Adaptive Mixed Criticality Scheduling with Deferred Preemption

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

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

***Abstract*—Adaptive Mixed Criticality (AMC) scheduling has previously been shown to be the most effective fixed priority approach for scheduling mixed criticality systems, while the idea of final non-preemptive regions has been shown to improve the schedulability of systems with a single criticality level. In this paper, we combine AMC with the concept of non-preemptive regions by making the final part of each task's execution at each criticality level non-preemptive. We derive schedulability analysis for this approach, and provide an effective algorithm for choosing each task's priority and the durations of its non-preemptive regions. Evaluations illustrate the benefits of this approach in terms of increased schedulability.**

I. Introduction

The formal study of mixed criticality systems (MCSs) is a relatively new undertaking, starting with the paper by Vestal (of Honeywell Aerospace) in 2007 [31]. Since then a standard model has emerged (see for example [6], [5], [18], [27], [20]). For dual (HI- and LO-) criticality systems executing on a uniprocessor and using priority based scheduling, this standard model has the following properties:

- A mixed criticality system is defined to execute in either of two modes: a HI-criticality mode or a LO-criticality mode.
- Each task is characterised by its criticality (HI or LO), 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), denoted by $C(HI)$ and $C(LO)$. A key aspect of the standard model is that $C(HI) \geq C(LO)$.
- The system starts in the LO-criticality mode, and remains in that mode as long as all jobs execute within their LO-criticality computation times ($C(LO)$).
- If any job executes for its $C(LO)$ execution time without completing then the system immediately moves to the HI-criticality mode.
- As the system moves to the HI-criticality mode all LO-criticality tasks are abandoned, and no further LO-criticality jobs are executed.
- The system remains in the HI-criticality mode.
- Tasks are assumed to be independent of each other (they do not share any resource other than the processor).

This abstract behavioural model has been very useful in allowing key properties of mixed criticality systems to be derived, but it has been necessary to extend the model to allow for more realistic characteristics such as allowing some LO-criticality work to execute in the HI-criticality mode and for the LO-criticality mode itself to be reinstated [5], [30], [29], [23], [14]. Work has also focused on criticality-aware resource control protocols that allow resource sharing between tasks [13], [32], [25].

In terms of scheduling, there have been a number of schemes proposed. Most are based on either fixed priority or EDF scheduling. In this paper, we focus on the former.

With fixed priority scheduling, most approaches use Audsley's algorithm [1] for priority assignment. We briefly outline the published schemes for dual criticality systems:

- Criticality Monotonic Priority Ordering (CrMPO) – all HI-criticality tasks have priorities that are higher than those of all LO-criticality tasks; within a criticality band, deadline monotonic priority ordering is used.
- Static Mixed Criticality (SMC-NO) – this is the original scheme proposed by Vestal [31] in which the analysis of HI-criticality tasks uses $C(HI)$ values for all tasks, but the analysis of LO-criticality tasks uses $C(LO)$ values; there is no runtime monitoring.
- Static Mixed Criticality (SMC) [3] – runtime enforcement ensures that LO-criticality tasks never execute for more than $C(LO)$, this can then be exploited in the analysis.
- Adaptive Mixed Criticality (AMC) [5] – this approach fully utilises the property of the implementation model that no LO-criticality task is released after a HI-criticality task has executed for its $C(LO)$ computation time without signalling completion; this is the approach built upon in this paper and is reviewed in more detail in Section III.
- Period Transformation (PT) [28], [31], [19] – the periods of HI-criticality tasks are transformed (reduced) so that the optimal priority order is also a criticality monotonic priority order (i.e. CrMPO).
- Two Priority (TwoP) – After the mode change the priorities of the HI-criticality tasks are modified to enhance schedulability [4].

Of the above, AMC appears to have the best performance once overheads are taken into account [5], [19]. It has formed the basis of a number of further studies into scheduling mixed criticality systems [32], [33], [13].

For standard fixed priority scheduling of normal (single criticality) task sets preemptive, non-preemptive and deferred preemption dispatching behaviours have been studied. With fully preemptive scheduling, a task switch occurs immediately there is a release of a task with higher priority than the currently executing task. With fully non-preemptive scheduling, the higher priority task will only run when the lower priority task completes (or delays in some other way). With deferred preemption (or cooperative scheduling) [12] preemption is only allowed at specified times or points in the task's execution. The released high priority task must wait until the next preemption point before it can preempt. The preemption times are usually controlled by the RTOS.

