An Enhanced Bailout Protocol for Mixed Criticality Embedded Software

Iain Bate[1], Alan Burns[1] and Robert I. Davis[1,2]
[1]Department of Computer Science, University of York, York, UK
[2]INRIA, France
{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 software but minimises the negative impact on lower criticality software via a timely return to normal operation. We show how the bailout protocol can be integrated with existing techniques, utilising both offline slack and online gain-time to further improve performance. Static analysis is provided for schedulability guarantees, while scenario-based evaluation via simulation is used to explore the effectiveness of the protocol.

Index Terms—Real-Time Systems, Mixed Criticality, Fixed Priority Scheduling, Mode Changes.

Preliminary publication

This paper extends initial research into a bailout protocol for mixed criticality systems presented at ECRTS 2015 [1]. The additional material includes: An extended worked example illustrating, in figures 1 and 2, the behaviour of the bailout protocol as compared to the baseline Adaptive Mixed Criticality (AMC) scheduling policy. Extensions to reclaim gain-time, which becomes available when a task executes for less than its worst-case execution time budget. Integration of this technique with the bailout protocol is described in Section 5. An extended scenario based evaluation, in Section 6. This examines the benefits of gain-time reclamation in conjunction with the baseline Adaptive Mixed Criticality (AMC) scheduling policy and with the bailout protocol. The evaluation also covers additional metrics including the number of times that the system has to go into a HI-criticality mode, and the amount of time spent in that mode. It is also extended to show how a variety of different factors impact the performance of the bailout protocol and other scheduling policies, thus showing the broad range of circumstances in which the protocol is effective. Finally, in Section 8 we show how the bailout protocol can be adapted to systems with multiple criticality levels.

1 Introduction

An increasingly important trend in the design of real-time and embedded software systems is the integration of components with different levels of criticality onto a common hardware platform. 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 [2]. 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. 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 [3], a standard model has emerged (see for example [4]–[9]). For dual criticality systems (with the two levels: HI-criticality and LO-criticality) this standard model has the following properties:

- A mixed criticality system is defined to execute in one of two modes: a *normal* mode and a *HI-criticality* mode.
- All software is structured as concurrently executing *tasks* that are scheduled by a dependable RTOS (Real-Time Operating System) supporting fixed priority preemptive scheduling.
- 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 $C(HI) \geq C(LO)$ [3].
- The system starts in the normal mode, and remains in that mode as long as all jobs execute within their LO-criticality execution times ($C(LO)$).
- If any HI-criticality job executes for its $C(LO)$ execution time without completing then the system immediately degrades to the HI-criticality mode.
- 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.
- As the system moves to the HI-criticality mode all LO-criticality tasks are abandoned. No further LO-criticality jobs are executed.
- The system remains in the HI-criticality mode.

The movement from normal mode to HI-criticality mode is a form of graceful degradation. Following a timing anomaly only the HI-criticality tasks are guaranteed to meet their deadlines.

The motivation for the standard model having two values for the Worst-Case Execution Time (WCET) [3], [10] is taken from either of two situations often seen in industrial practice [11]. 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] [12]. 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 sound [11], 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 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 [2]. 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 normal 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.

The main contribution of this paper is the introduction of the *Bailout Protocol* 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. While the bailout protocol allows LO-criticality jobs to be dropped, 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 in the normal mode of execution whereby it can again guarantee all tasks. (Note many forms of analysis actually reduce their complexity by assuming all released jobs will complete). The bailout protocol aims to restore the normal mode as soon as possible following an interval of HI-criticality only activity, and so minimise the number of LO-criticality jobs that miss their deadlines or are not executed. The bailout protocol thus reduces the amount of time spent in the HI-criticality mode. In addition, we show how the protocol can be complemented by techniques based on gain-time reclamation [13] and slack stealing [14], [15] to further reduce both the number of times the system enters HI-criticality mode and the amount of time that it spends in that mode.

To comply with the requirements of MCS, scheduling policies and protocols must ensure that HI-criticality tasks always meet their deadlines, and that all tasks meet their deadlines when the system is in normal mode. Schedulability analysis provides the answers to these questions. Beyond such compliance, the relative effectiveness of the different protocols is judged on the basis of criteria such as the number of times the system enters the HI-criticality mode, the amount of time spent in that mode, and the number of LO-criticality jobs that either miss their deadlines or are abandoned. Scenario-based assessment using large-scale simulations provides information about these metrics although the results obtained are only valid for the range of scenarios explored.

The remainder of the paper is organised as follows. In Section 2, we discuss related work, introduce the formal system model used in this paper, and recapitulate on the basic schedulability analysis for MCS which we build upon. Approaches to degraded service are considered in Section 3. In Section 4, we define the *bailout protocol* for MCSs, and in Section 5 show how it can be integrated with techniques that make use of spare capacity that is either available both off-line, or becomes available at runtime, to improve performance. A key aspect of this paper is the evaluation of MCS protocols via scenario-based simulation; this is addressed in Section 6. Analysis for the bailout protocol is given in Section 7. An extension of the protocol to more than two criticality levels is outlined in Section 8. Finally, Section 9 concludes with a summary and a discussion of future work.

2 Background

Background material on MCS research can be obtained from the following sources [3]–[6], [8], [16]–[18]. An ongoing survey of MCS research by [10] is available from the MCC (Mixed Criticality Systems on Many-core Platforms) project website[3]. We note that while mixed criticality behaviour has some similarities to traditional mode changes, there are also significant differences [2], [19]. These include the mode change being driven by a particular temporal rather than functional behaviour, permitting a more specific schedulability analysis.

2.1 System Model and Assumptions

In this paper, we are interested in the Fixed Priority Preemptive Scheduling (FPPS) of a mixed criticality system comprising a static

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 significant system failure.

3. http://www.cs.york.ac.uk/research/research-groups/rts/mcc/.

set of n sporadic tasks which execute on a single processor. We assume without loss of generality that each task τ_i has a unique priority, given by its index. Thus task τ_1 has the highest priority and task τ_n the lowest. We assume a discrete time model in which all task parameters are given as integers. Each task, τ_i, is defined by its period (or minimum arrival interval), relative deadline, worst-case execution time, and level of criticality (defined by the system engineer responsible for the entire system): (T_i, D_i, C_i, L_i). We restrict our attention to constrained-deadline systems in which $D_i \leq T_i$ for all tasks. Further, we assume that the processor is the only resource that is shared by the tasks, and that the overheads due to the operation of the scheduler and context switch costs can be bounded by a constant, and hence included within the worst-case execution times attributed to each task.

In a mixed criticality system, further information is needed in order to undertake schedulability analysis. In general a task is defined by: (T, D, $\vec{C}$, L), where $\vec{C}$ is a vector of values – one per criticality level, with the constraint $L1 > L2 \Rightarrow C(L1) \geq C(L2)$ for any two criticality levels $L1$ and $L2$. In this paper we are mainly concerned with dual criticality systems, with criticality levels LO and HI (where LO < HI). Thus each LO-criticality task has a single worst-case execution time estimate $C(LO)$, while each HI-criticality task has two worst-case execution time estimates $C(LO)$ and $C(HI)$ with $C(HI) \geq C(LO)$.

2.2 Current Scheduling Analysis and its Limitations

Although the standard model of mixed criticality system behaviour 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 [16], [20], [21] 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 by [5] first computes the worst-case response times for all tasks in the normal 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:

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

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-criticality 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 normal mode when the system experiences an *idle instant*[4]. This is a well-known protocol for controlling mode changes [22]. 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.

3 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.

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 [19].

Various forms of degraded service have been proposed for LO-criticality tasks in the literature: Run all tasks, but extend their periods and/or deadlines – sometimes called the elastic task model [23]. Run all tasks but reduce the executions times of LO-criticality tasks (i.e. $C(HI) \leq C(LO)$ for these tasks) [24] – perhaps by switching to simpler version of the software. Drop jobs from a specific subset of tasks [25], [26] or skip s_i in every m_i jobs of each task [27].

4. An idle instant is an instant in time at which there are no jobs with execution time outstanding that were released prior to that time.

In comparison with the bailout protocol presented in this paper, the above methods prescribe specific changes to the behaviour of LO-criticality tasks either increases in their periods, decreases in their execution times (and hence the need for different versions of the software), or dropping specific jobs e.g. 1 job in every 3. The bailout protocol on the other hand does not change the primary behaviour of LO-criticality tasks, but rather focuses on re-instating them fully as quickly as possible. This has less impact on overall schedulability, since similar to AMC, there are no guarantees for LO-criticality tasks in the HI-criticality / bailout mode. In systems where transitions to HI-criticality mode are rare, and some missed jobs of LO-criticality tasks can be tolerated, then the bailout protocol may provide an effective solution. In systems where LO-criticality tasks must continue to provide some level of guaranteed service even when the system is in its degraded mode, then other methods need to be used.

We note that the approach taken by the bailout protocol is orthogonal to those of job dropping [26] and weakly-hard guarantees for LO-criticality tasks proposed in [27], hence it is possible that the different techniques could be combined; such work is however beyond the scope of this paper.

An orthogonal approach to improving the overall service for LO-criticality tasks was adopted by Santy et al. [7]. 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 [24] using Robust Priority Assignment techniques [28] 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 normal mode following an interval of HI-criticality behaviour. As mentioned previously, this can be achieved by waiting for an idle instant. Santy et al. [7] explored this approach, and also developed a protocol for multiprocessor scheduling where there may be no idle instant across all processors [29]. Further work by Ren et al. [30] focused on partitioned multiprocessor scheduling. Here, each HI-criticality task is associated with a group of LO-criticality tasks. Thus the overrun of the HI-criticality task can only impinge on the execution of LO-criticality tasks in the same task group. Task groups are scheduled according to EDF, with servers used within each group to ensure mixed criticality guarantees.

4 The Bailout Protocol

We now describe the Bailout Protocol assuming two levels of criticality in the system software.

4.1 Protocol, modes, and mechanisms

At run-time, dual criticality systems are typically defined to be in one of two modes: *normal mode* and *HI-criticality mode*; however, these terms can be confusing. With the bailout protocol, we defined three modes: *normal mode*, *bailout mode* and *recovery mode*. Normal mode is as defined above. 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, then 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: $BF = BF - (C(LO) - e)$. Note, 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 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 [5].

(ix) If during bailout mode, an idle instant occurs, then an immediate transition is made to normal mode, and BF is reset to zero [6].

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) amounting to only a few instructions, requiring only $O(1)$ time, and incorporated into existing RTOS code for context switching or execution time budget monitoring. All actions take place at the release or

5. 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 7.4 in Section 7 and the discussion that follows it.

