A Bailout Protocol for Mixed Criticality Systems

Iain Bate, Alan Burns, Robert I. Davis
Department of Computer Science, University of York, York, UK.
email: {iain.bate, alan.burns, rob.davis}@york.ac.uk

***Abstract*—To move mixed criticality research into industrial practice requires models whose run-time behaviour is acceptable to systems engineers. Certain aspects of current models, such as abandoning lower criticality tasks when certain situations arise, do not give the robustness required in application domains such as the automotive and aerospace industries. In this paper a new bailout protocol is developed that still guarantees high criticality tasks but minimises the negative impact on lower criticality tasks via a timely return to normal operation. We show how the bailout protocol can be integrated with existing techniques, utilising offline slack to further improve performance. Static analysis is provided for the strong schedulability guarantees, while scenario-based evaluation via simulation is used to explore the effectiveness of the protocol.**

I. Introduction

An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. At the same time, these platforms are migrating from single cores to multi-cores and, in the future, many-core architectures. Criticality is a designation of the level of assurance against failure needed for a system component, where the level of assurance needed depends on both the likelihood of failure and the consequences of that failure [21]. A mixed criticality system (MCS) is one that has two or more distinct levels (for example safety critical and mission critical). Perhaps up to five levels may be identified (see, for example, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards).

Most of the complex embedded systems found in, for example, the automotive and avionics industries are evolving into integrated rather than federated mixed criticality systems in order to meet stringent non-functional requirements relating to cost, space, weight, heat generation and power consumption; the latter being of particular relevance to mobile systems.

The fundamental research question underlying these initiatives and standards is: how, in a disciplined way, to reconcile the conflicting requirements of *partitioning* for assurance and *sharing* for efficient resource usage. This question gives rise to theoretical problems in modeling and verification, and systems problems relating to the design and implementation of the necessary hardware and software run-time controls.

Although the formal study of mixed criticality systems is a relatively new endeavour, starting with the paper by Vestal (Honeywell Aerospace) in 2007 [30], a standard model has emerged (see for example [6], [8], [15], [19], [23], [27]) For dual criticality systems this standard model has the following properties:

- Each task is characterised by its criticality level (e.g. HI- or LO-criticality), the minimum inter-arrival time of its jobs (period denoted by T), deadline (relative to the release of each job, denoted by D) and worst-case execution time (one per criticality level up to the criticality level of the task), denoted by $C(HI)$ and $C(LO)$. A key aspect of the standard MCS model is that $C(HI) \geq C(LO)$ [30].
- The system executes in one of two modes. It starts in the LO-criticality mode, and remains in that mode as long as all jobs execute within their LO-criticality execution times ($C(LO)$).
- If any LO-criticality job executes for its $C(LO)$ execution time without completing then that job is immediately aborted by a runtime monitoring mechanism.
- If any HI-criticality job executes for its $C(LO)$ execution time without completing then the system immediately moves to the HI-criticality mode. As the system moves to this mode all LO-criticality tasks are abandoned. No further LO-criticality jobs are executed. The system remains in the HI-criticality mode.

The motivation for the standard model having two values for the Worst-Case Execution Time (WCET) [30], [14] is taken from either of two situations often seen in industrial practice [22]. The first situation involves the High WaterMark (HWM), i.e. the largest execution time observed during testing, which is highly reliable as testing for functional correctness is intensive (e.g. MCDC coverage). This value would be taken as $C(LO)$. However for the most critical software an engineered safety margin is added to give a $C(HI)$ value. Values for this engineered safety margin come from industrial practice and are based on engineering judgement and experience. A margin of around 20% is typical in aerospace applications[1]. It is considered sufficiently unlikely that this value will be exceeded[2]. The second situation is when static or hybrid analysis is used to obtain a WCET, which can be treated as $C(HI)$. Even though this value is considered safe [22], it is often too pessimistic, and its use may lead to difficulties in obtaining a schedulable system. Again the HWM may be used as $C(LO)$. In both cases, it is necessary that the system is schedulable when all tasks execute for $C(LO)$; however it is also important to gracefully degrade when $C(LO)$ is exceeded, i.e. HI-criticality tasks must still meet their deadlines and as

[1]Note, we know of no theoretical support for using such a value, rather such margins come from engineering experience.

[2]In some systems, further runtime monitoring may be employed to ensure that such overruns, however unlikely, do not lead to complete system failure.

few as possible of the LO-criticality tasks miss their deadlines.

The abstract behavioural model described above has been useful in allowing key properties of mixed criticality systems to be derived, but it is open to criticism from systems engineers that it does not match their expectations [21]. In particular, in the HI-criticality mode LO-criticality tasks should not be abandoned. Some level of service should be maintained if at all possible, as LO-criticality tasks are still critical. It should be possible for the system to return to the LO-criticality mode as soon as conditions are appropriate. In this mode all functionality should be provided.

Clearly, in general, if the system is in the HI-criticality mode and all HI-criticality tasks are executing for the maximum time defined for such tasks then the LO-criticality tasks will not be able to receive enough execution time to guarantee that their deadlines are met. However, in many situations the worst-case conditions will not be experienced and in this case LO-criticality tasks should receive some level of service.

In the following section, we discuss related work. Approaches to degraded service are considered in Section III. We then define, in Section IV, a new *bailout protocol* for MCSs in which HI-criticality tasks are not allowed to fail (they are too important to fail) and therefore LO-criticality tasks must sacrifice their quality of service by not starting a certain number of jobs. The actual number of sacrificed jobs depends on the size of the bailout and the time needed for recovery. However, once the bailout has been serviced the LO-criticality tasks can return to their full timely behaviour. A key aspect of this paper is the evaluation of MCS protocols via scenario-based simulation; this is addressed in Section VI. Analysis for the bailout protocol is given in Section VII. Section VIII concludes with a summary and a discussion of future work.

II. Related Work

Background material on MCS research can be obtained from the following sources [5]–[8], [18], [19], [23], [30]. and from an ongoing survey [14]. While mixed criticality behaviour has some similarities to traditional mode changes, there are also significant differences [21], [12]. These include the mode change being driven by temporal rather than functional behaviour, permitting a more specific schedulability analysis.

A. Current Scheduling Analysis and its Limitations

Although the standard model requires an immediate change to the HI-criticality mode and the consequential abandonment of all active LO-criticality jobs, the analysis of this model has shown [3]–[5] that the mixed criticality schedulability problem is strongly NP-hard even if there are only two criticality levels. Hence only sufficient rather than exact analysis is possible. One of the consequences of this constraint is that a significant proportion of the available analyses that have been produced for MCSs actually assume that any LO-criticality job that has been released by the time of the mode change will complete, rather than being aborted.