A form of deferred preemption, in which a task has a final non-preemptive region (FNPR), was explored by Davis and Bertogna [15] in 2012. They showed that this approach dominates both fully preemptive and fully non-preemptive scheduling. They derived a means of optimally determining both the length of the final non-preemptive regions and the priorities of the tasks. This work is reviewed in Section III.

In this paper we combine the AMC scheme [5] for mixed criticality scheduling with dispatching based on deferred preemption with final non-preemptive regions [15]. The integration of the two approaches takes a particular form that improves schedulability for both LO-criticality and HI-criticality tasks in the LO-criticality mode, and HI-criticality tasks in the HI-criticality mode. The new scheme, AMC-NPR (AMC with non-preemptive regions), is shown to dominate AMC, and to significantly outperform existing schemes based on fixed priorities for a wide range of system parameters.

A. Illustrative Example

We now give a simple motivational example to illustrate the AMC-NPR protocol.

Consider the two task system depicted in the following table. Each task's deadline is equal to its period. As the overall utilisation of this task set ($C_1(LO)/T_1 + C_2(HI)/T_2 = 2/4 + 14/20$) is greater than 1 then no simple scheme, such as CrMPO, SMC, SMC-NO or PT, can schedule the system. A scheme is required that drops work when the HI-criticality task needs more than 7 time units of computation. As there is only one HI-criticality task, nothing can be gained from changing its priority after the criticality mode change therefore the TwoP scheme has no advantage over AMC.

τ_i	L	$C_i(\text{LO})$	$C_i(\text{HI})$	$D_i = T_i$
τ_1	LO	2	-	4
τ_2	HI	7	14	20

Considering the AMC scheme, its behaviour is shown in Figure 1. In the worst case, task τ_2 requires 15 time units before it has executed for its $C(LO)$ computation time of 7. If it then signals a mode change because it has not completed, it will run for a further 7 time units, completing at time 22 which is after its deadline at time 20. A worst-case response time of 22 is short, but it is not short enough to meet the task's deadline.

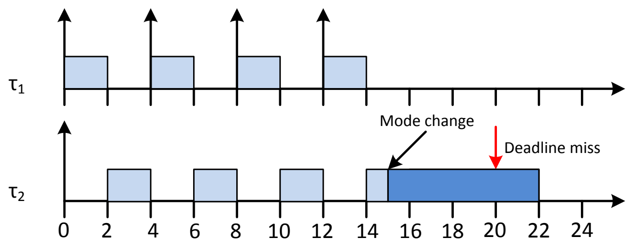

Fig. 1. Execution under AMC

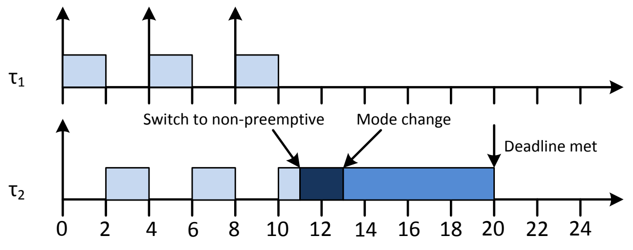

Fig. 2. Execution under AMC-NPR

Under the AMC-NPR scheme introduced in this paper, task τ_2 can be given a non-preemptive region at the end of its LO-criticality computation. Its dispatching is defined to be $(5, \bar{2}, 7)$; that is, 5 time units of preemptive execution followed by a region of 2 time units which are non-preemptive, followed by a further 7 time units of preemptive execution. We will later show that this final block of HI-criticality execution can also benefit from having its own final non-preemptive region, but that is not needed in this example.

The use of deferred preemption introduces blocking even when tasks are independent. In our example a non-preemptive region of 2 time units in task in τ_2 can lead to blocking of τ_1 of at most 2 time units. As this task has a computation time of 2 and a deadline of 4 it can tolerate this blocking. Figure 2 shows the worst-case behaviour of task τ_2 under AMC-NPR. It executes for 5 time units by time 11, and then enters its final non-preemptive region (in LO-criticality mode) and executes for a further 2 time units. If it completes then both tasks would continue unabated; however, if this job of τ_2 is not complete after 7 time units of execution, then it will induce a mode change and run for a further 7 time units, completing at time 20, and thereby meeting its deadline.

