criticality (AMC) [4] employ a degraded mode where new releases of LO-criticality jobs are abandoned to ensure that HI-criticality tasks remain schedulable when some of the jobs of those tasks do not conform to their normal behavior and hence exceed low assurance bounds on their execution times (i.e. LO-criticality execution time estimates). The original runtime protocol for the AMC scheme transitions to degraded mode whenever a job of a HI-criticality task τ_i exceeds its $C_i(LO)$ budget. However, if the rest of the system is not exhibiting worst-case behavior in terms of task execution times and the phasing of job releases, then such a transition may be

unnecessarily early leading to many more dropped jobs of LO-criticality tasks than are in fact necessary to maintain the high levels of assurance needed to provide HI-criticality tasks with robust timing guarantees.

Via a consideration of the AMC-rtb schedulability test [4], we employed *analysis-runtime co-design* techniques to modify the trigger conditions for entering and exiting degraded mode to correspond to the worst case that is accounted for by the analysis. Thus we modified the runtime protocol to enter and exit degraded mode based on whether or not there is an active (i.e. unfinished) job of some HI-criticality task τ_i that is at least $R_i(LO)$ (it's LO-criticality response time) from the start of the priority level-i busy period during which it was released.

By design, the new scheme, referred to as AMC-RH, can be analysed more precisely using the AMC-rtb schedulability test. Further, we showed that the new scheme can also benefit from increasing the LO-criticality execution time budgets of HI-criticality tasks until the system is just schedulable. Thus increasing their LO-criticality response times and hence further delaying the onset of degraded mode and expediting return from it. This variant of the scheme is referred to as AMC-RHS, since it takes advantage of static slack.

A systematic scenario-based evaluation compared the new scheme, AMC-RH, to the original AMC scheme and also to the Bailout Protocol [39], [40]. The schemes were judged on the basis of criteria characterizing the degradation in the level of service provided to LO-criticality tasks. The new scheme provided significant advantages in terms of a reduction in: (i) the number of times degraded mode is entered (NiD), (ii) the amount of time spent in degraded mode (TiD), and (iii) the number of LO-criticality jobs that are either not executed or miss their deadlines (JNE+LDM). Table I shows the values of these metrics for the Bailout Protocol and the AMC-RH scheme as a percentage of the values for the original AMC scheme. Note, these summary results are based on a comparison of the mean values obtained in experiments on 500 task sets, see Section V-E, Figures 1–6 for further details. In the case of task sets with semi-harmonic periods, representative of those found in avionics and automotive systems, the most important metric (JNE+LDM) was reduced by a factor of 40 using AMC-RH, compared to a reduction by a factor of 3 using the Bailout Protocol.

	Semi-harmonic periods			Non-harmonic periods		
	NiD	TiD	JNE+LDM	NiD	TiD	JNE+LDM
BP	100%	36.9%	34.8%	100%	78.4%	83.4%
AMC-RH	16.8%	1.7%	**2.5%**	19.9%	4.1%	**8.7%**

TABLE I
PERFORMANCE COMPARISON WITH RESPECT TO THE ORIGINAL AMC+ SCHEME. (NOTE, SMALLER VALUES IMPLY IMPROVED PERFORMANCE).

Finally, we note that the performance of all of the schemes can be improved by using an extra background priority queue to permit LO-criticality jobs that would otherwise have been dropped in degraded mode to be run in what would otherwise have been idle time, providing a last chance to meet their deadlines. However, this use of *lazy execution* [41], with jobs of the same task executing at different priorities, impacts mutual exclusion protocols [55] and increases blocking effects, which may hamper its practical application.