For example, the Adaptive Mixed Criticality (AMC Method 1 or AMC-rtb) approach presented at RTSS in 2011 [6] first computes the worst-case response times for all tasks in the LO-criticality mode (denoted by $R(LO)$). This is accomplished by solving, via fixed point iteration, the following response-time equation for each task τ_i:

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

where $\mathbf{hp(i)}$ is the set of all tasks with priority higher than that of task τ_i.

During the criticality change the only concern is HI-criticality tasks, for these tasks:

$$R_i(HI) \;=\; C_i(HI) \;+\; \sum_{\forall j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \;+$$

$$\sum_{\forall k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (2)$$

where $\mathbf{hpH(i)}$ is the set of HI-criticality tasks with priority higher than that of task τ_i and $\mathbf{hpL(i)}$ is the set of LO-criticality tasks with priority higher than that of task τ_i. So $\mathbf{hp(i)}$ is the union of $\mathbf{hpH(i)}$ and $\mathbf{hpL(i)}$. Note $R_i(HI)$ is only defined for HI-criticality tasks.

This equation takes into account the fact that LO-criticality tasks cannot execute for the entire busy period of a HI-criticality task in the HI-criticality mode. A change to the HI-criticality mode must occur at or before $R_i(LO)$ which 'caps' the interference from LO-criticality tasks as $R_i(HI)$ must be greater than $R_i(LO)$.

The cap is however at the maximum possible level. The maximum number of LO-criticality jobs are assumed to interfere and each of these jobs is assumed to complete – each inducing the maximum interference of $C_k(LO)$. Note that if, for any HI-criticality task, $R_i(HI) \leq D_i$ during the transition to the HI-criticality mode then the task will remain schedulable once the HI-critically mode is fully established and there is no interference from LO-criticality tasks.

This AMC approach assumes that once the system goes into the HI-criticality mode then it will stay in that mode. As discussed in the introduction this is not an acceptable behaviour in practice. A simple but necessary extension to AMC is therefore to allow a switch back to the LO-criticality mode when the system experiences an *idle instant* (a point in time at when there are no ready jobs that were released prior to that time). This is a well-known protocol for controlling mode changes [29]. In this paper we will refer to this extended approach as AMC+.

In the remainder of this paper, for AMC and AMC+, we assume that any job of a LO-criticality task that is released before HI-criticality mode is entered may complete its execution, since this is allowed by the analysis; however, LO-criticality jobs released during HI-criticality mode are abandoned by these schemes.

III. Degraded Service for LO-Criticality Tasks

The key properties of MCS scheduling are (i) that if all tasks execute within their $C(LO)$ bounds then all deadlines for all task will be satisfied, and (ii) that HI-criticality tasks will always meet their deadlines, assuming that they execute within their $C(HI)$ bounds.

Notwithstanding these key static properties of a system, an actual implementation must exhibit clear and effective behaviours for all of its potential run-time characteristics. In particular, for a dual criticality system, if at some point during its execution only the HI-criticality jobs can be guaranteed, then what level of service can be expected for the LO-criticality jobs? As indicated in the introduction it is not acceptable to permanently abandon these tasks just because they cannot be fully guaranteed.

The dual requirement (both to meet all deadlines and to have sensible behaviour when deadlines are missed) is not a contradiction, rather it is a necessary property of any robust system model. MCSs have, in this regard, a number of similarities to fault tolerance systems: faults should be avoided, but also faults should be tolerated and result in minimum disturbance to the system [12].

There are various forms of degraded service that can be defined for LO-criticality tasks: Run all tasks, but extend their periods and/or deadlines – sometimes called the elastic task model [28]. Run all tasks but reduce the executions times of LO-criticality tasks (i.e. $C(HI) \leq C(LO)$ for these tasks) [13] – perhaps by switching to simpler version of the software. Drop jobs from a specific subset of tasks [24], [20].

An orthogonal approach to improving the overall service for LO-criticality tasks was adopted by Santy et al. [27]. They effectively scale the $C(LO)$ values using sensitivity analysis until the system is just schedulable. Using these values at runtime makes the system more robust, since LO-criticality tasks can execute for longer, and HI-criticality tasks are less likely to exceed their larger budgeted $C(LO)$ values, making the system less likely to enter its HI-criticality mode. This approach was subsequently refined by Burns and Baruah [13] using Robust Priority Assignment techniques [17] that permit priorities to change during the sensitivity analysis process.

A further important aspect of providing service for LO-criticality tasks is the ability to restore the system to its LO-criticality mode following an interval of HI-criticality behaviour. As mentioned previously, this can be achieved by waiting for an idle instant. Santy et al. explored this approach [27], and also developed a protocol for multiprocessor scheduling where there may be no idle instant across all processors [26].

In this paper, we introduce the *Bailout Protocol*. This protocol allows jobs to be dropped, but rather than abandon jobs that have been released, and so waste the consumed execution time and potentially leave them in an inconsistent state, it allows these jobs to continue. However, it disables the release of new jobs of LO-criticality tasks until the system is back into the normal mode of execution whereby it can again guarantee all tasks. As noted previously many forms of analysis actually reduce their complexity by assuming all released jobs will complete. The bailout protocol aims to restore LO-criticality mode as soon as possible following an interval of HI-criticality activity, and so minimise the number of LO-criticality jobs that miss deadlines or are not executed.

IV. The Bailout Protocol

We describe the Bailout Protocol assuming two levels of criticality in the system software. We focus on single processor systems (or individual cores in a partitioned multiprocessor system) and the fixed priority pre-emptive scheduling (FPPS) of sporadic tasks with constrained deadlines.

At run-time, dual criticality systems are typically defined to be in one of two modes: *LO-criticality mode* and *HI-criticality mode*. With the bailout protocol, we defined three modes: *normal mode*, *bailout mode* and *recovery mode*. Normal mode corresponds to the traditional LO-criticality mode, since every LO-criticality job with a release time and a deadline in a single normal mode interval must be guaranteed to complete by its deadline. Bailout and recovery modes correspond to the traditional HI-criticality mode.

The bailout protocol comprises the following modes and mechanisms, which operate only in the mode for which they are described. In all modes, LO-criticality tasks are prevented from executing for more than their $C(LO)$ values. LO-criticality tasks dispatched in normal mode, continue to execute in both bailout and recovery modes. (Note, such jobs may miss their deadlines in these modes, but continue to execute provided they do not exceed $C(LO)$).

Normal mode:

(i) While all jobs of HI-criticality tasks execute for no more than their $C(LO)$ values, the system remains in normal mode.