6. It can easily happen that $BF > 0$ when the processor becomes idle, for example if a HI-criticality job exceeds its $C(LO)$ and when it completes there are no other jobs with remaining execution.

completion of a job, which are well defined RTOS operations in FPPS, or when a job executes for $C(LO)$ without signalling completion. In the case of a LO-criticality task, the action required in the latter case corresponds to execution time budget enforcement, as needed in any high integrity implementation whether AMC or the bailout protocol were being employed or not. Such an overrun may be detected via a timer interrupt and the job aborted. In the case of a HI-criticality job executing for $C(LO)$ without signalling completion, then the action required is to change to HI-criticality mode, preventing further releases of LO-criticality jobs. Since the HI-criticality job continues to execute, such a mode change may be soundly deferred until the next scheduling point (i.e. job release or completion), and so no timer interrupt is required; rather only execution time monitoring is needed. (This is the case with both AMC and the bailout protocol). We note that the bailout protocol does not change task priorities, nor introduce any additional context switches which are not also present under basic FPPS.

4.2 Example

We now give an example illustrating the behaviour of the bailout protocol. This example includes five tasks: τ_1, τ_2, and τ_5 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 table 1 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.

TABLE 1
Example task parameters

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
τ_5	LO	12	-	92	92	92	-

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 (since $C(LO) = 4$). 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 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 normal behaviour before they complete.

We note that without the bailout protocol, this system would not revert to normal mode until an idle instant occurred, hence the third jobs of both tasks τ_1 and τ_2 would not be executed, and the system would not return to normal mode until time $t = 54$. This is illustrated in Figure 2 which shows the schedule for the same behaviour under AMC. This example serves to illustrate the advantages of the bailout protocol, fewer jobs LO-criticality jobs are dropped, and the system returns to normal mode 14 time units after the HI-criticality behaviour is detected, rather than 38 time units after.

4.3 Discussion

A more general comparison can also be made between the bailout protocol and AMC+. Recall that AMC+ relies on the simple *idle-instant* protocol [22] to revert to normal mode. Since the bailout protocol also returns to normal mode on an idle instant (operation (ix) in bailout mode and potentially also operation (xii) in recovery mode), but can also make earlier transitions back to normal mode, it dominates AMC+ in terms of the time taken between entering HI-criticality / bailout mode and returning to normal mode. Stated otherwise, the bailout protocol takes no longer than AMC+ to return to normal mode, assuming the same initial pattern of task executions.

In the extreme case where all jobs take their maximum execution time (either $C(LO)$ or $C(HI)$) then the interval needed to recover back to normal mode can still be no greater with the bailout protocol; it may however be shorter due to the bailout fund being reduced by the budgets of abandoned LO-criticality jobs (operation (vii) in the protocol). In the worst-case, when there are also no abandoned LO-criticality jobs to reduce the bailout fund, then the interval needed to recover back to normal mode is the same as that for AMC.

It is also interesting to consider, for a schedulable system, the longest possible time that may elapse between entering HI-criticality / bailout mode and the transition back to normal mode. This is the same for both the bailout protocol and AMC+. For a schedulable system, in the worst-case, both must wait for an idle instant. We now derive an upper bound A on the length of time that can elapse before such an idle instant occurs. We pessimistically assume the worst-case possible behaviour of both HI- and LO-criticality tasks. Each LO-criticality task may give rise to a single job which executes during the HI-criticality mode (such jobs must have been released just before the transition to the HI-criticality mode). In the case of the HI-criticality tasks, we assume (pessimistically) that at the transition each task has an outstanding job that has been delayed from executing for as long as possible and now requires its $C(HI)$ execution time before completing at its deadline. Subsequent jobs are then released as soon as possible also requiring $C(HI)$. This scenario is captured by the following recurrence relation:

$$A = \sum_{\forall j \in \mathbf{aH}} \left\lceil \frac{L + (D_j - C_j(HI))}{T_j} \right\rceil C_j(HI) + \sum_{\forall k \in \mathbf{aL}} C_k(LO) \tag{3}$$

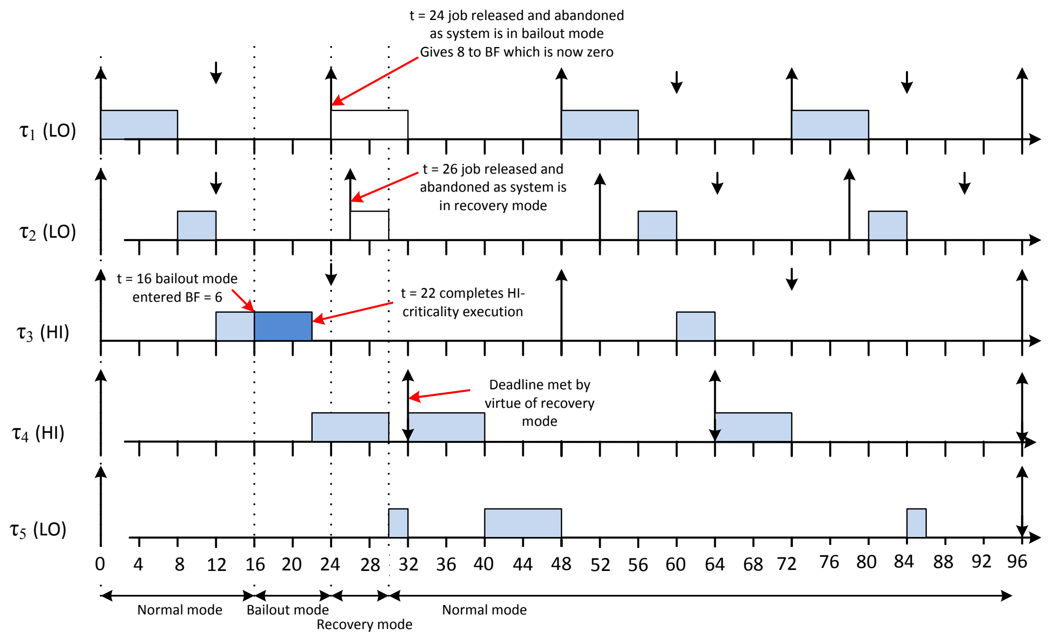
t = 24 job released and abandoned as system is in bailout mode Gives 8 to BF which is now zero
t = 26 job released and abandoned as system is in recovery mode
t = 16 bailout mode entered BF = 6
t = 22 completes HI-criticality execution
Deadline met by virtue of recovery mode
τ1 (LO)
τ2 (LO)
τ3 (HI)
τ4 (HI)
τ5 (LO)
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96
Normal mode
Bailout mode
Recovery mode
Normal mode

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

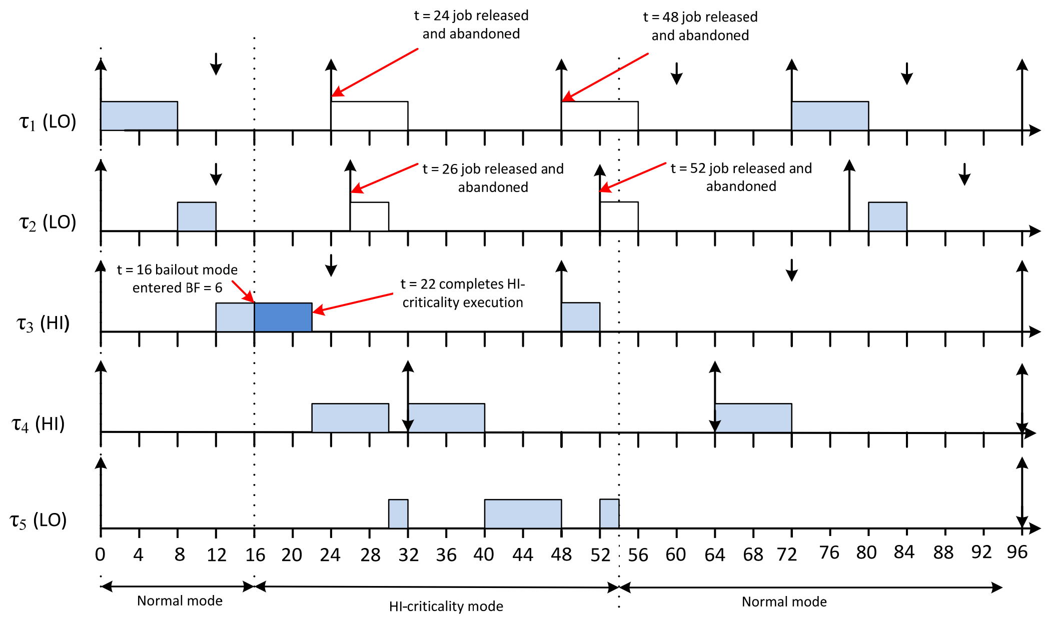
t = 24 job released and abandoned
t = 48 job released and abandoned
t = 26 job released and abandoned
t = 52 job released and abandoned
t = 16 bailout mode entered BF = 6
t = 22 completes HI-criticality execution
τ1 (LO)
τ2 (LO)
τ3 (HI)
τ4 (HI)
τ5 (LO)
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88 92 96
Normal mode
HI-criticality mode
Normal mode

Fig. 2. Example showing the operation of AMC, including normal and HI-criticality modes.

where aL is the set of all LO-criticality tasks, and aH is the set of all HI-criticality tasks. Iteration starts with an initial value of $A = \sum_{\forall j \in \mathbf{aH}} C_j(HI) + \sum_{\forall k \in \mathbf{aL}} C_k(LO)$ and ends on convergence, which is guaranteed since the utilization of HI-criticality tasks computed using their $C(HI)$ values cannot exceed 1.

Note, increasing execution time budgets, as discussed in the next section, may increase the maximum time required to return to normal mode due to the increase in LO-criticality execution which may take place after the transition to the HI-criticality mode. We note that while the system is guaranteed to return to LO-criticality mode after an interval of at most A. Such a guarantee is not particularly useful, since further HI-criticality behaviour may force an almost immediate return to the HI-criticality mode.

5 Improvements

In this section we describe two methods, one offline and the other online, which are complementary to the bailout protocol. These methods help to reduce 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.

5.1 Slack Time: Increasing Execution Time Budgets

The offline method was introduced by Santy et al. [7] and further refined by Burns and Baruah [24]. It uses sensitivity analysis [31], [32] to explore by how much execution budgets, normally set to $C(LO)$ values, can be changed without making the system unschedulable, effectively making use of the available slack in the system [33]. Intuitively, this method is compatible with the bailout protocol, since it effectively increases the execution time budgets, normally based on $C(LO)$ values, while ensuring that the system remains provably schedulable. (Note, it is important here to distinguish between the worst-case execution time estimates e.g. $C(LO)$ obtained for the software, and the potentially larger values with a greater engineering margin, that can be used at runtime as execution time budgets).

The specific method we use is as follows: First, we increase the execution time budgets 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 (3)). 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 WCET estimates 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 [34] is used along with the single task schedulability test (i.e. (1) and (3)) 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 i.e. $C(HI)$ for HI-criticality tasks, and $C(LO)$ for LO-criticality tasks), 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.

We note that 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 $C(LO)$.

5.2 Gain Time

Gain Time refers to the difference between the execution time actually used by a job and the execution time budget that it was allocated. We assume that jobs have an initial execution time budget given by $c = C(BU)$, where $C(BU)$ is the execution budget for the task, either $C(LO)$ or derived as described in section 5.1 above. At runtime, it is likely that many jobs will complete in less than their execution time budgets. A number of mechanisms exist that can make this gain time available for use by other jobs [33], [35], [36], while ensuring that schedulability is unaffected.