B. Organisation

The remainder of the paper is organised as follows. The system model is outlined in Section II, with related work discussed in Section III. The AMC-NPR scheduling policy is defined and analysed in Section IV. An outline of how it can be extended to deal with multiple criticality levels is given in Section V. Section VI provides an empirical evaluation of the performance of the AMC-NPR scheme. Finally, Section

VII concludes with a summary of the main contributions of the paper.

II. System Model

A system is defined as a finite set of N sporadic tasks which execute on a single processor. 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), 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). These parameters are however not independent, in particular the worst-case execution time, C_i, is assumed to be derived by a process dictated by the criticality level. The higher the criticality level, the more conservative the verification process and hence the greater will be the value of C_i. We restrict ourselves to constrained deadline systems in which $D \leq T$ for all tasks. At run-time a job (single invocation of a task) will have fixed values of T, D and C. Its actual computation time is however unknown at the time it is released for execution.

In an MCS further information is needed in order to undertake schedulability analysis. In general a task is now 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 $L2$.

During different runs, any given system will, in general, exhibit different *behaviours*. We define the *criticality level of a behaviour* to be the lowest criticality level L such that no job executed for more than its $C(L)$ value.

From the perspective of static verification, the correctness criterion for an algorithm used to scheduling mixed-criticality task systems is as follows: for each criticality level L, *all jobs of all tasks with criticality* $\geq L$ *must complete by their deadlines in any criticality-L behaviour*.

For a dual criticality system, with criticality levels LO and HI (with LO < HI) the correctness criterion implies that all jobs must meet their deadlines at criticality level LO, and all HI-criticality jobs must meet their deadlines at criticality level HI. At run-time the system may move from level LO to level HI. This is referred to as a *criticality mode change*, with the system said to move from the LO-criticality mode to the HI-criticality mode.

A system starts in the LO-criticality mode. If it moves to the HI-criticality mode then it may subsequently return to the LO-criticality mode. This transition can, however, only occur if all newly released LO-criticality jobs can be guaranteed. This issue, of reimposing the pre-conditions on the system for LO-criticality behaviour, is essentially orthogonal to the topics addressed in this paper, and is therefore not considered further here (it is discussed in [3], [21], [22], [30], [29], [23], [14]).

In general, one scheduling algorithm is said to *dominate* another if it can schedule all of the task sets that the other can schedule. It is said to *strictly dominate* if it can schedule some additional task sets that the other algorithm cannot.

III. Related work

In this section we first review the analysis for AMC. We then give an overview of the analysis for deferred preemption. Both analyses have their origins in standard Response Time Analysis (RTA) for fixed priority preemptive scheduling [24], [2]. With RTA one first computes the worst-case response time R_i for each task τ_i and then compares this value with the task's deadline D_i (i.e. test for $R_i \leq D_i$ for all tasks τ_i). For sporadic tasks with constrained deadlines, the worst-case response time is obtained as follows:

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

where $\mathbf{hp}(i)$ denotes the set of tasks with priority higher than that of task τ_i. This equation can be solved using standard techniques for solving recurrence relations (i.e. fixed point iteration). We note that convergence may be speeded up using the techniques described in [17].

A. AMC Analysis

In the paper that introduced AMC [5], two forms of analysis were provided. Here we use the simpler form (AMC Method 1 or AMC-rtb) as it has been shown to give a good approximation to the more complex approach which becomes computationally expensive for higher numbers of criticality levels [19].

The analysis for AMC first computes the worst-case response times for each task τ_i in the LO-criticality mode (denoted by $R_i(LO)$). This is accomplished by solving the following response-time equation for each task:

$$R_i(LO) \ = \ C_i(LO) \ + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{2}$$

During the criticality change we are only concerned with HI-criticality tasks, so for these tasks:

$$R_i(HI) \ = \ C_i(HI) \ + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \ + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(HI)}{T_k} \right\rceil C_k(LO) \tag{3}$$

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

Equation (3) is conservative for AMC as it does not take 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 by $R_i(LO)$ if task τ_i has not yet completed, and hence (3) can be modified as follows:

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

$$\sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \quad (4)$$

which caps the interference from LO-criticality tasks since $R_i(HI)$ cannot be smaller than $R_i(LO)$.