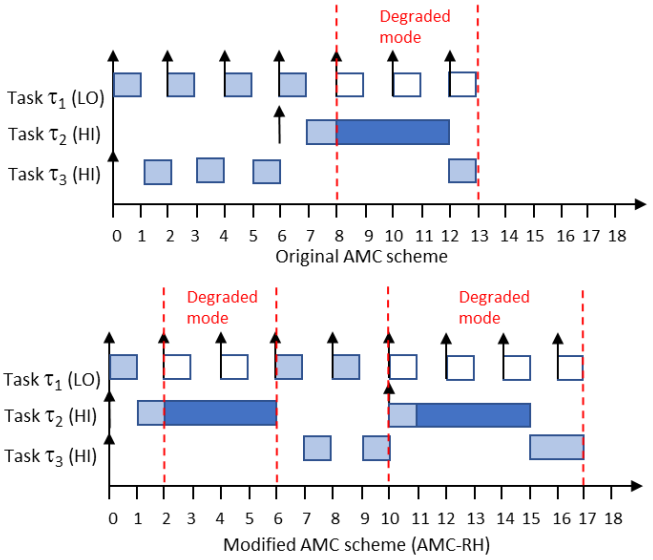

Fig. 13. Schedules for original and modified AMC schemes

APPENDIX A: SCHEDULABILITY

In this appendix, we show that the analysis provided by the AMC-rtb schedulability test (1) and (2) is not exact for AMC-RH, but is less pessimistic for AMC-RH than it is for the original AMC scheme. This can be seen by considering an example task set as follows: $\tau_1 = \{C_1(LO) = 1, T_1 = 2, D_1 = 2, L_1 = LO\}$, $\tau_2 = \{C_2(LO) = 1, C_2(HI) = 5, T_2 = 10, D_2 = 10, L_2 = HI\}$, and $\tau_3 = \{C_3(LO) = C_3(HI) = 4, T_3 = 100, D_3 = 18, L_3 = HI\}$. Note, that under fixed priority preemptive scheduling, the only viable priority ordering is for task τ_1 to have the highest priority and task τ_3 the lowest. Further, it is easy to see that with this priority ordering, tasks τ_1 and τ_2 are schedulable, hence we focus on the schedulability of the lowest priority task τ_3.

From (1), we have $R_3(LO) = 10$, including interference from 5 jobs of task τ_1 (at 1 time unit each) and 1 job of task τ_2 (at 1 time unit). Further, from (2) we have $R_3(HI) = 19$, including interference from the same 5 jobs of task τ_1 released within $R_3(LO)$ (at 1 time unit each) and 2 jobs of task τ_2 (at 5 time units each). The test therefore concludes that task τ_3 is unschedulable; however, this is not in fact the case.

Figure 13 shows the schedules that result in the worst-case response time for a job of task τ_3 under the original and modified AMC schemes. In the diagram, job execution within a task's LO-criticality WCET estimate $C_i(LO)$ is shown in light blue, while execution after that estimate is exceeded is shown in dark blue. Jobs of LO-criticality tasks that are dropped in degraded mode are shown as an empty box.

Under the original AMC scheme, degraded mode is entered as soon as any job of task τ_2 executes for $C_2(LO) = 1$ without completing (i.e. exhibits HI-criticality behavior). It can be verified, by considering the different possible offsets of task

τ_2 from the initial release of tasks τ_1 and τ_3 at $t = 0$, that the worst-case interference on task τ_3 occurs when task τ_2 is released at $t = 6$. By $t = 6$, three jobs of τ_1 have executed and τ_3 has executed for 3 time units. A further job of τ_1 is also released at $t = 6$ and executes, followed by task τ_2. At $t = 8$, τ_2 has executed for $C_2(LO) = 1$ without completing, and so degraded mode is entered, and hence further releases of the LO-criticality task τ_1 are not permitted. Task τ_2 executes for a further 4 time units, followed by τ_3, which completes its final time unit of execution for a worst-case HI-criticality response time of 13. (Note, under the AMC scheme, the parameters of this example are such that it is impossible for task τ_3 to be subject to interference from more than one job of τ_2).