The method we use comes from the Extended Priority Exchange algorithm [35] and operates in conjunction with the bailout protocol, *only* in normal mode. In normal mode, whenever a job completes in an execution time e, which is less than its budget (i.e. $e < c$), then the gain time $c - e$ is added to the execution time budget of the next lower priority active job (i.e. the next job in the run queue). This has no effect on schedulability, since the higher priority job (running first) could have legitimately executed for this gain time without any deadlines being missed. Passing gain time from one job to another in this way makes it less likely that jobs requiring more execution time than expected will actually exceed their execution time budgets, in turn making the system more robust to overruns (i.e. jobs exceeding $C(LO)$) and less likely to enter bailout mode. We denote this scheme as BPG and BPGS if static slack is used as well as gain time. We note that the gain time mechanism can be employed with AMC+ and AMC+S, in which case (unlike with the bailout protocol) it can operate in both normal and HI-criticality modes, but is only beneficial in the normal mode.

The gain time mechanism has a low overhead with $O(1)$ budget accounting at the completion of each job. This mechanism could potentially be improved by representing gain time in terms of the capacity of servers running at different priorities, with tasks, (including the idle task) first using spare capacity from the highest priority server with available capacity. Such an approach would better preserve any gain time generated. For example if the processor became idle, then spare capacity would not simply be discarded, but instead it would be gradually idled away, hence even after an idle period, tasks could still potentially benefit from previously generated gain time. Although theoretically superior, such an approach would require more complex runtime support than the standard mechanism which can be simply implemented by passing the remaining execution time budget at completion to the task at the head of the ready queue (the next task to run). In this paper, we therefore explore only the standard mechanism. We also note that in bailout mode, the gain time mechanism is not used, since the bailout protocol effectively makes use of gain time to hasten recovery.

6 Scenario-Based Evaluation

In this section, we present a scenario-based evaluation of the performance of the bailout protocol using an experimental framework / simulation. This is a commonly-used approach to evaluating real-time systems [37]–[39] when it is not practical to do effective 'what-if' analysis by other means. 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 either do not execute or miss their deadlines. Our evaluation aims to provide an understanding of how the different scheduling schemes (AMC+, AMC+S, AMC+SG, BP, BPS, BPSG) meet the needs of mixed-criticality systems. The first step in this process is the selection of evaluation metrics.

6.1 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) *Number 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 number of LO-criticality jobs that are abandoned.
3) *LO-criticality Deadline Misses* (LDM): The number of LO-criticality jobs that are executed, but miss their deadlines.
4) *Time in HI-criticality mode* (TiH) - How much time is spent in the HI-criticality mode (equates to bailout and recovery modes for the schemes using the bailout protocol).
5) *Number of times in HI-criticality mode* (NiH) - How many times the system enters the HI-criticality mode (equates to bailout and recovery modes for the schemes using the bailout protocol).

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 proportion of LO-criticality jobs that fail to meet their deadlines, either by missing their deadlines (LDM) or not being executed (JNE). This is the main metric that we explore via scenario based assessment. Although the simulator computes LDM, this number is far smaller than JNE, we therefore do not separately show LDM in the graphs presented in subsequent sections.

6.2 Experimental Framework

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

6.2.1 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 4).
3) *Bailout Protocol - Slack (BPS)* – The bailout protocol enhanced by offline increases in execution time budgets making use of static slack (section 5.1).
4) *Bailout Protocol - Slack and Gain Time (BPSG)* – The bailout protocol enhanced by both increasing execution budgets offline, and via runtime reclamation of gain time, as described in section 5.2.
5) *Adaptive Mixed Criticality - (AMC+)* – The standard AMC scheme [5] (section 2.2), enhanced so the system resumes LO-criticality execution after an idle instant.
6) *Adaptive Mixed Criticality - Slack (AMC+S)* – The AMC+ scheme, enhanced by offline increases in execution time budgets making use of static slack (section 5.1).
7) *Adaptive Mixed Criticality - Slack and Gain Time (AMC+SG)* – The AMC+ scheme enhanced by both increasing execution budgets offline, and via runtime reclamation of gain time (section 5.2).

6.2.2 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 in one of two ways. *Harmonic periods* were 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 [40]. *Non-harmonic periods* were chosen at random according to a log-uniform distribution corresponding to a range 10ms to 1 second (rounded to 0.1ms). In both cases, deadlines were set equal to periods.
2) *Execution Times* - LO-criticality utilisation $U(LO)$ values for each task where determined according to the Uunifast algorithm [41], thus ensuring an unbiased distribution of values that sum to the target utilisation for the system (Default 80%). 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 (see below). Finally, best case execution times (BCET) were chosen at random between 80% and 100% of C(LO). (This small variation is representative of code from Safety Critical Systems).
3) *Criticality Factor* (CF) - Determines the ratio of HI-criticality to LO-criticality execution times $C(HI) = CF.C(LO)$. The default value used was $CF = 2.0$ with CF varied from 1.25 to 2.5 in specific experiments aimed at illustrating the effect that the ratio of HI-criticality to LO-criticality execution time has on the performance of the various scheduling schemes.
4) *Criticality Probability* (CP) - Tasks were randomly chosen to be either HI- or LO-criticality, with a probability of CP of being HI-criticality. The default value used was $CP = 0.5$ with CP varied from 0.3 to 0.7 in specific experiments aimed at illustrating the effect that the proportion of HI-criticality tasks has on performance.
5) *Failure Probability* (FP) - In the simulation, jobs of HI-criticality tasks had a probability of FP of exceeding their $C(LO)$ execution time. The default value used was $FP = 10^{-4}$ with FP varied from 10^{-5} to 1 in specific experiments experiments aimed at illustrating the effect that higher failure probabilities have on performance.

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

6.2.3 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. We therefore first examined in detail a baseline configuration using the default parameter settings described above, and then conducted a series of experiments using a variety of other configurations where each parameter was varied over a representative range with the others held constant. The baseline configuration used had 80% LO-criticality utilisation, with $CF = 2.0$ and $CP = 0.5$; meaning that many of the task sets had an overall utilisation exceeding 100% when accounting for HI-criticality execution times. This illustrates one of the benefits of some of the mixed-criticality scheduling approaches in that task sets with overall utilisation exceeding 100% are schedulable as LO-criticality tasks do not have to be executed all of the time [5].

In all of the configurations examined in our experiments, we required that the task sets chosen had at least one task that was unschedulable according to exact analysis of FPPS [42], but were schedulable according to AMC-rtb [5]. 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 $CP \pm 10\%$ multiplied by the total number of tasks (recall that each individual task had a probability of CP of being HI-criticality).

6.2.4 Simulation

Our experiments covered 100 task sets for each of the configurations considered. For each scheduling scheme, we simulated the runtime behaviour of each task set, starting with a different random seed. (The same random seeds were used for each of the scheduling schemes to ensure a precise like-for-like comparison). 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 FP (default 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 FP used to determine if HI-criticality behaviour would be exhibited was deliberately set to a relatively high value by default as we wanted to stress the system behaviour (later experiments explored other values). 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[7].

Note for the schemes making use of statically available slack, the $C(BU)$ parameters were computed via offline sensitivity analysis, as described in Section 5.1, 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.

6.3 Baseline Evaluation Results

Our baseline 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, AMC+SG, BP, BPS, and BPSG.

Figure 3 shows the percentage of LO-criticality jobs not executed ($JNE(\%)$) for each of the schemes, for task sets with harmonic periods. We observe that for our baseline configuration, the bailout protocol (BP) is effective in reducing the percentage of LO-criticality jobs that are not executed compared to the AMC+ scheme. Here, increasing execution time budgets ($C(BU)$) by making use of static slack, leads to a roughly similar reduction in $JNE(\%)$ as BP. Since the bailout protocol and making use of static slack and gain time are complementary techniques, the BPSG scheme provides significantly better performance than AMC+SG or BP.

Figure 6 shows the results for non-harmonic task sets. Here the bailout policy is less effective at reducing the number of LO-criticality jobs not executed. This is because on average the busy periods tend to be shorter with non-harmonic task sets, with major peaks in the overall load not occurring as frequently. This means that an idle instant typically occurs shortly after entry into HI-criticality mode allowing both AMC+ and the bailout policy to recover back to LO-criticality (normal) mode in a similar time, with a similar number of LO-criticality jobs not executed.

Figures 4, 5, 7 and 8 provide further assessment of the performance of the different schemes. These results show that the percentage of time ($TiH(\%)$) in HI-criticality mode (or bailout and recovery modes) and the number of times that the system enters HI-criticality mode as a percentage of the number of jobs of HI-criticality tasks ($NiH(\%)$) are largest for the AMC+ scheme and smallest for BPSG. The bailout policy, which operates once HI-criticality mode is entered, does not act to reduce the number of times that the system enters HI-criticality mode, hence the $NiH(\%)$ values are very similar for AMC+ and BP, for AMC+S and BPS, and for BPSG and AMC+SG. As expected, both statically increasing LO-criticality budgets using static slack and runtime reclamation of gain time are highly effective in reducing the number of times that the system enters HI-criticality mode ($NiH(\%)$) and hence also the proportion of time spent in that mode ($TiH(\%)$).

Figure 5 shows that the bailout protocol, the use of static slack, and the runtime reclamation of gain time are all effective in reducing the total time spent in HI-criticality mode in harmonic task sets. We note that in contrast to the harmonic case, for non-harmonic task sets (see Figure 8) the bailout policy is unable to achieve an appreciable reduction in the time in HI-criticality mode compared to the equivalent AMC policy. This is due to the fact that the busy periods are on average much shorter with non-harmonic task sets and so once HI-criticality mode is entered, an idle instant is quickly reached allowing the system to recover to LO-criticality (normal) mode. This is the reason why the time in HI-criticality mode is much shorter for non-harmonic task sets.

7. We note that functional testing, even that requiring MCDC coverage, is not in general sufficient to determine WCETs when the hardware platform has components that cause execution times to be dependent on the execution history e.g. caches. Hence the need for an engineering margin to define $C(HI)$, and a non-zero probability that $C(LO)$ is exceeded during operation.