(ii) If any HI-criticality job executes for its $C(LO)$ value without signalling completion it must take out a loan of $C(HI) - C(LO)$; this loan is always granted, and the system moves into the bailout mode. The bailout fund (BF) is initialised to $BF = C(HI) - C(LO)$.

Bailout mode:

(iii) If any HI-criticality job executes for its $C(LO)$ value without signalling completion then it must also take out a loan of $C(HI) - C(LO)$, adding to the bailout fund: $BF = BF + C(HI) - C(LO)$.

(iv) If any HI-criticality job completes with an execution time of e, with $e \leq C(LO)$ then it donates its underspend (if any), reducing the bailout fund: $BF = BF - (C(LO) - e)$.

(v) If any LO-criticality job completes with an execution time of e, with $e \leq C(LO)$ then it donates its underspend (if any) to the bailout fund: $BF = BF - (C(LO) - e)$. (Such a job would need to have been released in an earlier normal mode).

(vi) If any HI-criticality job with a loan completes with an execution time of e, with $C(LO) < e \leq C(HI)$ then it donates its loan underspend, reducing the bailout fund: $BF = BF - (C(HI) - e)$.

(vii) LO-criticality jobs released in bailout mode are abandoned (not started). Further, when the scheduler would otherwise have dispatched such a job, the job's budget of $C(LO)$ is donated to the bailout fund: $BF = BF - C(LO)$.

(viii) If the bailout fund becomes zero (note BF is constrained to never become negative), then the lowest priority HI-criticality job with outstanding execution is recorded (let this job be J_k) and the recovery mode is entered [3].

(ix) If during bailout mode, an idle instant occurs, then a transition is made to normal mode, and BF is reset to zero.

Recovery mode:

(x) LO-criticality jobs released in recovery mode are abandoned (not started).

(xi) If any HI-criticality job executes for its $C(LO)$ value without signalling completion, then the system re-enters bailout mode – as described in (ii) above.

(xii) When the job J_k noted at the point when recovery mode was last entered completes, then the system transitions to normal mode.

The bailout protocol is designed to have a simple implementation, with each operation (i) to (xii) requiring only $O(1)$ time, and all actions taking place at the release, completion, or context switch to a job, all of which are already well defined RTOS operations in FPPS, or when a job executes for $C(LO)$ without signalling completion, which is necessarily required by a MCS implementing AMC.

We now give an example illustrating the behaviour of the bailout protocol. This example includes four tasks: τ_1 and τ_2 are LO-criticality tasks, while τ_3 and τ_4 are HI-criticality. Task τ_1 has the highest priority and task τ_4 the lowest. The parameters of the tasks are given in the table below. The tasks are schedulable according to the AMC-rtb schedulability test with the $R(LO)$ and $R(HI)$ upper bounds on the worst-case response times given in the table below.

τ_i	L	$C_i(LO)$	$C_i(HI)$	T_i	D_i	$R(LO)$	$R(HI)$
τ_1	LO	8	-	24	12	8	-
τ_2	LO	4	-	26	12	12	-
τ_3	HI	4	10	48	24	16	22
τ_4	HI	8	8	32	32	24	30

Figure 1 illustrates the behaviour of the bailout protocol. At time $t = 16$, task τ_3 has executed for $C(LO)$ without signalling completion, hence bailout mode is entered. As $C(HI) = 10$, BF is initialised to 6. Task τ_3 completes its HI-criticality execution at time $t = 22$; however, the system cannot simply resume normal mode behaviour, since then the releases of task τ_1 and τ_2 at $t = 24$ and $t = 26$ respectively would result in task τ_4 (HI-criticality) missing its deadline. Instead, since $BF > 0$, the system remains in bailout mode. At time $t = 24$ the second job of task τ_1 is released; however, as the system is in bailout mode, and the task is of LO-criticality, then the job is abandoned at the time it would have

[3]Job J_k defines the extent of the recovery mode, which is necessary to ensure that no HI-criticality job can be subject to more interference than accounted for by the analysis of AMC, for further details see Theorem 2 in Section VII and the discussion that follows it.

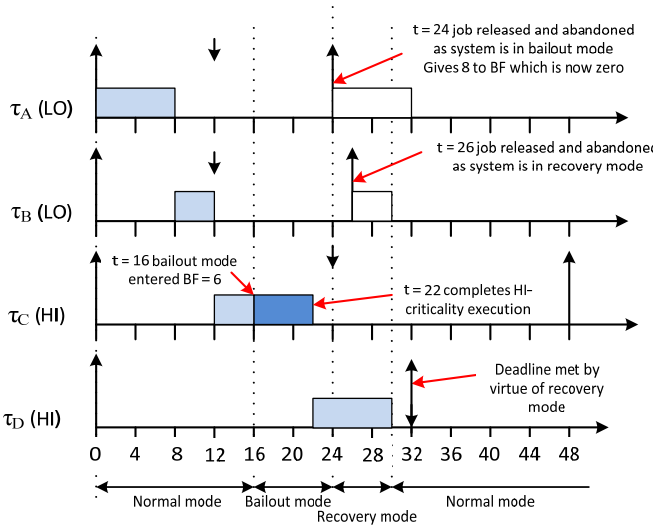

Fig. 1. Example showing the operation of the bailout protocol, including normal, bailout and recovery modes.

started to execute ($t = 24$ in this case) repaying the bailout fund, which now goes to zero. However, the system still cannot resume normal mode operation, as doing so would result in task τ_4 (HI-criticality) missing its deadline due to interference from the second job of task τ_2. Instead the system enters recovery mode and records the lowest priority HI-criticality job with outstanding execution. This is the first job of task τ_4. When this job completes at $t = 30$, the system re-enters normal mode. It is interesting to note that in this example, if task τ_4 were a LO-criticality task, then recovery mode would end immediately (i.e. at the same time as bailout mode at $t = 24$), the second job of task τ_2 would not be abandoned, and task τ_4 would miss its deadline. This shows that under the bailout protocol, (in common with AMC) LO-criticality jobs with release times and deadlines that span some HI-criticality behaviour cannot be guaranteed to meet their deadlines, even if the system returns to LO-criticality behaviour before they complete.

V. Increasing Execution Time Budgets

In this section we describe an offline method which is complementary to the bailout protocol, reducing the number of times that a given system will go into bailout mode, and the amount of time that it spends in that mode, hence reducing the number of LO-criticality jobs that miss their deadlines or are abandoned.

The offline method was introduced by Santy et al. [27] and further refined by Burns and Baruah [13]. It uses sensitivity analysis [11], [25] to explore by how much $C(LO)$ values can be changed without making the system unschedulable, effectively making use of the statically available slack in the system [16]. Intuitively, this method is compatible with the bailout protocol, since it effectively increases budgeted $C(LO)$ values while ensuring that the system remains provably schedulable.