Under AMC-RH, each time task τ_2 exceeds its LO-criticality response time $R_2(LO) = 2$, degraded mode is entered and further jobs of LO-criticality task τ_1 are no longer released until the job of τ_2 completes, which happens by its HI-criticality response time $R_2(HI) = 6$. As a consequence, each time τ_2 exhibits HI-criticality behavior and executes for $C_2(HI) = 5$ within $R_3(LO)$, two jobs of task τ_1 are *necessarily* skipped. This reduces the total interference on task τ_3 within $R_3(LO)$ to effectively three jobs of task τ_1 (at 1 time unit each) and one job of task τ_2 (at 5 time units), or other combinations that entail no more interference. Hence by $R_3(LO) = 10$, task τ_3 suffers at most interference of 8 time units and hence executes for 2 time units leaving 2 time units of execution to complete. With a further 5 time units of interference from a final job of task τ_2, this equates to a worst-case HI-criticality response time of 17 for τ_3, which is therefore schedulable under AMC-RH.

Observe that for this example, the AMC-rtb analysis provided by (1) and (2) is considerably more pessimistic for the original AMC scheme than it is for AMC-RH. If the deadline on task τ_3 were 15, then the task set would be schedulable under the original AMC scheme, but not under AMC-RH.

In the following we show that the exact value of $R_i(HI)$ of each HI-criticality task τ_i under AMC-RH upper bounds that for the original AMC scheme.

For any scenario (i.e. pattern of task releases) and priority level-i busy period culminating in the completion of a HI-criticality task τ_i in its worst-case HI-criticality response time under the original AMC scheme, all of the interfering releases of higher priority LO-criticality tasks would also occur under AMC-RH (and possibly other releases as well). This is the case because under AMC-RH, it is not possible (by definition of the $R_j(LO)$ values used in the runtime protocol) for degraded mode to be entered until at least one job of some HI-criticality task τ_j has executed for its $C_j(LO)$ without completing, which is the rule that the original AMC scheme uses for entry into degraded mode. Hence, AMC-RH cannot enter degraded mode before the original AMC scheme does. Since the original AMC scheme does not exit degraded mode until an idle instant, i.e. at or after task τ_i completes, it follows that the exact value of $R_i(HI)$ of each HI-criticality task τ_i under the original AMC scheme lower bounds that under AMC-RH. Since the LO-criticality behavior is the same in both cases, it follows that exact schedulability under the original AMC scheme *dominates* that under AMC-RH (dominance rather than equivalence is assured by the previous example showing that there exists at least one case where exact schedulability is worse under AMC-RH).

Finally, the AMC-max test [4] can be used to obtain more accurate but still inexact schedulability results for the original AMC scheme. The gains over AMC-rtb in terms of additional schedulable task sets are relatively small [4] and hence the majority of subsequent work, including the recent adoption of AMC by industry [7], [8], has been underpinned by the simpler AMC-rtb test. The AMC-max test is not compatible with the modified AMC runtime protocol, due to the way in which the test is formulated. The AMC-max test considers jobs of each HI-criticality task τ_j released prior to the mode change time at time s as contributing $C_j(LO)$, which is not necessarily the case with the modified protocol. The modified AMC runtime protocol makes a trade-off in schedulability versus an improved level of service for LO-criticality tasks, which is completely hidden when the AMC-rtb test is used.

APPENDIX B: IMPLEMENTATION

In this appendix, we provide a brief summary of how the modified AMC runtime protocol can be implemented.

First, the RTOS needs to track the start time $s[i]$ of each currently active priority level-i busy period, see Section III for a definition. This can be achieved via $O(1)$ additional operations on each job release as follows. When a new job of a task τ_i is released at time t and task τ_i is inserted in the run-queue: (i) if τ_i is added to the head of the run-queue, i.e. it has the highest priority of any task with an active job, then $s[i] = t$, (ii) otherwise the busy period start time is inherited, $s[i] = s[k]$, from the task τ_k that is immediately ahead of task τ_i in the run-queue, i.e. task τ_k is the next higher priority task with an active job. Note, correct tracking of busy period start times requires that tasks with simultaneous job releases are added to the run-queue in priority order, highest priority first.