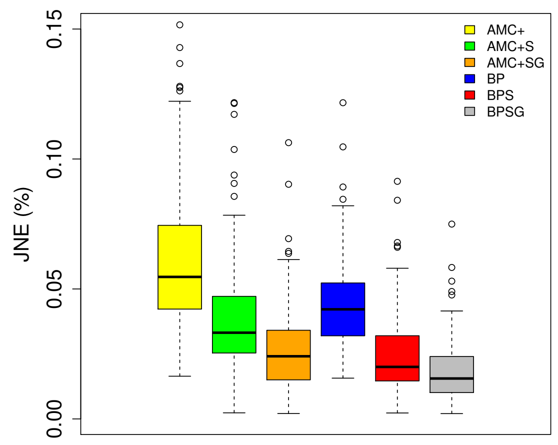

Fig. 3. Results for $JNE(\%)$ - 80% LO-criticality Utilisation: Harmonic Periods

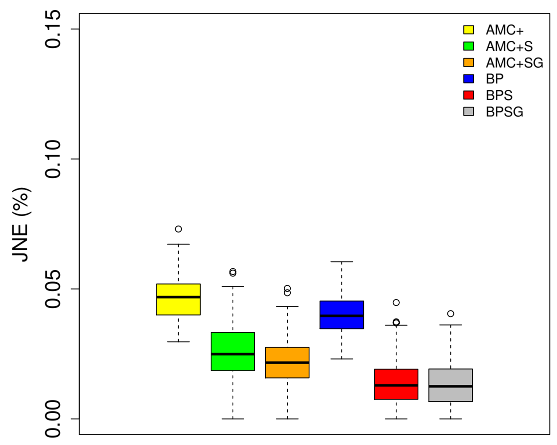

Fig. 6. Results for $JNE(\%)$ - 80% LO-criticality Utilisation: Non-Harmonic Periods

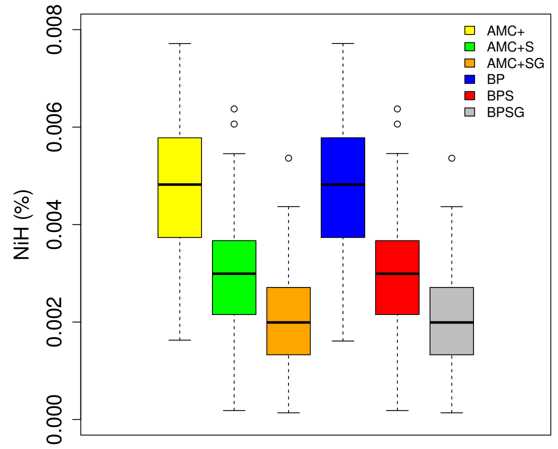

Fig. 4. Results for $NiH(\%)$ - 80% LO-criticality Utilisation: Harmonic Periods

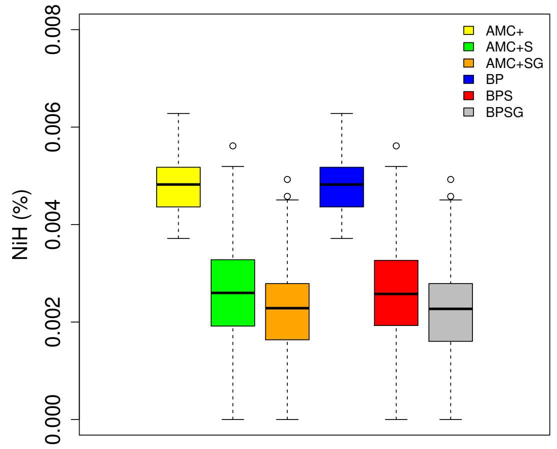

Fig. 7. Results for $NiH(\%)$ - 80% LO-criticality Utilisation: Non-Harmonic Periods

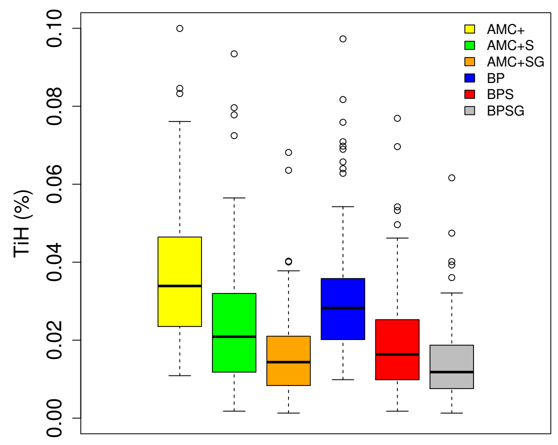

Fig. 5. Results for $TiH(\%)$ - 80% LO-criticality Utilisation: Harmonic Periods

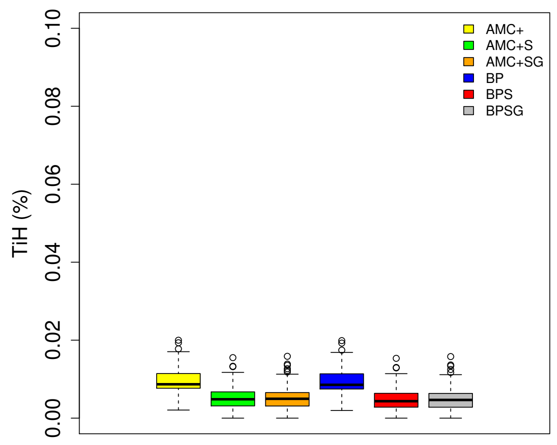

Fig. 8. Results for $TiH(\%)$ - 80% LO-criticality Utilisation: Non-Harmonic Periods

6.4 Additional Evaluation Results: Varying Parameters

In this section, we provide additional evaluation results showing how the performance of the different scheduling schemes changes when specific parameters are varied. The parameters varied were as follows:

- LO-criticality utilisation (Default 0.8).
- Criticality Factor CF (Ratio of HI-criticality to LO-criticality execution time. Default $CF = 2.0$).
- Criticality Probability CP (Probability that a task is of HI-criticality. Default $CP = 0.5$).
- Failure Probability FP (Probability that a job of a HI-criticality task exceeds $C(LO)$. Default $FP = 10^{-4}$).

In each of the experiments, one parameter was varied while the others were held constant at their default values. The results of these experiments show the average values of the three metrics of interest: $JNE(\%)$, $NiH(\%)$, and $TiH(\%)$. Recall that $JNE(\%)$ is the percentage of LO-criticality jobs that are not executed, $NiH(\%)$ is the number of times HI-criticality mode is entered as a percentage of the maximum possible, i.e the total number of jobs of HI-criticality tasks. Finally, $TiH(\%)$ is the percentage of the simulation interval spent in HI-criticality mode. The experiments were repeated for both harmonic and non-harmonic task sets.

6.4.1 Varying LO-criticality utilisation

Figures 9 and 12 show how $JNE(\%)$ changes as the overall LO-criticality utilisation is varied from 0.65 to 0.95. We observe that static slack stealing for increased execution time budgets, gain time reclamation and the bailout policy are all effective in reducing $JNE(\%)$. At high utilisation levels, the basic AMC+ and BM policies result in substantially higher values of $JNE(\%)$ with harmonic task sets than with non-harmonic task sets. This is because with harmonic task sets conditions of peak load i.e. long processor busy periods reoccur much more frequently. Since harmonic task sets are easier to schedule[8], using slack to increase execution time budgets retains substantial effectiveness at high levels of utilisation (e.g. 0.95). As long processor busy periods are more frequent with harmonic task sets and become longer with increasing utilisation, in this case the bailout policy becomes more effective compared to $AMC+$ as utilisation levels increase.

Figures 10 and 13 show how $NiH(\%)$ changes as the overall LO-criticality utilisation is varied from 0.65 to 0.95. We note that in these experiments, both static slack stealing for increased execution time budgets and gain time reclamation are effective in reducing the number of times HI-criticality mode is entered. As expected; however, the bailout policy has no noticeable effect compared to AMC+. This is because the bailout policy only comes into effect once HI-criticality mode has been entered.

Figures 11 and 14 show how $TiH(\%)$ changes as the overall LO-criticality utilisation is varied from 0.65 to 0.95. Here there are clear differences in performance between harmonic and non-harmonic task sets. With non-harmonic task sets, there are very few long busy periods thus when HI-criticality mode is entered it is soon exited as an idle instant is reached. This means that the percentage of the total time spent in the HI-criticality mode is much less than with harmonic tasks sets, and also explain why the bailout policy is unable to significantly reduce the time in HI-criticality mode. This is also the case with gain time reclamation, since the busy periods are too short for substantial gain time to accumulate and prevent the transition to HI-criticality mode. Static slack stealing for increased execution time budgets is still effective in this case, since it reduces the number of times HI-criticality mode is entered which impacts the total time in that mode.

8. The utilisation bound is 1 for pure harmonic task sets and 0.69 for non-harmonic task sets.

6.4.2 Varying the Criticality Factor (CF)

Figures 15 and 18 show how $JNE(\%)$ changes as the Criticality Factor (CF) is varied from 1.25 to 2.5.

We observe that with harmonic task sets, $JNE(\%)$ decreases with increasing CF for the BM and AMC+ schemes. This is because with small values of CF, schedulable task sets can be generated that include low priority but HI-criticality tasks with long periods and long execution times. The presence of such tasks increases the time in HI-criticality mode (see Figure 17) and thus also $JNE(\%)$. This effect is not apparent with non-harmonic task sets since they are much harder to schedule and so do not readily permit such tasks.

In both the harmonic and non-harmonic cases, the use of both static slack stealing to increase execution time budgets and gain time reclaiming are highly effective in reducing the number of times that the system enters HI-criticality mode ($NiH(\%)$), thus also reducing the amount of time spent in that mode ($TiH(\%)$) - see Figures 16 to 20. These techniques have less effect as the value of CF increases, since they have to mitigate the increasing effect of longer HI-criticality execution times.

6.4.3 Varying the Criticality Probability (CP)

Figures 21 to 26 show how $JNE(\%)$, $NiH(\%)$, and $TiH(\%)$ change as the Criticality Probability (CP) controlling the proportion of HI-criticality tasks varies from 0.3 to 0.7. Here the key behaviours of the schemes remain as reported for the baseline configurations discussed in detail in section 6.3. The predominant effect of increasing the proportion of HI-criticality tasks is to increase the number of times that the system enters HI-criticality mode and thus also the proportion of LO-criticality jobs not executed and the proportion of time spent in HI-criticality mode. The $NiH(\%)$ value remains relatively constant since that measure is normalised to the number of HI-criticality jobs.

6.4.4 Varying the Failure Probability (FP)

Figures 27 to 32 show how the normalized metrics $JNE(\%)/FP$, $NiH(\%)/FP$, and $TiH(\%)/FP$ change as the Failure Probability (FP) controlling the proportion of HI-criticality jobs that exceed their LO-criticality execution time budget varies from 10^{-5} to 1. These graphs show that the metrics $JNE(\%)$, $NiH(\%)$, and $TiH(\%)$ have an approximately linear relationship with the Failure Probability for Failure Probabilities of 10^{-2} and below, taking nearly constant values for each scheduling scheme. The figures show that the relative performance of the various schemes is effectively independent of the likelihood of HI-criticality tasks exhibiting HI-criticality behavior. At very high Failure Probabilities e.g. $10^{-1} = 0.1$ and 1, then there are typically multiple jobs exhibiting HI-criticality behavior within each HI-criticality mode interval. Thus the metric $JNE(\%)/FP$ reduces, tending towards values just below 100 for $FP = 1$, meaning that when every job of a HI-criticality task exhibits HI-criticality behavior, almost 100% of the LO-criticality jobs are not executed. The $NiH(\%)/FP$, and $TiH(\%)/FP$ metrics behave similarly.