Finally we note that if, for any HI-criticality task, $R_i(HI) \leq D_i$ during the transition to the HI-criticality mode then this task will remain schedulable once the HI-critically mode is fully established, and there is no interference from LO-criticality tasks.

B. Deferred Preemption

Allowing tasks to have a final non-preemptive region (FNPR) improves schedulability by balancing two factors:

- Less interference for low priority tasks
- Increased blocking for high priority tasks

There is less interference as the release of a higher priority job during the final non-preemptive region will not lead to preemption and an extended response time, as it would with fully preemptive scheduling. Most high priority tasks can cope with some blocking, so as long as they are not rendered unschedulable then this blocking time can be used to allow regions of non-preemption in lower priority tasks.

In 2012, Davis and Bertogna [15] provided analysis for tasks with FNPRs, and introduced an optimal algorithm for assigning both task priorities and FNPR lengths. (Their algorithm is optimal in the sense that it is guaranteed to find a combination of FNPR lengths and priorities that leads to a schedulable system if such a combination exists). First, we summarise the parameter assignment algorithm. This consists of three steps (and follows the framework of Audsley's algorithm [1]):

1) Starting at the lowest priority level, identify the set of tasks that can be made schedulable at this priority level and note the size of the minimum FNPR that each task requires in order to be schedulable, assuming that all other unassigned tasks are given higher priorities.
2) Choose the task that can be made schedulable with the shortest FNPR and assign it to this priority level.
3) Move to the next highest priority level.

If at any priority level no task can be made schedulable via a FNPR then the task set is unschedulable. By always choosing the minimum FNPR the blocking impact on high priority tasks is minimised. If schedulable tasks can be found at each priority level with a FNPR of length one, then that equates to a fully preemptive system, since preemption cannot take place within a single processor clock cycle.

To find the minimum FNPR length a binary search is performed using the scheduling equations (given below). If a task is schedulable with a FNPR equal to one time unit then that is the minimum. If a task cannot be scheduled with a FNPR equal to its full computation time C then that task is not a candidate for this particular priority level. A minimum value in the range 1 to C is found for all other tasks.

With preemptive scheduling, the first job released at the critical instance has the longest response time. Hence (1) can be used to compute the worst-case response time. The introduction of a FNPR complicates the analysis somewhat as this is no longer the case. The non-preemptive region can push high priority work into the next invocation of the task, which can result in this second job having a longer response time. Similarly the second job can impact on the third job and so on. We note that this effect can occur even when all of the tasks have constrained deadlines as shown in [16]. In general, all jobs in the busy-period at the task's priority level need to be checked to see which gives the largest response time. The analysis used by Davis and Bertogna [15] follows the approach introduced by Bril [10], [11].

The length of the FNPR for task τ_i is denoted by F_i. The worst-case start time $R^s_{i,0}$ for the first job of τ_i to *begin* its final non-preemptive region is given by (5). Note the use of floor plus one rather than ceiling in this equation. This is to cater for higher priority tasks that are released at the same point in time that τ_i has completed execution of $C_i - F_i$. These tasks need to interfere as τ_i has not yet started its final region; however, once it has started its FNPR, then it cannot be preempted and instead can induce a blocking effect on higher priority tasks of at most $F_i - 1$. By convention this 'first' job is given the index 0 – so we are computing the worst-case start time of the final non-preemptive region of the 0th job of task τ_i:

$$R^s_{i,0} \; = \; B_i \; + \; (C_i - F_i) \; + \sum_{\tau_j \in \mathbf{hp}(i)} (\left\lfloor \frac{R^s_{i,0}}{T_j} \right\rfloor + 1) C_j \quad (5)$$

where the blocking term B_i is given by:

$$B_i \; = \; \max_{\tau_k \in \mathbf{lp}(i)} (F_k - 1) \quad (6)$$

where $\mathbf{lp}(i)$ is the set of tasks with priorities lower than that of task τ_i.