The specific method we use is as follows: First, we increase the $C(LO)$ values of *all* HI-criticality tasks as much as

possible while ensuring that the system remains schedulable according to AMC-rtb analysis (i.e. (1) and (2)). We do this by forming a binary search for the largest value of α such that the system remains schedulable when all HI-criticality task's $C(LO)$ values are replaced by $C(BU) = min(C(HI), \alpha C(LO))$. Note we use $C(BU)$ rather than $C(LO)$ to emphasize that these are no longer the LO-criticality WCETs associated with those HI-criticality tasks, but rather *execution time budgets* that will be used to police normal mode behaviour 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-rtb to begin with, and the initial upper value is given by the largest $C(HI)/C(LO)$ for any HI-criticality task. At each step of the binary search, Audsley's Optimal Priority Assignment algorithm [1] 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. Second, we use a similar process to further increase, if possible, the $C(BU)$ value for each individual task in turn, since after the first step, some but not all of the $C(BU)$ values may still be increased without making the system unschedulable. (We do this for all HI-criticality tasks in order of increasing deadlines).

At runtime, we use FPPS along with the bailout protocol, replacing all occurrences of $C(LO)$ for HI-criticality tasks by the larger $C(BU)$ values. We refer to the basic bailout protocol as BP, and the more sophisticated approach described here as BPS (Bailout Protocol with Sensitivity analysis). For systems that are schedulable under classical FPPS (i.e. assuming that all jobs may take an execution time that corresponds to their own criticality level), then BPS has the useful property, unlike AMC+ and BP, that no LO-criticality jobs miss their deadlines. This is the case, since for such systems the first step described above will result in $C(BU) = C(HI)$ for all HI-criticality tasks. The AMC+ approach may also take advantage of increased $C(BU)$ values. We refer to such an approach as AMC+S.

VI. Scenario-Based Evaluation

In this section, we present a scenario-based evaluation of the performance of the bailout protocol using an experimental framework or simulation. Scenario-based evaluation is an essential complement to schedulability analysis as the latter only tells us under what conditions timing requirements are met, whereas we are also interested in the amount of time spent outside of normal mode, and consequently how many LO-criticality tasks miss their deadlines. Our evaluation aims to provide an understanding of how the different scheduling schemes (AMC+, AMC+S, BP, BPS) meet the needs of mixed-criticality systems. The first step in this process is the selection of evaluation metrics.

A. Evaluation Metrics

We use the following key evaluation metrics. This combination of metrics covers the percentage of deadlines missed, broken down into HI- and LO-criticality tasks, as well as providing insight into the operation of the bailout protocol.

1) *Percentage of HI-criticality Deadline Misses ($\#HDM$)*: These deadline misses should not be experienced with the bailout or AMC schemes, but may occur with standard FPPS.
2) *Jobs Not Executed ($\#JNE$)*: The percentage of LO-criticality jobs that are abandoned.
3) *Percentage of LO-criticality Deadline Misses ($\#LDM$)*: For LO-criticality jobs that are executed.

The most important metric is $\#HDM$, since any valid protocol must ensure first that there are no HI-criticality deadline misses. Given that, then the next metric to optimise is the percentage of LO-criticality jobs that fail to meet their deadlines, either by missing their deadlines ($\#LDM$) or not being executed $\#JNE$. Although the simulator computes $\#LDM$, this number is far smaller than $\#JNE$, we therefore do not separately show $\#LDM$ in the graphs presented below; these values are however shown in the accompanying tables.

B. Experimental Framework

The experimental framework consists of four principal components: scheduling schemes, task set generation, configurations, and simulation.

Scheduling Schemes: The scheduling schemes were implemented using a layered approach, with FPPS used to schedule the tasks, and additional mechanisms used to control release, dispatch and execution of jobs according to the different approaches considered:

1) *Default (FPPS)* – Basic FPPS where execution time overruns are allowed.
2) *Bailout Protocol (BP)* – The basic bailout protocol (section IV).
3) *Bailout Protocol - Slack (BPS)* – The bailout protocol enhanced by offline increases in execution time budgets making use of static slack (section V).
4) *Adaptive Mixed Criticality - (AMC+)* – The standard AMC scheme [6] (section II-A), enhanced so the system resumes LO-criticality execution after an idle instant.
5) *Adaptive Mixed Criticality - Slack (AMC+S)* – The AMC+ scheme, enhanced by offline increases in execution time budgets making use of static slack (section V).

Task Set Generation: Task sets of cardinality 20 were generated according to the following parameters.

1) *Periods and Deadlines* - The period of each of the tasks was chosen at random from a set of harmonics of two base frequencies (e.g. 25, 50, 100, 250, 500, 1000 and 20, 40, 80, 200, 400, 800ms) as typically found in automotive and avionics systems [9]. The deadlines were implicit.
2) *Execution Times* - LO-criticality utilisation $U(LO)$ values for each task where determined according to the Uunifast algorithm [10], thus ensuring an unbiased distribution of values that sum to the target utilisation for the system. LO-criticality execution times were then set to $C(LO) = U(LO).T$, and HI-criticality execution times to $C(HI) = CF.C(LO)$ where CF is the criticality factor ($CF = 2.0$). Finally, best case execution times

(BCET) were chosen at random between 80% and 100% of C(LO).

3) *Criticality* - ($CP = 0.5$) Tasks were randomly chosen to be either HI- or LO-criticality, with a 50% probability of being HI-criticality.

We note that since $CF = 2.0$ and $CP = 0.5$, then the total HI-criticality utilisation was approximately equal to the total LO-criticality utilisation.

Configurations: An important issue for this research is understanding how the different scheduling schemes perform in different circumstances, in terms of both typical and worst-case behaviours. The configurations we used were as follows.

1) *Config 1 - 70% LO-criticality Utilisation* - Only task sets with LO-criticality utilisation of 70% were used.
2) *Config 2 - 90% LO-criticality Utilisation* - One of the benefits of some of the mixed-criticality scheduling approaches is that task sets with overall utilisation exceeding 100% are schedulable as LO-criticality tasks do not have to be executed all of the time [6]. For this configuration, only task sets with LO-criticality utilisations of 90% were used, meaning that many of the task sets had an overall utilisation exceeding 100% (when accounting for HI-criticality execution times).

In both configurations, we required that the task sets chosen had at least one task that was unschedulable according to exact analysis of FPPS [2], but were schedulable according to AMC-rtb [6]. Thus the configurations represent cases where both LO- and HI- criticality jobs may miss their deadlines under classical FPPS, but not when the AMC or Bailout schemes are employed. Further, we required that the number of HI-criticality tasks was actually in the range 40% to 60% of the total number of tasks (recall that each individual task had a 50% probability of being HI-criticality).