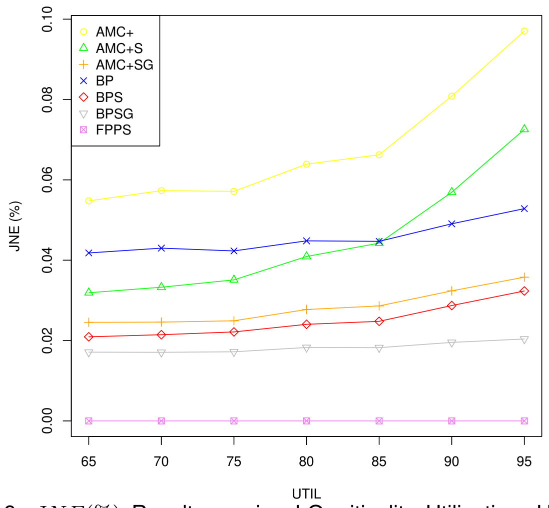

Fig. 9. $JNE(\%)$ Results varying LO-criticality Utilisation: Harmonic Periods

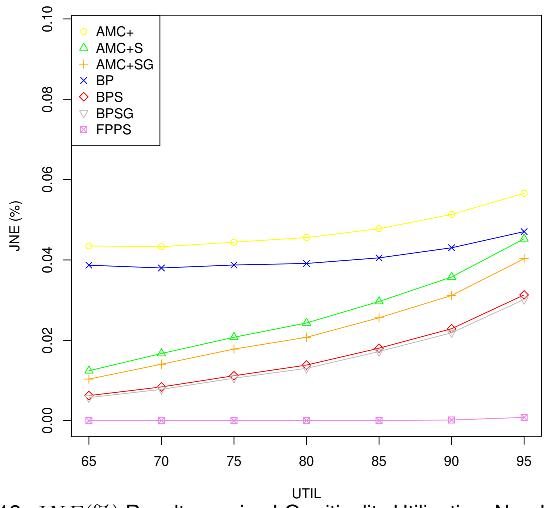

Fig. 12. $JNE(\%)$ Results varying LO-criticality Utilisation: Non-Harmonic Periods

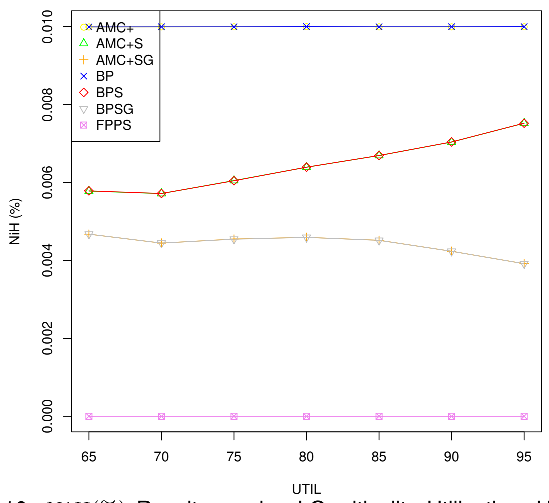

Fig. 10. $NiH(\%)$ Results varying LO-criticality Utilisation: Harmonic Periods

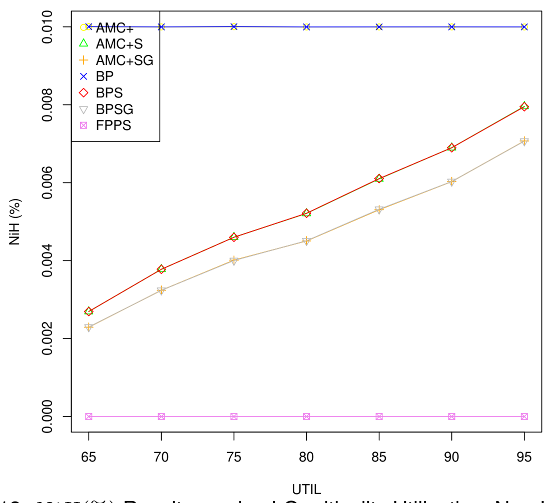

Fig. 13. $NiH(\%)$ Results varying LO-criticality Utilisation: Non-Harmonic Periods

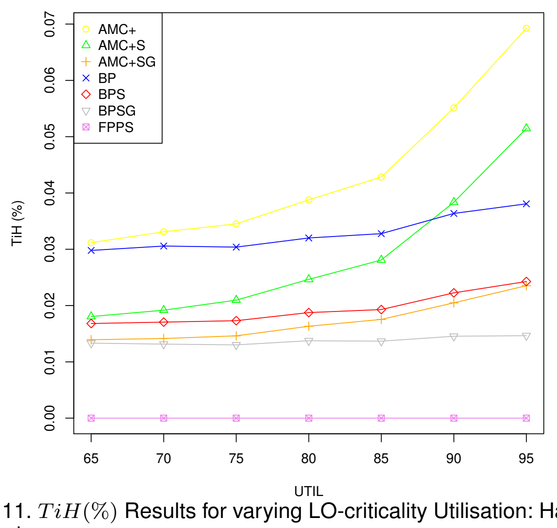

Fig. 11. $TiH(\%)$ Results for varying LO-criticality Utilisation: Harmonic Periods

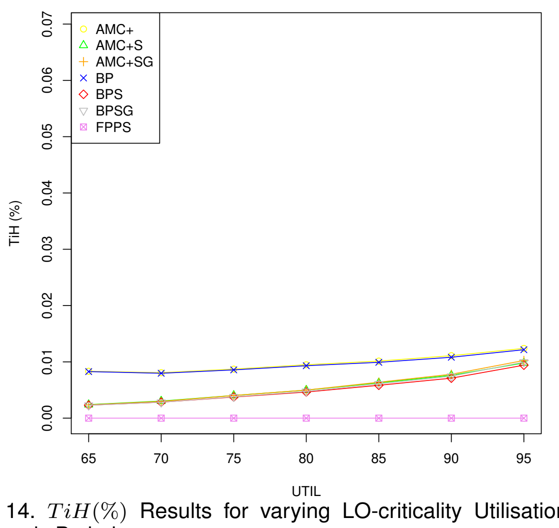

Fig. 14. $TiH(\%)$ Results for varying LO-criticality Utilisation: Non-Harmonic Periods

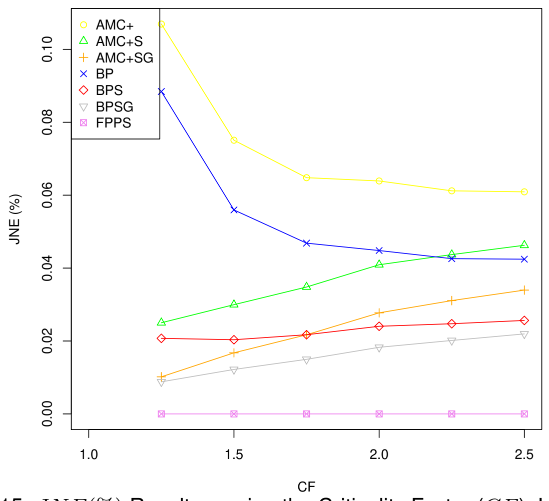

Fig. 15. $JNE(\%)$ Results varying the Criticality Factor (CF): Harmonic Periods

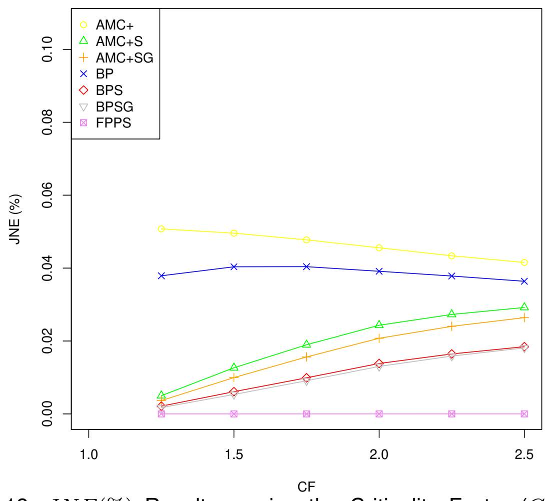

Fig. 18. $JNE(\%)$ Results varying the Criticality Factor (CF): Non-Harmonic Periods

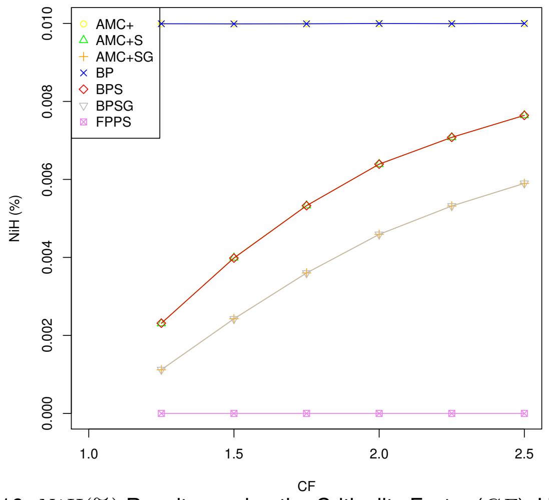

Fig. 16. $NiH(\%)$ Results varying the Criticality Factor (CF): Harmonic Periods

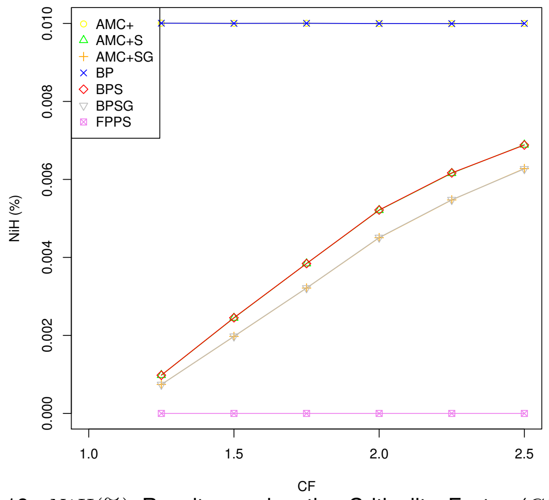

Fig. 19. $NiH(\%)$ Results varying the Criticality Factor (CF): Non-Harmonic Periods

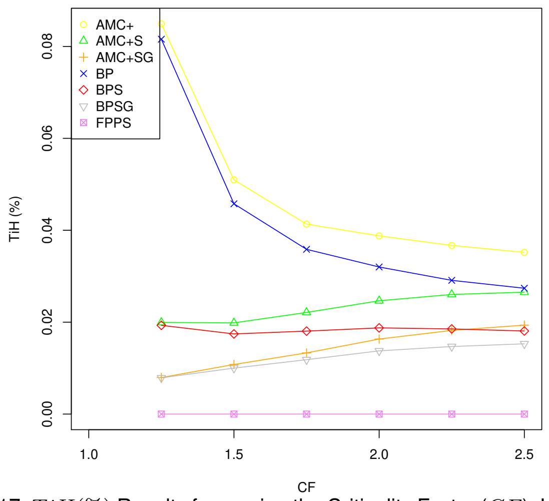

Fig. 17. $TiH(\%)$ Results for varying the Criticality Factor (CF): Harmonic Periods