Once $R^s_{i,0}$ has been computed via fixed point iteration, the worst-case response time can be found: $R_{i,0} = R^s_{i,0} + F_i$. If a value of $R^s_{i,0} > D_i - F_i$ is computed then iteration stops and the task is deemed unschedulable (for this value of F_i).

Subsequent jobs of τ_i can be analysed by generalising (5). For the gth job:

$$R^s_{i,g} \; = \; B_i + (g{+}1)C_i {-} F_i + \sum_{\tau_j \in \mathbf{hp}(i)} (\left\lfloor \frac{R^s_{i,g}}{T_j} \right\rfloor + 1) C_j \quad (7)$$

with

$$R_{i,g} = R^s_{i,1} + F_i - gT_i \quad (8)$$

The overall worst-case response time, R_i, is the maximum of the response times $R_{i,g}$ for all of the jobs in the longest priority level-i busy period:

$$R_i \; = \; \max_{g=0 \ldots G_i - 1} (R_{i,g}) \quad (9)$$

The number of jobs to check, G_i, is obtained from the length of the priority level-i busy period, which can be obtained by solving (10), and using this value in (11).

$$V_i \; = \; B_i \; + \sum_{\tau_j \in \mathbf{hep}(i)} \left\lceil \frac{V_i}{T_j} \right\rceil C_j \quad (10)$$

where $\mathbf{hep(i)}$ is the set of tasks with priorities higher than or equal to that of task τ_i.

$$G_i = \left\lceil \frac{V_i}{T_i} \right\rceil \tag{11}$$

Hence g takes the values $0, 1, \ldots, G_i - 1$.

IV. AMC-NPR

For a dual criticality system we can improve schedulability (for AMC) by allowing a FNPR at each criticality level. So for all tasks, $C(LO)$ is split into a region of length $C(LO) - F(LO)$ which may execute preemptively or be split into non-preemptive regions of length no greater than $F(LO)$, and a final non-preemptive region of length $F(LO)$. For HI-criticality tasks; execution between $C(LO)$ and $C(HI)$ is similarly divided into a region of length $C(HI) - C(LO) - F(HI)$ which may execute preemptively or be split into non-preemptive regions of length no greater than $F(HI)$, and a final non-preemptive region of length $F(HI)$. Note if $C(HI) = C(LO)$ then there is a single FNPR. In this paper we fix $F(HI)$ such that $F(HI) = F(LO)$ if $C(HI) - C(LO) \geq F(LO)$ or $C(HI) = C(LO)$, with $F(HI) = C(HI) - C(LO)$ otherwise. Hence if $F(LO) \geq C(HI) - C(LO) > 0$ then there are two adjacent non-preemptive regions of length $F(LO)$ and $F(HI)$ respectively, with preemption permitted at the boundary between them. The switch to non-preemptive execution is assumed to be managed by the RTOS as part of its execution time monitoring and control functions.

The introduction of $F(LO)$ can significantly improve the schedulability of the LO-criticality mode which is the normal mode of the system [15]. Moreover, for a schedulable system it can allow the execution time budgets based on $C(LO)$ to be increased thereby reducing the likelihood of the system switching to the HI-criticality mode. This is done by applying sensitivity analysis [26], [9] to the $C(LO)$ values of HI-criticality tasks, increasing them until the system is only just schedulable.

A. AMC-NPR scheduling policy

The AMC-NPR scheduling policy is essentially the same as AMC, but with deferred preemption:

- The system starts in the LO-criticality mode, and remains in that mode as long as all jobs execute within their LO-criticality computation times ($C(LO)$).
- All LO-criticality tasks are constrained by run-time monitoring and enforcement to never execute for more than their $C(LO)$ values.
- If any HI-criticality job executes for its $C(LO)$ execution time without completing then the system immediately moves to the HI-criticality mode.
- As the system moves to the HI-criticality mode all LO-criticality tasks that have not started executing are abandoned. LO-criticality jobs that have started are allowed to complete, although they are no longer guaranteed to meet their deadlines. No further LO-criticality jobs are executed.
- The system remains in the HI-criticality mode until an idle tick is encountered, at that time a return to the LO-criticality mode can be undertaken.

The final step is not fundamental to AMC-NPR, any safe protocol for changing back to the LO mode is acceptable.