Simulation: Our experiments covered 1,000 task sets for each of the two configurations considered. The duration of each simulation run was 10^{11} time units, each time unit was 0.1ms, thus this was sufficient for 10^5 jobs of the longest period task.

In the simulation, job releases were strictly periodic. On each release, an actual execution time was chosen for the job as follows. If the job was from a LO-criticality task, then this value was chosen at random from a uniform distribution in the range $[BCET, C(LO)]$. If the job was from a HI-criticality task, then a random boolean variable with a probability of 10^{-4} of returning $true$ was used to determine if the job would exhibit HI-criticality behaviour. If $true$ was returned, then its execution time was chosen at random from a uniform distribution in the range $[C(LO), C(HI)]$, otherwise the range was $[BCET, C(LO)]$. The probability used to determine if HI-criticality behaviour would be exhibited was deliberately set to a relatively high value as we wanted to stress the system behaviour. In practice such a high value is perhaps unlikely, but possible, for example if the High WaterMark testing used to determine $C(LO)$ had not revealed the worst-case path[4].

[4]We note that functional testing, even that requiring MCDC coverage, is not in general sufficient to determine WCETs.

Note for the schemes making use of statically available slack, the $C(BU)$ parameters were computed via offline sensitivity analysis, as described in Section V, before running the simulator. These values were then used by the simulator to determine when the system should transition to HI-criticality or bailout mode, with the $C(LO)$ values used in the selection of job execution times, as explained above. We note that 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 budget accounting.

C. Evaluation Results

Our evaluation results are shown using box and whisker plots as this helps illustrate important statistical properties. The box itself represents the range of values between quartiles (25 and 75 percentiles). The horizontal line in the middle of the box is the median. There are then vertical lines from the box to two horizontal lines, above and below it. These horizontal lines show the 5 and 95 percentiles respectively. Finally there are small circles. These are the outlying values that are outside of the 5 to 95 percentile range. The box and whisker plot gives a strong indication of typical performance, the variance observed, and information about the outliers. In each figure, each scheduling scheme is coloured coded according to the legend in the top right, with the information appearing in the order AMC+, AMC+S, BP, and BPS.

The percentage of LO-criticality jobs not executed ($\#JNE$) for each of the scheduling schemes, is shown for task sets with harmonic periods in Fig. 3 and Fig. 2.

We observe that in both configurations, the bailout protocol (BP) is effective in reducing the percentage of LO-criticality jobs that are not executed compared to the baseline AMC+ scheme. In configuration 2, increasing execution time budgets ($C(BU)$) by making use of static slack, leads to a roughly similar reduction in $\#JNE$ as BP, albeit with larger variability caused by variability in the amount of slack available in the various task sets.

Since the bailout protocol and making use of static slack are complementary techniques, the BPS scheme provides significantly better performance than AMC+S or BP. The results for configuration 1 (Fig. 2) are similar; however, since the system is more lightly loaded, there is more static slack available and so the improvements due to that technique are increased.

The values for JNE, LDM, and HDM are given, as a percentage of the total number of jobs of that type (i.e. HI- or LO-criticality) in the table below. Note that with the AMC, AMC+S, BP, and BPS schemes, there were no HI-criticality deadlines misses ($\#HDM$). Note also that there were very few LO-criticality deadline misses, only non-executed jobs. With basic FPPS, there were a small but highly significant number of HI-criticality jobs that missed their deadlines. The frequency of these deadline misses was 1.8×10^{-9} (i.e. $1.8 \times 10^{-7}\%$) in the second configuration. This may not be acceptable in a real system. The small but non-zero values for $\#JNE$ for FPPS are because under that policy, the simulation

abandoned any jobs that were released while the previous job of the same task was still active (having already missed its deadline), thus avoiding any possibility of a cascading overload. We note that in both configurations, on average the BPS scheme reduced the number of LO-criticality jobs not executed by over a factor of three compared to the AMC+ scheme.

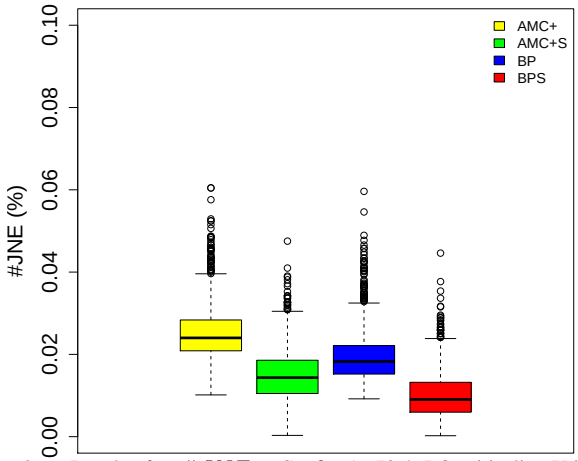

Fig. 2. Results for $\#JNE$ - Config 1: 70% LO-criticality Utilisation: Harmonic Periods

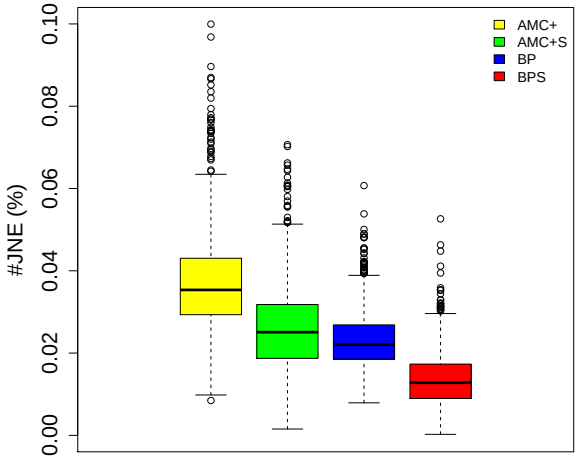

Fig. 3. Results for $\#JNE$ - Config 2: 90% LO-criticality Utilisation: Harmonic Periods

TABLE I
PERFORMANCE METRICS FOR SYSTEMS WITH HARMONIC PERIODS

U=0.7	AMC+	AMC+S	BP	BPS	FPPS
JNE	0.026%	0.015%	0.02%	0.01%	0%
LDM	3.0e-7%	1.6e-6%	3.0e-7%	2.5e-6%	0%
HDM	0%	0%	0%	0%	0%

U=0.9	AMC+	AMC+S	BP	BPS	FPPS
JNE	0.039%	0.027%	0.024%	0.014%	1.8e-7%
LDM	6.2e-7%	6.7e-7%	6.2e-7%	1.1e-6%	6.4e-7%
HDM	0%	0%	0%	0%	1.8e-7%

The bailout and AMC-based schemes sacrifice a small percentage of LO-criticality jobs in order to ensure that the deadlines of HI-criticality tasks are met (i.e. $\#HDM = 0$). Nearly all of these LO-criticality jobs are abandoned without execution ($\#JNE$); however, some can start but not meet their deadlines. All LO-criticality jobs that are started under these schemes are however completed. By comparison basic FPPS executes (very nearly) all jobs, but significantly both LO- *and* HI-criticality jobs can miss their deadlines. Note, we did not simulate the basic AMC scheme as that would have a very high value for $\#JNE$ as *all* LO-criticality jobs would be abandoned after the system first entered HI-criticality mode.