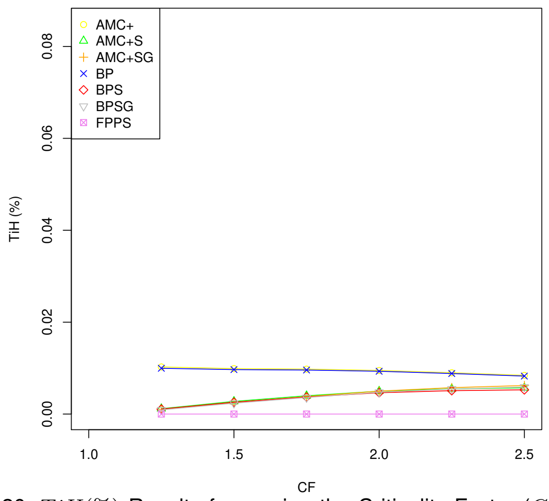

Fig. 20. $TiH(\%)$ Results for varying the Criticality Factor (CF): Non-Harmonic Periods

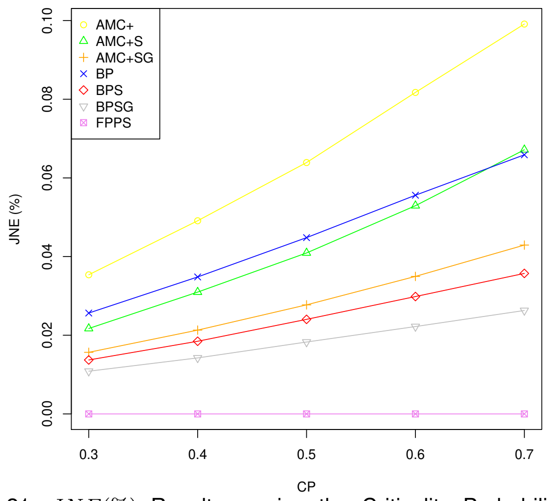

Fig. 21. $JNE(\%)$ Results varying the Criticality Probability (CP): Harmonic Periods

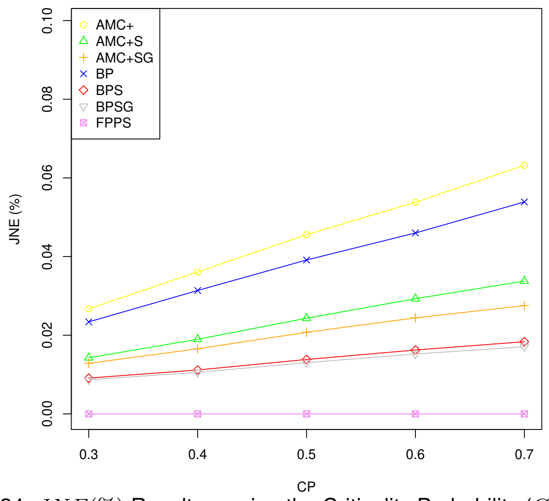

Fig. 24. $JNE(\%)$ Results varying the Criticality Probability (CP): Non-Harmonic Periods

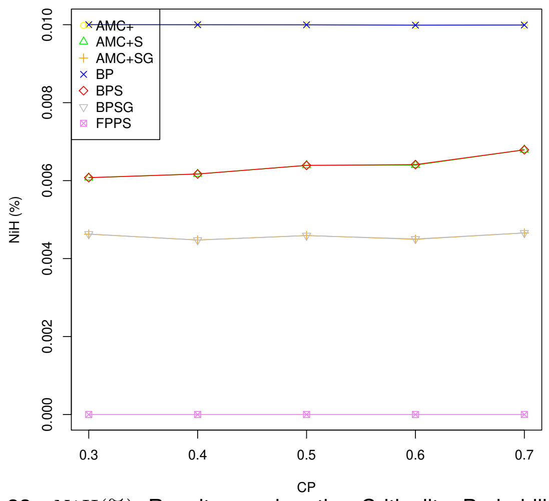

Fig. 22. $NiH(\%)$ Results varying the Criticality Probability (CP): Harmonic Periods

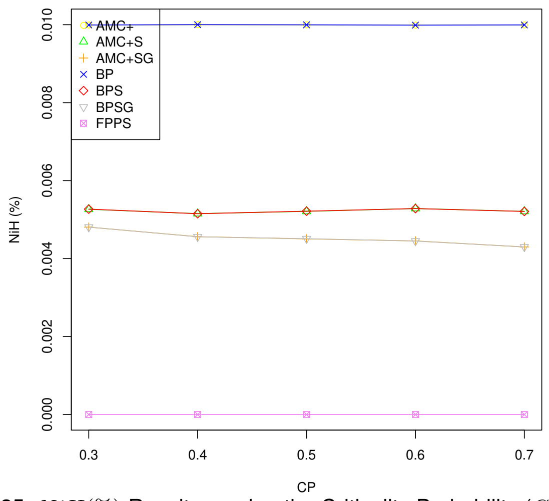

Fig. 25. $NiH(\%)$ Results varying the Criticality Probability (CP): Non-Harmonic Periods

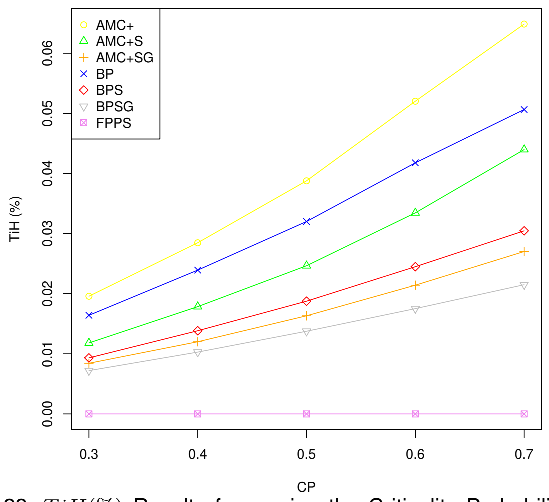

Fig. 23. $TiH(\%)$ Results for varying the Criticality Probability (CP): Harmonic Periods

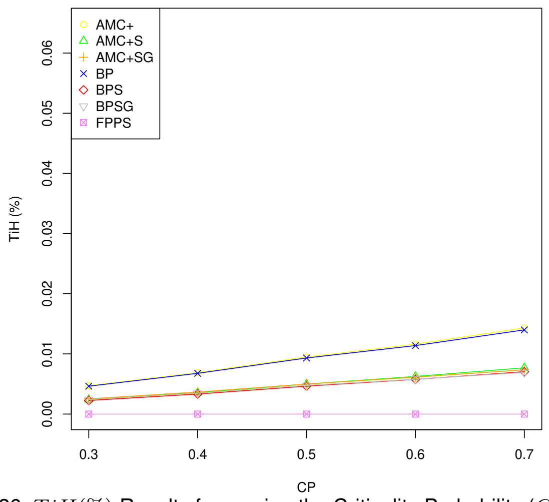

Fig. 26. $TiH(\%)$ Results for varying the Criticality Probability (CP): Non-Harmonic Periods

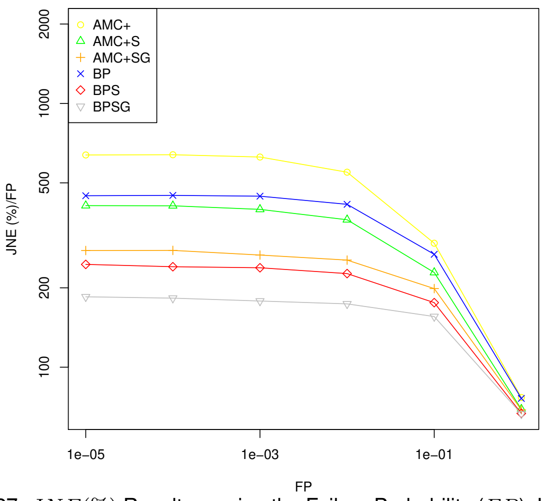

Fig. 27. $JNE(\%)$ Results varying the Failure Probability (FP): Harmonic Periods

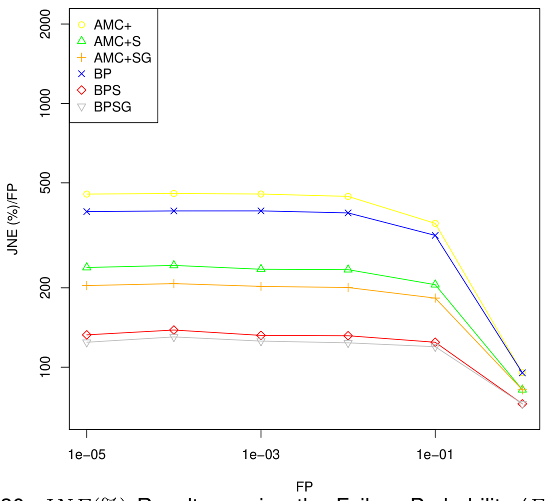

Fig. 30. $JNE(\%)$ Results varying the Failure Probability (FP): Non-Harmonic Periods

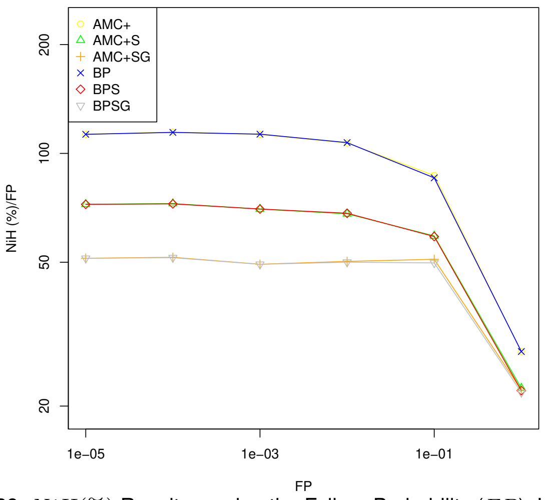

Fig. 28. $NiH(\%)$ Results varying the Failure Probability (FP): Harmonic Periods

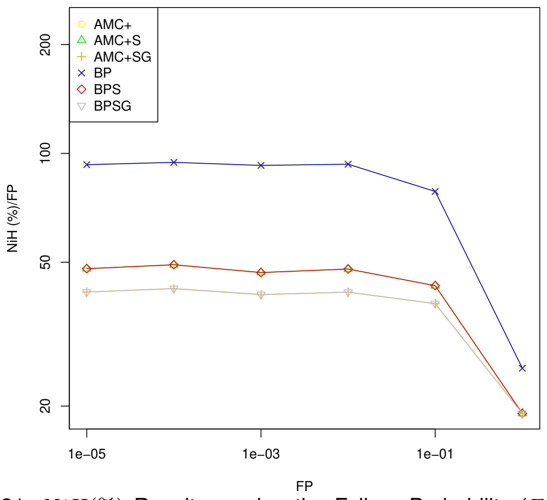

Fig. 31. $NiH(\%)$ Results varying the Failure Probability (FP): Non-Harmonic Periods