B. Schedulability Analysis for AMC-NPR

In this section, we provide sufficient schedulability analysis for AMC-NPR. We initially assume that priorities have been assigned and $F(LO)$ values are given. In the following subsection we show how to optimise the choice of these parameters.

To determine worst-case response times for the LO-criticality mode we replace the AMC equation (i.e. (2)) with the framework given by the series of equations (5) to (11). Hence, for the LO-criticality mode, (7) becomes:

$$R^s_{i,g}(LO) = B_i(LO) + (g+1)C_i(LO) - F_i(LO) + \sum_{\tau_j \in \mathbf{hp}(i)} \left(\left\lfloor \frac{R^s_{i,g}(LO)}{T_j} \right\rfloor + 1 \right) C_j(LO) \tag{12}$$

where $B_i(LO)$ is given by:

$$B_i(LO) = \max_{\tau_j \in \mathbf{lp}(i)} (F_j(LO) - 1) \tag{13}$$

Equations (8) and (9) become:

$$R_i(LO) = \max_{g=0 \ldots G_i(LO)-1} (R^s_{i,g}(LO) + F_i(LO) - gT_i) \tag{14}$$

with the value of $G_i(LO)$ found by computing the longest possible priority level-i busy period using $C(LO)$ values in (10) and (11).

Once all the $R_i(LO)$ values have been computed, and schedulability proven for the LO-criticality mode, it is then necessary to check the HI-criticality mode and the transition to the HI-criticality mode.

Recall that a task can have a number of non-preemptive regions provided than none of them are longer than $F(LO)$. Only one such region in any lower priority task can actually cause blocking for task τ_i. In the HI-criticality mode, tasks need to be able to deal with blocking of duration $F(LO)$ from any lower priority task. It is therefore possible for a task to also have a FNPR for its HI-criticality execution of a length $F(HI)$ that is no greater than $F(LO)$ without any increase in blocking or any reduction in schedulability.

To compute the worst case start time $R^s_i(HI)$ of the FNPR $F_i(HI)$ of HI-criticality task τ_i in the HI-criticality mode, we need to consider a number of different scenarios. Each scenario corresponds to one of the $g = 0 \ldots G_i(LO) - 1$ jobs of task τ_i that can execute during the longest priority level-i LO-criticality busy period. Since any of these jobs could have the worst-case LO-criticality response time, it follows that we must examine each of them, and the subsequent evolution of the busy period, to determine the worst-case HI-criticality response time of task τ_i. We now use g to indicate both the

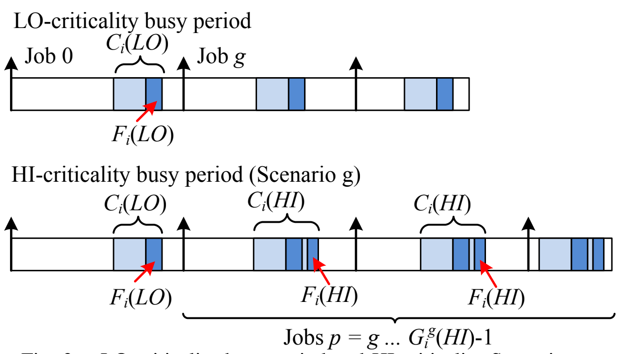
LO-criticality busy period
Job 0
$C_i(LO)$
Job g
$F_i(LO)$
HI-criticality busy period (Scenario g)
$C_i(LO)$
$C_i(HI)$
$C_i(HI)$
$F_i(LO)$
$F_i(HI)$
$F_i(HI)$
Jobs $p = g \ldots G_i^g(HI)$-1

Fig. 3. LO-criticality busy period and HI-criticality Scenario g.

scenario and the index of the job of task τ_i in the LO-criticality busy period. Figure 3 illustrates both the LO-criticality busy period and one corresponding HI-criticality busy period, that is scenario g.