Second, the RTOS needs to manage entry to and exit from degraded mode via a programmable timer interrupt and an ordered expiry-queue of absolute response time expiry values: $s[i] + R_i(LO)$ for the tasks with active jobs. When a job of a HI-criticality task is released, its response time expiry value is inserted (requiring $O(\log n)$ operations) into the expiry-queue. While the system is in normal mode, the programmable timer is set such that it will interrupt at the expiry time indicated by the value at the head of the expiry-queue, if any. When the timer interrupt goes off, the system enters degraded mode. On completion of a job of a HI-criticality task, its response time expiry value is removed from the expiry-queue (requiring $O(1)$ operations). Further, with AMC-RH, if the system is in degraded mode and the expiry time now at the head of the expiry-queue has not yet been reached or the expiry-queue is empty, then degraded mode is exited and the timer interrupt reset. For AMC-RA, degraded mode is only exited on an idle instant, i.e. when the run-queue becomes empty.

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] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–395, 1986. [Online]. Available: https://doi.org/10.1093/comjnl/29.5.390

[2] N. C. Audsley, A. Burns, M. Richardson, K. W. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292(8), September 1993. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/sej.1993.0034

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*. IEEE Computer Society, 1990, pp. 182–190. [Online]. Available: https://doi.org/10.1109/REAL.1990.128746

[4] S. K. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. IEEE Computer Society, 2011, pp. 34–43. [Online]. Available: https://doi.org/10.1109/RTSS.2011.12

[5] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "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, 2020, pp. 3–11. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00012

[6] ——, "A comprehensive survey of industry practice in real-time systems," *Real-Time Syst.*, p. 41, 2021. [Online]. Available: https://doi.org/10.1007/s11241-021-09376-1

[7] S. Law, I. Bate, and B. Lesage, "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*, ser. LIPIcs, S. Quinton, Ed., vol. 133. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 8:1–8:22. [Online]. Available: https://doi.org/10.4230/LIPIcs.ECRTS.2019.8

[8] ——, "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*, L. Cucu-Grosjean, R. Medina, S. Altmeyer, and J. Scharbarg, Eds. ACM, 2020, pp. 100–110. [Online]. Available: https://doi.org/10.1145/3394810.3394814

[9] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 239–243. [Online]. Available: https://doi.org/10.1109/RTSS.2007.47

[10] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 82:1–82:37, 2018. [Online]. Available: https://doi.org/10.1145/3131347

[11] ——, "Mixed criticality systems: A review (12th edition)," Department of Computer Science, University of York, Tech. Rep. MCC-1(M), available at https://www-users.cs.york.ac.uk/~burns/review.pdf, 2019.

[12] S. K. Baruah and A. Burns, "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*, ser. Lecture Notes in Computer Science, A. B. Romanovsky and T. Vardanega, Eds., vol. 6652. Springer, 2011, pp. 174–188. [Online]. Available: https://doi.org/10.1007/978-3-642-21338-0_13

[13] H. Huang, C. D. Gill, and C. Lu, "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*, M. D. Natale, Ed. IEEE Computer Society, 2012, pp. 23–32. [Online]. Available: https://doi.org/10.1109/RTAS.2012.16

[14] Q. Zhao, Z. Gu, and H. Zeng, "PT-AMC: integrating preemption thresholds into mixed-criticality scheduling," in *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, E. Macii, Ed. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 141–146. [Online]. Available: https://doi.org/10.7873/DATE.2013.042

[15] Q. Zhao, Z. Gu, H. Zeng, and N. Zheng, "Schedulability analysis and stack size minimization with preemption thresholds and mixed-criticality scheduling," *J. Syst. Archit.*, vol. 83, pp. 57–74, 2018. [Online]. Available: https://doi.org/10.1016/j.sysarc.2017.03.007

[16] T. Fleming and A. Burns, "Extending mixed criticality scheduling," in *Workshop on Mixed Criticality Systems (WMC)*, 2013, pp. 7–12.