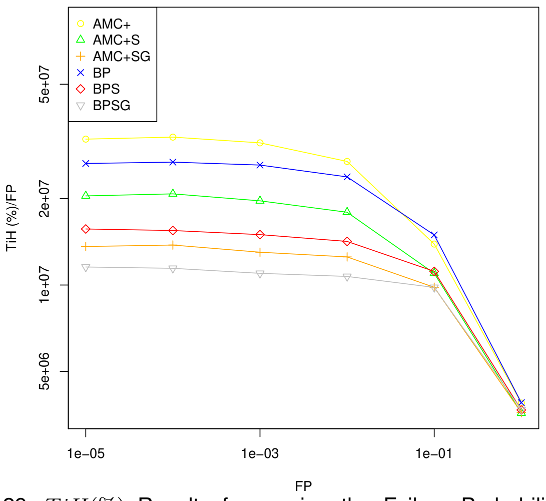

Fig. 29. $TiH(\%)$ Results for varying the Failure Probability (FP): Harmonic Periods

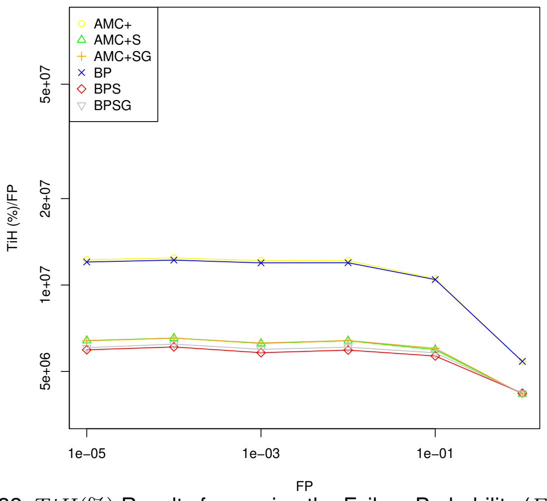

Fig. 32. $TiH(\%)$ Results for varying the Failure Probability (FP): Non-Harmonic Periods

Note lines for the FPPS policy are omitted from figures 27 to 32, since it is not possible to show zero values on graphs with a log scale.

6.5 Summary of evaluation results

The results of our evaluation can be summarised as follows. The scenario-based simulations showed that the bailout protocol is highly effective in reducing the percentage of LO-criticality tasks abandoned $JNE(\%)$ to ensure correct HI-criticality behaviour, and also for harmonic task sets, in reducing the percentage of time spent in the HI-criticality mode $TiH(\%)$. With non-harmonic task sets, the short busy periods mean that after a HI-criticality task exhibits HI-criticality behavior, on average an idle instant is quickly reached. Thus both AMC and bailout-based policies show similar performance in terms of the percentage of time spent in HI-criticality mode $TiH(\%)$. Since the bailout policy acts only once HI-criticality mode has been entered, it has very little effect on the number of times that HI-criticality mode is entered and thus $NiH(\%)$. Both off-line use of slack and online reclamation of gain-time reduce both the number of times the system enters HI-criticality mode, thus reducing $NiH(\%)$, and as a consequence reducing the overall amount of time it spends in that mode, and so reducing $TiH(\%)$. Both of these methods complement the bailout protocol. For both harmonic and non-harmonic task sets, in our baseline simulation, the BPSG scheme reduced the percentage of LO-criticality jobs not executed $JNE(\%)$ by approximately a factor of three compared to the AMC+ scheme.

Importantly, in all of our experiments there were no HI-criticality deadlines misses (HDM) using the AMC, AMC+S, AMC+SG, BP, BPS, and BPSG schemes. There were also 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 number of HI-criticality deadline misses depended on the particular simulation conditions (peak loads being required), but nevertheless represented a level of failures which may not be acceptable in a real system.

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. Nearly all of these LO-criticality jobs are abandoned without execution; 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.

The additional evaluation results illustrated in Figures 9 to 32 showed that the advantages of utilising the bailout policy, increasing execution budgets using static slack stealing, and also runtime reclamation of gain time persist across a wide range of scenario parameters, including the mix of HI- and LO-criticality tasks, the ratio of their execution times, and the LO-criticality utilization. We also showed that the relative performance of the different schemes is largely independent of the probability of HI-criticality behavior occurring (up to very high probabilities where 10% or more of the HI-criticality jobs exhibit HI-criticality behavior). These results indicate that the bailout protocol combined with increased execution budgets using static slack stealing, and also runtime reclamation of gain time (i.e. BPSG) is likely to be effective in a wide range of practical cases.

7 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 (3) in Section 2.2), 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 (provided that the $C(HI)$ execution times are not violated).

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 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 7.1. *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:*

$$\forall j \;\; W^B(t_e, j) \leq W^N(t_e, j) \tag{4}$$

then all jobs released and not immediately abandoned[9] *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 normal 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 4.1 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 7.2. *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 (4) holds i.e. $\forall j \;\; W^B(t_e, j) \leq W^N(t_e, j)$.*

9. Recall that LO-criticality jobs released in recovery mode are immediately abandoned.

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:

$$\forall j \;\; W^B(t, j) \leq W^N(t, j) + BF \tag{5}$$

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 (5) 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):

$$\forall j \in hp(k) \;\; W^B(t, j) = 0 \tag{6}$$

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.

$$\forall j \in lep(k) \;\; W^B(t, j) \leq W^N(t, j) + BF \tag{7}$$

Case 2: BF reduction (abandonment): Recall that under mechanism (vii) a job of a LO-criticality task that is released in 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 (6), 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 (7) 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 (6) and (7) 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:

$$\forall j \in hp(k) \;\; W^B(t_e, j) = 0 \tag{8}$$

and

$$\forall j \in lep(k) \;\; W^B(t_e, j) \leq W^N(t_e, j) + BF \tag{9}$$

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 7.3. *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 7.1, 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 7.3 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 7.4. *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 7.3 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 7.3 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 (3) in Section 2.2). 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 (4), 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).

8 Extension to Multiple Criticality Levels

The bailout protocol defined in this paper can be extended to more criticality levels in a straightforward way. We assume there are m criticality levels L^1 to L^m. Although we assume an arbitrary value for m, in practice its is unlikely to be more than about five.

The criticality level of each task τ_i is denoted by L_i. In general, each task τ_i may have an execution time $C_i(L^k)$ defined for each criticality level L^k where $L^k \leq L_i$. The AMC policy and its analysis have been extended to such a model [43]. In this case, when a task of criticality higher than L^k executes for its $C(L^k)$ execution time without completing, then the system criticality level is set to the higher of the current level[10] and L^{k+1}. Once the system is at criticality level L^{k+1}, then jobs of tasks with lower criticality may complete, but all newly released jobs of those tasks are abandoned. Thus as the system moves up through the criticality levels, which it may do step by step, so it sheds load from tasks of lower criticality. At any point, when an idle instant occurs, then the system reverts back to the lowest criticality level.

While the above model is interesting in theory, in practice it is unlikely that tasks will have more than two execution time budgets defined [44], one for the lowest criticality level (typically obtained via measurement taking a "high water mark") and one for the task's own criticality level (which may be obtained via a more rigourous process in compliance with the appropriate standard, for example involving static analysis, MCDC testing etc.). In an abuse of our previous notation, we refer to these values as $C(LO)$ and $C(HI)$. In terms of the more general model, it is therefore the case that

10. A task may still execute an incomplete job once the system criticality level is higher than that of the task, which is why this check is needed.

$\forall k | L^k < L_i, C(L^k) = C(LO)$, and $C(L_i) = C(HI)$. Further, the behaviour of the AMC policy is such that if a job of task τ_i executes for its $C(LO)$ without signaling completion, then the system criticality level is set to the higher of the current level and the criticality level L_i of the task.

We now show that the bailout protocol can be easily adapted to the above model where there are multiple criticality levels, but each task only has two distinct execution time budgets $C(LO)$ and $C(HI)$. As well as the concepts of normal, bailout and recovery modes introduced by the bailout protocol, we also need to record the *system criticality level* corresponding to a criticality level from L^1 to L^m. In normal mode, the system criticality level is always L^1, while with this extended model, the bailout and recovery modes cover the higher criticality levels. The system may move up through the criticality levels while in the bailout mode. It remains at a single criticality level while in recovery mode, and then either transitions directly back to normal mode and the lowest criticality level, or back to bailout mode if further high criticality behaviour occurs. The rationale for this simple approach of returning directly to the lowest criticality level is that transitions to high criticality levels are expected to be rare and after such an event the aim is simply to return the system to its normal operating behaviour as soon as possible. The alternative would be to devise a more complex protocol that is able to step down the criticality levels one at a time. In our view the small gains that might be obtained by such an approach are out-weighted by the increase in complexity of the algorithms and accounting needed.

To support multiple criticality levels, the following minor adaptations are needed to the bailout protocol described as a set of rules from (i) to (xii) in Section 4.1. For ease of reference, we repeat those rules below, with the additions shown in *italic*. Where we refer to "HI-criticality" tasks, we mean any task whose criticality level is greater than L^1, as opposed to LO-criticality tasks whise criticality level is L^1. Note that the bailout aspects of the protocol remain precisely the same. The only differences are to do with transitions between system criticality levels, and which tasks can run at those levels.

Normal mode:

(i) While all jobs of HI-criticality tasks execute for no more than their $C(LO)$ values, then the system remains in normal mode, *and the system criticality level is L^1.*

(ii) If any job of a HI-criticality task (i.e. a task with $L_i > L^1$) 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 $BF = C(HI) - C(LO)$. *The system criticality level is set to L_i, the criticality level of the task exhibiting HI-criticality behaviour.*

Bailout mode:

(iii) If any job of a HI-criticality task 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: $BF = BF + C(HI) - C(LO)$. *The system criticality level is set to the higher of the current level and L_i, the criticality level of the task exhibiting HI-criticality behaviour.*

(iv) If any job of a HI-criticality task 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 job of a LO-criticality task (i.e. a task with $L_i = L^1$) completes with an execution time of e, with $e \leq C(LO)$ then it donates its underspend (if any) to the bailout fund: $BF = BF - (C(LO) - e)$. Note, such a job would need to have been released in an earlier normal mode.

(vi) If any job of a HI-criticality task 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) *Jobs of tasks of criticality lower than the current system criticality level are abandoned (not started)*. Further, when the scheduler would otherwise dispatch 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 (i.e. of any criticality level $> L^1$) with outstanding execution is recorded (let this job be J_k) and the recovery mode is entered.

(ix) If during bailout mode, an idle instant occurs, then an immediate transition is made to normal mode, and BF is set to zero. *The system criticality level is set to L^1.*

Recovery mode:

(x) *Jobs of tasks of criticality lower than the current system criticality level 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. *The system criticality level is set to the higher of the current level and L_i, the criticality level of the task exhibiting HI-criticality behaviour.*

(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 system criticality level is set to L^1.*

We now show that the bailout protocol for multiple criticality level systems correctly schedules any task set that is schedulable under the extended AMC policy, analysis for which is given in [43].