VII. Analysis of the Bailout Protocol

In this section, we prove important properties of the bailout protocol. For systems that are deemed schedulable by AMC-rtb analysis (see (1) and (2) in Section II-A), we claim that if the system is scheduled at runtime using FPPS and the bailout protocol, then:

P1. LO-criticality jobs that are released and complete in normal mode, with no intervening start of a bailout mode, are guaranteed to meet their deadlines.

P2. HI-criticality jobs released at any time are guaranteed to always meet their deadlines.

Stated otherwise, the AMC-rtb test is a sufficient schedulability test for MCS using FPPS and employing the bailout protocol. We note that: (i) LO-criticality tasks that are released during bailout or recovery modes are abandoned, and so effectively miss their deadlines. (ii) LO-criticality tasks that are dispatched in normal mode, but complete after the start of a bailout mode are not guaranteed to meet their deadlines.

We now prove, via a set of Lemmas and Theorems, Properties P1 and P2 of the bailout policy. Consider a system that is schedulable according to AMC-rtb analysis, and is scheduled at runtime using FPPS and the bailout protocol. Let S be some *bailout scenario*, corresponding to an arbitrary but valid sequence of job releases under which the system operates the bailout protocol due to one or more jobs of HI-criticality tasks exceeding their LO-criticality execution times. Let N be the *alternate normal scenario* for S. The alternate normal scenario N has its job releases at exactly the same times as scenario S; however, unlike scenario S where jobs may take arbitrary but valid execution times (i.e. $\leq C(LO)$ for LO-criticality tasks and $\leq C(HI)$ for HI-criticality tasks) all jobs in scenario N require exactly their LO-criticality execution times $C(LO)$, hence under scenario N, the system is always in normal (i.e. LO-criticality) mode and all deadlines are met. We will show that S behaves in an equivalent way to N.

For bailout scenario S, let $W^B(t,k)$ be the *total pending workload* due to jobs of priority k and higher (i.e. in $hep(k)$) that have execution outstanding at time t. Note that at the release of a job, we recognise its LO-criticality execution time up to a maximum of $C(LO)$ as contributing to the total pending workload; however, the additional HI-criticality execution time up to $(C(HI) - C(LO))$ is only considered as contributing to the total pending workload once the job has executed for $C(LO)$ without signalling completion. Let $W^N(t,k)$ be the total pending workload at priority k and higher at time t in the alternate normal scenario N. Further, let $[t_s, t_e)$ be an interval during which the system is in bailout mode in scenario S. Thus t_s is the start of a bailout mode

interval, and t_e the end, hence t_e is also the start of recovery mode.

Lemma 1. *For any arbitrary bailout scenario S, provided that at the end t_e of each bailout mode interval $[t_s, t_e)$, the total pending workload for every priority level j, is no greater than that for the alternate normal scenario N, i.e:*

$$\forall j \ W^B(t_e, j) \leq W^N(t_e, j) \tag{3}$$

then all jobs released and not immediately abandoned[5] at or after time t_e with deadlines prior to a subsequent transition to bailout mode are guaranteed to meet their deadlines.

Proof: Consider the bailout mode interval $[t_s, t_e)$, and an arbitrary job J_i released at or after time t_e with a deadline prior to any subsequent transition into bailout mode. As FPPS is used, the response time of job J_i depends *only* on (i) the total pending workload for priority i at time t_e i.e. $W^B(t, i)$ and (ii) the higher priority workload released at or after time t_e, but before the completion of job J_i. By the Lemma, (i) is no greater than in the alternate normal mode scenario. Further, (ii) is also no greater, since this workload comprises only jobs released after time t_e, all of which (by the Lemma) exhibit LO-criticality behaviour prior to the deadline of job J_i. (We note that the release times of these jobs are the same in both the bailout scenario and its alternate normal scenario; however, some releases may be abandoned in the bailout scenario due to the recovery mode behaviour immediately following t_e. This can only reduce the amount of workload compared to the alternate normal scenario). Hence the response time of job J_i is no greater than it would have been if the system had always executed in normal mode. Since job J_i is guaranteed to meet its deadline in normal mode, it is also guaranteed to meet its deadline in the bailout scenario with a transition into and out of bailout mode prior to its release ■

We now classify the mechanisms of the bailout protocol into three basic types of operation as follows. (Note the numbering below e.g. (ii) and (xi) refers to the clauses in the description of the bailout protocol given in section IV above)

- *BF increases*: (ii), (iii) and (xi): These mechanisms increase the bailout fund when a HI-criticality job executes for $C(LO)$ without signalling completion.
- *BF reductions (completion)*: (iv), (v), and (vi): These mechanisms involve a job at some priority k completing execution and reducing the bailout fund by any underspend with respect to the execution time that was previously accounted for.
- *BF reductions (abandonment)*: (vii): With this mechanism, a LO-criticality job released during the bailout interval, would have executed at some priority k, but is instead abandoned, donating its execution time to the bailout fund.

Note we do not consider mechanism (ix) further as at an idle instant the total pending workload at all priority levels is zero and hence there can be no impact on subsequent jobs. Mechanism (viii) indicates when the system exits bailout mode, which can only occur as a result of BF reductions due to either job completion or release.

Lemma 2. *Consider a bailout mode interval $[t_s, t_e)$ of an arbitrary bailout scenario S. Provided that at the start of the bailout mode interval, the total pending workload for every priority level j, is no greater for the bailout scenario (without yet recognising the additional execution time from the HI-criticality job that will cause the transition to bailout mode) than for its alternate normal mode scenario i.e. $\forall j \ W^B(t_s, j) \leq W^N(t_s, j)$ then at the end t_e of the bailout mode interval inequality (3) holds i.e. $\forall j \ W^B(t_e, j) \leq W^N(t_e, j)$.*

Proof: To prove the Lemma, we divide the bailout interval $[t_s, t_e)$ into a number of contiguous (non-overlapping) sub-intervals $[t_s, t_{e1})$, $[t_{s2}, t_{e2}) \ldots [t_{sn}, t_e)$. The end of each sub-interval is demarked by a BF reduction, due to either a job completion or release. We note there are no BF reduction operations within a sub-interval.