[17] S. K. Baruah and B. Chattopadhyay, "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, 2013, pp. 237–246. [Online]. Available: https://doi.org/10.1109/RTCSA.2013.6732224

[18] S. Baruah, A. Burns, and R. I. Davis, "An extended fixed priority scheme for mixed criticality systems," in *Workshop on Real-Time Mixed Criticality Systems (ReTiMics) 2013, 21st August, Taipei, Taiwan*, G. L. Laurent George, Ed., 2013, pp. 18–24. [Online]. Available: https://www-users.cs.york.ac.uk/~robdavis/papers/jitterRTCSA.pdf

[19] A. Burns and R. I. Davis, "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, 2013, pp. 125–134. [Online]. Available: https://doi.org/10.1109/ECRTS.2013.23

[20] ——, "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, 2014, pp. 21–30. [Online]. Available: https://doi.org/10.1109/RTSS.2014.12

[21] O. Gettings, S. Quinton, and R. I. Davis, "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*, J. Forget, Ed. ACM, 2015, pp. 237–246. [Online]. Available: https://doi.org/10.1145/2834848.2834850

[22] D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran, "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*, E. Bini and C. Pagetti, Eds. ACM, 2017, pp. 237–246. [Online]. Available: https://doi.org/10.1145/3139258.3139276

[23] R. I. Davis, S. Altmeyer, and A. Burns, "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*, R. Pellizzoni, Ed. IEEE Computer Society, 2018, pp. 140–151. [Online]. Available: https://doi.org/10.1109/RTAS.2018.00024

[24] A. Burns, R. I. Davis, S. K. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Trans. Computers*, vol. 67, no. 10, pp. 1478–1491, 2018. [Online]. Available: https://doi.org/10.1109/TC.2018.2831227

[25] A. Burns and R. I. Davis, "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, 2020, pp. 12–24. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00013

[26] S. Asyaban and M. Kargahi, "An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real Time Syst.*, vol. 54, no. 1, pp. 32–90, 2018. [Online]. Available: https://doi.org/10.1007/s11241-017-9287-2

[27] I. Pavic and H. Dzapo, "Commentary to: An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems," *Real Time Syst.*, vol. 56, no. 1, pp. 112–119, 2020. [Online]. Available: https://doi.org/10.1007/s11241-020-09345-0

[28] H. Su and D. Zhu, "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*, E. Macii, Ed. EDA Consortium San Jose, CA, USA / ACM DL, 2013, pp. 147–152. [Online]. Available: https://doi.org/10.7873/DATE.2013.043

[29] H. Su, P. Deng, D. Zhu, and Q. Zhu, "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, 2016, pp. 59–68. [Online]. Available: https://doi.org/10.1109/RTCSA.2016.16

[30] A. Burns and S. Baruah, "Towards a more practical model for mixed criticality systems," in *Workshop on Mixed Criticality Systems (WMC)*, 2013, pp. 1–6. [Online]. Available: https://www-users.cs.york.ac.uk/~robdavis/wmc2013/paper3.pdf

[31] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "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*, C. Seatzu, Ed. IEEE, 2013, pp. 1–8. [Online]. Available: https://doi.org/10.1109/ETFA.2013.6647967

[32] T. Fleming and A. Burns, "Incorporating the notion of importance into mixed criticality systems," in *Workshop on Mixed Criticality Systems (WMC)*, Dec 2014, pp. 33–38. [Online]. Available: https://www-users.cs.york.ac.uk/~robdavis/wmc2014/6.pdf

[33] Y. Abdeddaïm, "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*, ser. Lecture Notes in Computer Science, B. B. Hedia, Y. Chen, G. Liu, and Z. Yu, Eds., vol. 12519. Springer, 2020, pp. 131–146. [Online]. Available: https://doi.org/10.1007/978-3-030-65955-4_10

[34] S. Punnekkat, R. I. Davis, and A. Burns, "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*, ser. Lecture Notes in Computer Science, R. K. Shyamasundar and K. Ueda, Eds., vol. 1345. Springer, 1997, pp. 72–82. [Online]. Available: https://doi.org/10.1007/3-540-63875-X_44

[35] F. Santy, L. George, P. Thierry, and J. Goossens, "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*, R. Davis, Ed. IEEE Computer Society, 2012, pp. 155–165. [Online]. Available: https://doi.org/10.1109/ECRTS.2012.39

[36] L. Santinelli and Z. Guo, "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, 2018, pp. 313–320. [Online]. Available: https://doi.org/10.1109/ETFA.2018.8502493

[37] M. Völp, M. Roitzsch, and H. Härtig, "Towards an interpretation of mixed criticality for optimistic scheduling," in *Work-In-Progress Session 21st IEEE Real-Time and Embedded Technology and Applications Symposium*, 2015, pp. 15–16.