There are $G_i(LO)$ HI-criticality scenarios we need to consider. For each of these scenarios $g = 0 \ldots G_i(LO) - 1$, we assume that job g of task τ_i and all subsequent jobs of that task execute for $C_i(HI)$, while any prior jobs of task τ_i execute for $C_i(LO)$. Note this behaviour precludes LO-criticality jobs of other tasks being released after the worst-case start time ($R^s_{i,g}(LO)$) of the non preemptive region that ends the LO-criticality execution of job g. Similar to the analysis of AMC, we pessimistically assume that all jobs of higher priority, HI-criticality tasks in the busy period may execute for their $C(HI)$ computation times. (This assumption greatly simplifies the analysis, but makes it sufficient rather than exact). Since the HI-criticality execution of each job of task τ_i ends with a FNPR ($F_i(HI)$), then this may have a push-through effect causing the next job of the task to have a longer response time. Hence to determine the worst-case HI-criticality response time for each scenario we must determine the response time of job g and *also* the response times of all subsequent jobs of task τ_i to the end of the new busy period (which may well be longer than the LO-criticality one, as illustrated in Figure 3).

The length of the longest priority level-i busy period $G_i^g(HI)$ corresponding to scenario g is obtained as follow:

$$V_{i,g} = B_i + gC_i(LO) + \max(0, \left\lceil \frac{V_{i,g}}{T_i} \right\rceil - g)C_i(HI) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{V_{i,g}}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R^s_{i,g}(LO)}{T_k} \right\rceil C_k(LO)$$

and

$$G_i^g(HI) = \left\lceil \frac{V_{i,g}}{T_i} \right\rceil \tag{15}$$

For each scenario g, we must compute the response time of each job from job g to the last job in the busy period. We use the index p for this purpose ($p = g \ldots G_i^g(HI) - 1$), with the worst-case start time of the HI-criticality FNPR of job p in scenario g given by $R^s_{i,p,g}(HI)$:

$$R^s_{i,g,p}(HI) = B_i + gC_i(LO) + (p+1-g)C_i(HI) - F_i(HI) + \sum_{\tau_j \in \mathbf{hpH}(i)} (\left\lfloor \frac{R^s_{i,g,p}(HI)}{T_j} \right\rfloor + 1)\ C_j(HI) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R^s_{i,g}(LO)}{T_k} \right\rceil C_k(LO) \tag{16}$$

with $B_i = B_i(LO)$ since $F(HI) \leq F(LO)$.

Finally, we determine the worst-case HI-criticality response time of task τ_i across all jobs and all scenarios:

$$R_{i,g,p}(HI) = R^s_{i,g,p}(HI) + F_i(HI) - pT_i$$

$$R_i(HI) = \max_{g=0 \ldots G_i(LO)-1} \left\{ \max_{p=g \ldots G_i^g(HI)-1} (R_{i,g,p}(HI)) \right\}$$

C. Priority and FNPR length assignment

In the previous subsection, we presented analysis for the AMC-NPR scheduling policy, assuming that priorities and FNPR lengths were given. In this section, we show how priorities and FNPR lengths can be assigned using the greedy FNR-PA algorithm introduced by Davis and Bertogna [15].

Recall that in our model, the final non-preemptive regions of LO- and HI-criticality execution of a task (i.e. $F(LO)$ and $F(HI)$) are dependent, with $F(HI)$ fully determined by $F(LO)$, such that $F(HI) = F(LO)$ if $C(HI) - C(LO) \geq F(LO)$ or $C(HI) = C(LO)$, otherwise $F(HI) = C(HI) - C(LO)$. This ensures that the HI-criticality blocking never exceeds the LO-criticality blocking, and thus allows the HI-criticality FNPRs to be added with impunity, since there is no penalty in terms of schedulability. Subject to the restriction that $F(HI) \leq F(LO)$, then by utilising the greedy FNR-PA algorithm of Davis and Bertogna [15], we can obtain a *weakly optimal* assignment of $F(LO)$ (and hence $F(HI)$) as well as task priorities. By *weakly optimal*, we mean that the algorithm is guaranteed to find a set of parameters that make the system schedulable according to the sufficient schedulability analysis (given in the previous subsection) whenever such a set of parameters exists with $F(HI) \leq F(LO)$.

Starting with a full set of N unassigned tasks (both LO- and HI-criticality); for each of the N priority levels, starting at the lowest, the modified FNR-PA algorithm proceeds as follows:

1) For each unassigned task, use a binary search to determine the minimum F (if one exists) between the values 1 and $C(HI)$ such that the task is schedulable (according to the sufficient test given in the previous section) with $F(LO)$ set to the minimum of $C(LO)$ and F; and, if it is a HI-criticality task, $F(HI)$ set to the minimum of $C(HI) - C(LO)$ and $F(LO)$ (or simply set to $F(LO)$ if $C(LO) = C(HI)$.
2) If no such task exists, declare the task set *unschedulable*.
3) Of the schedulable tasks found in the first step, choose the one with the smallest F and assign it to this priority level;

if there is more than one task with the same minimal F, chose a LO- over a HI-criticality task; otherwise break ties arbitrarily.