Initial step: At the start of the bailout interval $t = t_s$, by the Lemma $\forall j \ W^B(t_s, j) \leq W^N(t_s, j)$ without recognising the additional execution time from the HI-criticality job causing the transition to bailout mode. We now recognise this additional execution time $C(HI) - C(LO)$. Hence we have:

$$\forall j \ W^B(t, j) \leq W^N(t, j) + BF \tag{4}$$

where the initial value of BF is $C(HI) - C(LO)$.

First sub-interval: During the first sub-interval, HI-criticality tasks may execute for their $C(LO)$ without signalling completion and add to BF via mechanism (iii). Since BF is incremented by $C(HI) - C(LO)$ for each such job, it follows that (4) continues to hold. The sub-interval ends with a BF reduction operation.

Case 1: BF reduction (completion): Completion of a job at priority k at time t implies the following (since no workload can be pending at a higher priority than k otherwise the job at priority k would not be executing):

$$\forall j \in hp(k) \ W^B(t, j) = 0 \tag{5}$$

Further, as the job may have completed earlier than previously accounted for, either (a) via requiring less execution time than its $C(LO)$ value (mechanisms (iv) and (v)), or (b) via requiring less time for HI-criticality execution than was previously accounted for in BF (mechanism (vi)), then BF can be decremented by any underspend and the following holds. This is the case because (a) $W^N(t, j)$ includes workload that would have been pending if the job had required its full $C(LO)$ execution time, and (b) BF had previously been adjusted to include all of $C(HI) - C(LO)$ and we now know that not all of that execution time was required.

$$\forall j \in lep(k) \ W^B(t, j) \leq W^N(t, j) + BF \tag{6}$$

Case 2: BF reduction (abandonment): Recall that under mechanism (vii) a job of a LO-criticality task that is released in

[5]Recall that LO-criticality jobs released in recovery mode are immediately abandoned.

Bailout mode is abandoned (not started) and at the time t that this job would otherwise have started to execute, it donates its budget of $C(LO)$ to the bailout fund ($BF = BF - C(LO)$. Donation of this budget implies (5), since the fact that the job would have executed at time t means that there can be no pending higher priority jobs (workload) at that time. Further, the total pending workload at priorities lower than k is reduced by the execution time $C(LO)$ of the abandoned job. Hence the value of BF is reduced according to mechanism (vii), yet (6) still holds, since $W^N(t, j)$ includes $C(LO)$ for the abandoned job.

Subsequent sub-intervals: All subsequent sub-intervals in the bailout mode interval may be considered in the same way as the first sub-interval, thus (5) and (6) continue to hold at the end of each sub-interval, where k is the priority of the task that completes its execution or would have started to execute but has instead been abandoned. It follows that the bailout interval ends with some BF reduction operation due to a task at priority k, and at that time we have:

$$\forall j \in hp(k)\ W^B(t_e, j) = 0 \tag{7}$$

and

$$\forall j \in lep(k)\ W^B(t_e, j) \leq W^N(t_e, j) + BF \tag{8}$$

with $BF = 0$. Since $W^N(t_e, j) \geq 0$, it follows that $\forall j\ W^B(t_e, j) \leq W^N(t_e, j)$ ■

Theorem 1. *All jobs that are released (and not immediately abandoned because they are LO-criticality jobs released in recovery mode) and have their deadlines within an interval that does not include bailout mode (but may comprise recovery and normal mode) are guaranteed to meet their deadlines provided that the system is schedulable according to AMC-rtb analysis and is scheduled at runtime using FPPS and the bailout protocol.*

Proof: We consider all of the intervals in an arbitrary bailout scenario S, which has an alternate normal scenario N that is schedulable under FPPS. Since the system starts in normal mode, during the first interval in S before entering bailout mode, all jobs require at most their LO-criticality execution time $C(LO)$ and hence the theorem trivially holds for those jobs. Since the system starts in normal mode, at the start of the first bailout interval, and before recognising the additional execution time required by the job that causes the transition to bailout mode, we have $\forall j\ W^B(t_s, j) \leq W^N(t_s, j)$. From Lemma 2 it follows that $\forall j\ W^B(t_e, j) \leq W^N(t_e, j)$ holds at the end t_e of the first bailout interval. From Lemma 1, the Theorem therefore holds for the second interval between bailout modes. Further, since during this second interval, jobs only exhibit their LO-criticality execution times, it follows that at the start of the next bailout mode $\forall j\ W^B(t_s, j) \leq W^N(t_s, j)$. again holds. Induction over all of the bailout modes and intervals between them is sufficient to show that all jobs that are released and have their deadlines within a single interval between bailout modes are schedulable ■

Theorem 1 shows that all jobs released and not immediately abandoned in recovery or normal mode with deadlines prior to the start of the next bailout mode are guaranteed to meet their deadlines provided that the system is schedulable according to AMC-rtb analysis. This encompasses Property P1 – LO-criticality jobs that are released and complete in normal mode, with no intervening start of a bailout mode are guaranteed to meet their deadlines under the bailout protocol.

Theorem 2. *All jobs of HI-criticality tasks are guaranteed to meet their deadlines provided that the system is schedulable according to AMC-rtb analysis and is scheduled at runtime using FPPS and the bailout protocol.*

Proof: Theorem 1 suffices to show that any job of a HI-criticality task that is released in a recovery or normal mode interval and has a deadline prior to the start of the next bailout mode is schedulable. We are therefore left with two further cases to consider.

Case 1: A HI-criticality job that is released in a recovery mode or normal mode interval and completes in the next bailout mode interval or the recovery mode interval that follows it. The proof of Theorem 1 shows that $\forall j\, W^B(t_e, j) \leq W^N(t_e, j)$ holds at the end of any bailout mode interval. Hence any job that is released in recovery mode or normal mode is subject to interference from the time of its release to the start of the next bailout mode that is no greater than if the system operated continually in normal mode. The maximum possible time from the release of the job until it either completes or the next bailout mode is entered is therefore $R(LO)$ (see AMC-rtb analysis, i.e. (1) and (2) in Section II-A). This holds since in normal mode, the job must have executed for $C(LO)$ by $R(LO)$ after its release, and will hence trigger a transition to bailout mode if it has not completed by then. The maximum amount of interference from higher priority LO-criticality jobs is therefore limited to at most those releases within an interval of length $R(LO)$, as per the AMC-rtb analysis. Further, since that analysis assumes interference of $C(HI)$ from all releases of higher priority HI-criticality tasks, the response time of the job must be bounded by the worst-case response time computed by AMC-rtb.