[38] B. Hu, L. Thiele, P. Huang, K. Huang, C. Griesbeck, and A. C. Knoll, "FFOB: efficient online mode-switch procrastination in mixed-criticality systems," *Real Time Syst.*, vol. 55, no. 3, pp. 471–513, 2019. [Online]. Available: https://doi.org/10.1007/s11241-018-9323-x

[39] I. Bate, A. Burns, and R. I. Davis, "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, 2015, pp. 259–268. [Online]. Available: https://doi.org/10.1109/ECRTS.2015.30

[40] ——, "An enhanced bailout protocol for mixed criticality embedded software," *IEEE Trans. Software Eng.*, vol. 43, no. 4, pp. 298–320, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2016.2592907

[41] S. Iacovelli and R. Kirner, "A lazy bailout approach for dual-criticality systems on uniprocessor platforms," *Designs*, vol. 3, no. 1, 2019. [Online]. Available: https://www.mdpi.com/2411-9660/3/1/10

[42] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "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*, J. Forget, Ed. ACM, 2015, pp. 139–148. [Online]. Available: https://doi.org/10.1145/2834848.2834869

[43] R. Ernst and M. D. Natale, "Mixed criticality systems - A history of misconceptions?" *IEEE Des. Test*, vol. 33, no. 5, pp. 65–74, 2016. [Online]. Available: https://doi.org/10.1109/MDAT.2016.2594790

[44] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the Real-Time Systems Symposium - 1990, Lake Buena Vista, Florida, USA, December 1990*. IEEE Computer Society, 1990, pp. 201–209. [Online]. Available: https://doi.org/10.1109/REAL.1990.128748

[45] R. I. Davis, "On exploiting spare capacity in hard real-time systems," Ph.D. dissertation, University of York, UK, 1995. [Online]. Available: https://www-users.cs.york.ac.uk/~robdavis/papers/YCST-96-06.pdf

[46] N. C. Audsley, "On priority assignment in fixed priority scheduling," *Inf. Process. Lett.*, vol. 79, no. 1, pp. 39–44, 2001. [Online]. Available: https://doi.org/10.1016/S0020-0190(00)00165-4

[47] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007), 3-6 December 2007, Tucson, Arizona, USA*. IEEE Computer Society, 2007, pp. 3–14. [Online]. Available: https://doi.org/10.1109/RTSS.2007.11

[48] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," in *Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada: The Engineering of Correct and Reliable Software for Real-Time & Distributed Systems using Ada and Related Technologies 2004, Atlanta, GA, USA, November 14-14, 2004*, J. W. McCormick and R. E. Sward, Eds. ACM, 2004, pp. 1–8. [Online]. Available: https://doi.org/10.1145/1032297.1032298

[49] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. R. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[50] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of real-time embedded systems," in *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*. IEEE Computer Society, 2012, pp. 351–362. [Online]. Available: https://doi.org/10.1109/RTSS.2012.85

[51] D. Griffin, I. Bate, and R. I. Davis, "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, 2020, pp. 76–88. [Online]. Available: https://doi.org/10.1109/RTSS49844.2020.00018

[52] ——, "Implementation of the dirichlet rescale algorithm (drs) dgdguk/drs: v2.0.0 (open source software)," Nov. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4264857

[53] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real Time Syst.*, vol. 25, no. 1, pp. 5–37, 2003. [Online]. Available: https://doi.org/10.1023/A:1022920502619

[54] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2010, pp. 6–11. [Online]. Available: http://retis.sssup.it/waters2010/waters2010.pdf

[55] T. P. Baker, "Stack-based scheduling of realtime processes," *Real Time Syst.*, vol. 3, no. 1, pp. 67–99, 1991. [Online]. Available: https://doi.org/10.1007/BF00365393