The bailout protocol for systems with multiple criticality levels reflects precisely the behaviour of the extended AMC policy up until the point at which recovery is completed and a transition is made back to normal mode and system criticality level L^1. At that point, the protocol ensures (as it does for the dual criticality case), that no active job of a HI-criticality task can be subject to more interference than it would have been had the system simply remained in normal mode and system criticality level L^1. Limiting interference to no more than in normal mode is sufficient to ensure that all jobs of tasks started in recovery mode or the subsequent normal mode will meet their deadlines. Together with behaviour that is identical to that of AMC during the bailout and recovery modes, this means that all tasks with criticality level L^k are guaranteed to meet their deadlines provided that the system criticality level does not exceed L^k between their release time and their deadline. We now formally prove that this is the case.

We first note that Lemma 7.1 and Lemma 7.2 (in section 7) apply with minor adaptations as follows. With multiple criticality levels, then in Case 2 of Lemma 7.2 BF reduction due to abandonment may also occur due to jobs of HI-criticality tasks being abandoned because the system criticality level is higher than that of the task; however, the logic and the argument is identical to that stated in the Lemma for LO-criticality tasks. Theorem 7.3 (in section 7) also applies, again also considering abandonment of jobs of HI-criticality tasks that are below the system criticality level in the same way as abandonment of LO-criticality tasks.

Theorem 7.3 (in section 7) shows that jobs of LO-criticality tasks (criticality level L^1) that are released and complete in normal mode with no intervening start of a bailout mode (and thus system criticality level higher than L^1) are guaranteed to meet their deadlines under the bailout policy. It remains to consider HI-criticality tasks of an arbitrary criticality level L^k.

Theorem 8.1. *All jobs of HI-criticality tasks that have their release times and deadlines in a interval that does not include the system operating at a criticality level higher than that of the task will meet their deadlines, provided that the task set is schedulable according to analysis of the extended AMC policy, and scheduled at runtime using FPPS and the bailout protocol for systems with multiple criticality levels.*

Proof. Theorem 7.3 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: Let J be an arbitrary job of an arbitrary HI-criticality task τ_i 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 (without the system criticality level exceeding L_i). The proof of Theorem 7.3 shows that $\forall j \;\; W^B(t_e, j) \leq W^N(t_e, j)$ holds at the end of any bailout mode interval. In other words, for each priority level j, the total pending workload at priority level j and higher is no greater than it would have been had the system remained in normal mode from the start with all jobs taking their $C(LO)$ execution times. Hence for any job J that is released in recovery mode or normal mode and completes in the next bailout mode interval or the recovery mode interval that follows it, there is a valid alternative scenario (complete set of jobs with execution times and release times) whereby the system remains in normal mode until the release of job J and then produces an identical schedule from that time onwards for both the AMC policy, and FPPS with the bailout protocol for multiple criticality levels. Since in the alternative scenario, the extended AMC policy guarantees that the deadline of J is met provided the system does not exceed criticality level L_i before the job completes, then so does the bailout protocol for multiple criticality levels.

Case 2: Let J be an arbitrary job of an arbitrary HI-criticality task τ_i that is released in a bailout mode interval and completes in that interval or the recovery mode interval that follows it. Consider the schedule from a time s that equates to the end of the previous bailout mode interval (or the start of the system if there isn't one). The proof of Theorem 7.3 shows that $\forall j \; W^B(t_e, j) \leq W^N(t_e, j)$ holds at the end of any bailout mode interval (it also trivially holds at system start up in normal mode). Thus the workload condition holds at time s. It follows that there is an alternative valid scenario (complete set of jobs with execution times and release times) whereby the system remains in normal mode until time s and then produces an identical schedule for the interval of time from s to the deadline of J, for both the AMC policy, and FPPS with the bailout protocol for multiple criticality levels. Since in the alternative scenario, the extended AMC policy guarantees that the deadline of J is met provided the system does not exceed criticality level L_i before the job completes, then so does the bailout protocol for multiple criticality levels.

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. □

9 Conclusions

In mixed criticality systems (MCS) most criticality levels will require that deadlines are always met. However, it is also necessary to design these systems so they can make effective use of the processing resources available. This involves making realistic, as opposed to pessimistic assumptions about execution time budgets, and employing mechanisms that behave robustly in the rare 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 introduced 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 two existing mechanisms, based on off-line slack calculation and online gain time reclamation. We showed how both of these techniques can be employed along with the bailout protocol to further improve performance.

Our scenario-based simulations showed that the bailout protocol is highly effective in reducing the amount of time spent in the HI-criticality mode and thus in reducing the number of LO-criticality tasks abandoned to ensure correct HI-criticality behaviour. Both off-line use of slack and online reclamation of gain-time reduce the number of times that the system actually needs to enter a HI-criticality mode, and also the overall amount of time it spends in that mode. Both of these methods complement the bailout protocol. We used schedulability analysis techniques to show that the bailout protocol has the same level of guarantee as the best previously published approach (AMC). Finally, we showed how the bailout protocol permits straightforward extension to multiple criticality levels.

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. The authors would like to thank Frank Soboczenski for his help in producing the graphs used in this paper.

REFERENCES

[1] I. Bate, A. Burns, and R.I.Davis, "A bailout protocol for mixed criticality systems," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015.

[2] 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.

[3] 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)*, 2007, pp. 239–243.

[4] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2008, pp. 147–155.

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

[6] P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic sprodic tasks," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 135–144.

[7] F. Santy, P. Richard, M. Richard, and J. Goossens, "Relaxing mixed-criticality scheduling strictness for task sets scheduled with FP," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 155–165.

[8] 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)*, 2011, pp. 13–23.

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

[10] A. Burns and R. Davis, "Mixed criticality systems - a review," Department of Computer Science, University of York, Tech. Rep., 2014.

[11] P. Graydon and I. Bate, "Realistic safety cases for the timing of systems," *The Computer Journal*, vol. 57, no. 5, pp. 759–774, 2014.

[12] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. Cazorla, "Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study," in *IEEE International Symposium on Industrial Embedded Systems (SIES)*, June 2013, pp. 241–248.

[13] N. Audsley, R. Davis, A. Burns, and A. Wellings, *Appropriate mechanisms for the support of optional processing in hard real-time systems*, 1994, pp. 23–27.

[14] J. Lehoczky, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," in *Proc. of Real-Time Systems Symposium (RTSS)*, Dec 1992, pp. 110–123.

[15] R. Davis, K. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," in *Proc. of IEEE Real-Time Systems Symposium*. IEEE, 1993, pp. 222–231.

[16] S. 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*, vol. 61, no. 8, pp. 1140–1152, 2012.

[17] 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*, vol. 46, no. 3, pp. 305–331, 2010.

[18] S. Baruah and G. Fohler, "Certification-cognizant time-triggered scheduling of mixed-criticality systems," in *Proc. of IEEE Real-time Systems Symposium (RTSS)*, December 2011, pp. 3–12.

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

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

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

[22] K. Tindell and A. Alonso, "A very simple protocol for mode changes in priority preemptive systems," Universidad Politecnica de Madrid, Tech. Rep., 1996.

[23] 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)*, 2013, pp. 147–152.

[24] 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)*, 2013, pp. 1–6.

[25] 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)*, Sept 2013, pp. 1–8.

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

[27] O. Gettings, S. Quinton, and R. I. Davis, "Mixed criticality systems with weakly-hard constraints," in *Proc. of International Conference on Real Time and Networks Systems (RTNS)*. New York, NY, USA: ACM, 2015, pp. 237–246. [Online]. Available: http://doi.acm.org/10.1145/2834848.2834850

[28] R. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2007, pp. 3–14.

[29] 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.

[30] J. Ren and L. T. X. Phan, "Mixed-criticality scheduling on multiprocessors using task grouping," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, July 2015, pp. 25–34.

[31] 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*. Springer, 1997, pp. 72–82.

[32] E. Bini, M. D. Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," in *Proc. of Euromicro Conference on Real-Time Systems (ECRTS)*, 2006, pp. 13–22.

[33] R. Davis, "On exploiting spare capacity in hard real-time systems," Ph.D. dissertation, University of York, UK, 1995.

[34] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters*, vol. 79, no. 1, pp. 39–44, 2001.

[35] B. Sprunt, J. Lehoczky, and L. Sha, "Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 1988, pp. 251–258.

[36] G. Bernat, I. Broster, and A. Burns, "Rewriting history to exploit gain time," in *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2004, pp. 328–335.

[37] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," *Ada Letters*, vol. 24, no. 4, pp. 1–8, 2004.

[38] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," in *Proceedings of the IEEE*, 2003, pp. 127–144.

[39] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of real-time embedded systems," in *Proc. IEEE Real-Time Systems Symposium (RTSS)*, 2012, pp. 351–362.

[40] I. Bate and A. Burns, "An integrated approach to scheduling in safety-critical embedded control systems," *Real-Time Systems Journal*, vol. 25, no. 1, pp. 5–37, 2003.

[41] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Journal of Real-Time Systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[42] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority preemptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, pp. 284–292, 1993.

[43] T. Fleming and A. Burns, "Extending mixed criticality scheduling," in *Proc. of 1st Workshop on Mixed Criticality Systems (WMC)*, Dec 2013, pp. 7–12.

[44] A. Burns, "An augmented model for mixed criticality," in *Proc. of Dagstuhl seminar on Mixed Criticality on Multicore/Manycore Platforms*, 2015.

Iain Bate is a senior lecturer in Real-Time Systems at the Computer Science Department, University of York. His research interests include scheduling and timing analysis, design and analysis of safety-critical systems, and engineering of complex systems of systems including Wireless Sensor Networks. He is the Editor-in-Chief of the Journal of Systems Architecture, a frequent member of programme committees for distinguished international conferences, and a Visiting Professor at Mälardalen University in Sweden.

Alan Burns Alan Burns co-leads the Real-Time Systems Research Group at the University of York. His research interests cover a number of aspects of real-time systems including the assessment of languages for use in the real-time domain, distributed operating systems, the formal specification of scheduling algorithms and implementation strategies, and the design of dependable user interfaces to real-time applications. Professor Burns has authored/co-authored 500 papers/reports and books. Most of these are in the real-time area. His teaching activities include courses in Operating Systems and Real-time Systems. In 2009 Professor Burns was elected a Fellow of the Royal Academy of Engineering. In 2012 he was elected a Fellow of the IEEE.

Robert I. Davis is a Senior Research Fellow in the Real-Time Systems Research Group at the University of York, UK, and an Inria International Chair with the AOSTE team at Inria-Paris, Paris, France. Robert received his DPhil in Computer Science from the University of York in 1995. Since then he has founded three start-up companies, all of which have succeeded in transferring real-time systems research into commercial products. Roberts research interests include the following aspects of real-time systems: scheduling algorithms and analysis for single processor, multiprocessor and networked systems; analysis of cache related pre-emption delays, mixed criticality systems, and probabilistic hard real-time systems.