Case 2: A HI-criticality job that is released in a bailout mode interval and completes in that interval or the recovery mode interval that follows it. Such a job cannot be subject to more interference than considered in Case 1, and so is also schedulable. No job of a HI-criticality task that is released in a recovery mode, normal mode, or bailout mode interval, can complete after the end of the next recovery mode interval, since that recovery mode would by definition extend until such completion. Hence Cases 1 and 2 cover all further possibilities for the release and completion of HI-criticality jobs ■

We note that the presence of recovery mode is necessary to ensure that HI-criticality jobs always meet their deadlines. Without the recovery mode, i.e. permitting LO-criticality jobs to be released as soon as bailout mode ends, would provide scope for increased interference from high priority LO-criticality tasks beyond that considered by the AMC-rtb

analysis. Effectively the interval of LO-criticality interference on a high criticality task would be split into two parts and as $\lceil a \rceil + \lceil b \rceil \geq \lceil a + b \rceil$ this interference may then be larger. Finally, we note that despite the workload relationship given by (3), there is no guarantee that LO-criticality tasks that are released in normal mode and complete in a subsequent normal mode after a transition through bailout mode will meet their deadlines (as illustrated in the example shown in Figure 1).

VIII. Conclusions

In a mixed criticality system (MCS) most criticality levels will require that deadlines are always met. However, it is necessary to design systems that behave robustly in situations where software is behaving in an unexpected manner – particularly if estimated executions times are exceeded. A number of theoretical advances have been made in terms of scheduling MCSs. In this paper we move the theory closer to industrial application. In particular we consider how to minimise the consequences of partial (temporal) failures, and how to restore service for LO-criticality tasks while still guaranteeing the HI-criticality ones.

The paper introduces a bailout protocol that allows overrun of HI-criticality jobs to be accommodated by the non-execution of LO-criticality jobs. The number of non-executions is however kept to a minimum. The bailout protocol is described by analogy to the banking system; HI-criticality tasks cannot fail, loans are taken out by HI-criticality tasks and repaid by LO-criticality tasks. The bailout protocol is orthogonal to existing mechanisms based on offline slack calculation which can be employed to further improve performance.

Finally, we used schedulability analysis techniques to show that the bailout protocol has the same level of guarantee as the best previously published approach (AMC), and a scenario-based evaluation framework to show that it provides significantly improved performance for LO-criticality tasks. Specifically, whereas AMC abandons all LO-criticality tasks following a transition to HI-criticality mode, the bailout protocol can be shown to seldom miss any LO-criticality deadlines for realistic assumptions about task execution times.

Acknowledgements

The research described in this paper was funded, in part, by the ESPRC grant, MCC (EP/K011626/1) and the EU FP7 IP PROXIMA (611085). EPSRC Research Data Management: No new primary data was created during this study.

References

[1] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[2] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] S.K. Baruah. Semantic-preserving implementation of multirate mixed criticality synchronous programs. In *Proc. of Real-Time Networks and Systems (RTNS)*, 2012.

[4] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hlinený and A.ín Kucera, editors, *Proc. of the 35th International Symposium on the Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[5] S.K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152, 2012.

[6] S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[7] S.K. Baruah and G. Fohler. Certification-cognizant time-triggered scheduling of mixed-criticality systems. In *Proc. of IEEE Real-time Systems Symposium*, pages 3–12, December 2011.

[8] S.K. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *ECRTS*, pages 147–155, 2008.

[9] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems Journal*, 25(1):5–37, 2003.

[10] E. Bini and G.C. Buttazzo. Measuring the performance of schedulability tests. *Journal of Real-Time Systems*, 30(1-2):129–154, 2005.

[11] E. Bini, M. Di Natale, and G.C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. In *Proc. ECRTS*, pages 13–22, 2006.

[12] A. Burns. System mode changes - general and criticality-based. In *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*, pages 3–8, Dec 2014.

[13] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. of 1st International Workshop on Mixed Criticality Systems (WMC)*, pages 1–6, 2013.

[14] A. Burns and R. Davis. Mixed criticality systems - a review. Technical report, Department of Computer Science, University of York, 2014.

[15] A. Burns and R.I.Davis. Adaptive mixed criticality scheduling with deferred preemption. In *Proc. of Real-Time Systems Symposium (RTSS)*, Dec 2014.

[16] R.I. Davis. *On exploiting spare capacity in hard real-time systems*. PhD thesis, University of York, UK, 1995.

[17] R.I. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proc. of IEEE Real-Time Systems Symposium*, pages 3–14, 2007.

[18] F. Dorin, L George, P. Thierry, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Journal of Real-Time Journal*, 46(3):305–331, 2010.

[19] P. Ekberg and W. Yi. Bounding and shaping the demand of mixed-criticality sporadic sprodic tasks. In *ECRTS*, pages 135–144, 2012.

[20] T. Fleming and A. Burns. Incorporating the notion of importance into mixed criticality systems. In *Proc. of 2nd Workshop on Mixed Criticality Systems (WMC)*, pages 33–38, Dec 2014.

[21] P. Graydon and I. Bate. Safety assurance driven problem formulation for mixed criticality scheduling. In *Proc. of 1st Workshop on Mixed Criticality Systems (WMC)*, 2013.

[22] P. Graydon and I. Bate. Realistic safety cases for the timing of systems. *The Computer Journal*, 57(5):759–774, 2014.

[23] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling of certifiable mixed-criticality sporadic task systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, pages 13–23, 2011.

[24] H. Pengcheng, P. Kumar, N. Stoimenov, and L. Thiele. Interference constraint graph a new specification for mixed-criticality systems. In *Proc. Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2013.

[25] S. Punnekkat, R. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Proc. of the Conference of Advances in Computing Science - ASIAN '97*, pages 72–82. Springer, 1997.

[26] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, , and E. Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In *Proc. of Real-Time Networks and Systems (RTNS)*, 2013.

[27] F. Santy, P. Richard, M. Richard, and J. Goossens. Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP. In *Proc. Euromicro Conference on Real-Time Systems*, pages 155–165, 2012.

[28] H. Su and D. Zhu. An elastic mixed-criticality task model and its scheduling algorithm. In *Proc. of the Conference on Design, Automation and Test in Europe*, DATE, pages 147–152, 2013.

[29] K. Tindell and A Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politecnica de Madrid, 1996.

[30] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.