With the restriction that $F(HI) \leq F(LO)$, proof that the modified FNR-PA algorithm achieves weakly optimal priority and FNPR length assignment with respect to the sufficient schedulability analysis for AMC-NPR can be proven via the method given in [15]. In particular, since $F(LO)$ and $F(HI)$ are tied together in our model and $F(LO) \geq F(HI)$, we need only consider the set of $F(LO)$ values. (Each $F(HI)$ value is determined directly from the corresponding $F(LO)$ and *crucially* only impacts the schedulability of the specific task it applies to; it does not add anything beyond the contribution of $F(LO)$ to the blocking of higher priority tasks).

To prove optimality, one may observe that:

(i) Corollaries 2 to 5 from [15] apply to the analysis of AMC-NPR, substituting "AMC-NPR" for " "FPDS" and "$F(LO)$" for "$F(k)$" in the text.

(ii) The proof of Theorem 2 in [15] holds with the set of $F(LO)$ values substituted for the set of final non-preemptive regions, and where necessary, noting that the interference due to the computation time of a task exceeds its final non-preemptive region length, since $C(LO) \geq F(LO)$.

In general, tasks could potentially have separate, independent values for $F(LO)$ and $F(HI)$, with the possibility that $F(LO) < F(HI)$. In this case, additional analysis would be required to account for the fact that tasks could be blocked for longer during HI-criticality only operation. In this general case, it is unlikely that a greedy algorithm based on Audsley's approach could be used to determine an optimal parameter assignment. This is because of the trade off between $F(LO)$ and $F(HI)$. Consider assigning a task at a given priority level (it may be the only task that is schedulable at that priority), the task could be assigned a small value of $F(LO)$, but need a large value of $F(HI) >> F(LO)$ to be schedulable in the HI-criticality mode. Alternatively, with a larger value of $F(LO)$ it could require a much smaller $F(HI)$ value. It is not possible to know which option is best without knowing the relative priority order of the unassigned (higher priority) tasks, since it may be that a LO-criticality task cannot tolerate a large $F(LO)$ or a HI-criticality task cannot tolerate a large $F(HI)$. This would seem to exclude a greedy bottom-up approach to finding an optimal solution.

Despite the simple restriction of our model ($F(HI) \leq F(LO)$), AMC-NPR strictly dominates AMC, as is evident from the modified FNR-PA algorithm. If a task set was schedulable with AMC then at each priority level there would be a task that is schedulable in both modes with $F = 1$ (and hence is schedulable with AMC-NPR). The example in Section I-A shows that there are task sets that are schedulable with AMC-NPR that are not schedulable with AMC. Hence AMC-NPR strictly dominates AMC. Further, the AMC-NPR policy is in practice significantly more effective than AMC as shown in Section VI.

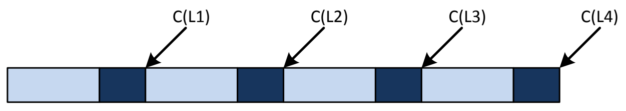

Fig. 4. Multiple Criticality Levels with AMC-NPR

V. Multiple Criticality Levels

In this section, we sketch how the AMC-NPR scheme for two criticality levels extends and scales to an increased number of levels. Many standards have up to five levels, and it is therefore necessary for any proposed scheme to permit extension to at least this number.

For M criticality levels, each task with criticality level L (L ranges from $L1$ to LM) has up to L computations times and each of these blocks of execution may have a FNPR. Figure 4 illustrates the code of a task with each interval of execution ($C(L1)$ to $C(L4)$) having a final non-preemptive region of the same duration.

The analysis is easily extended to 4 or more levels as is the parameter assignment method. The analysis for extending AMC is given by Fleming and Burns [19]. The basic equations are first solved to find $R(L1)$ for all tasks. This is used to compute $R(L2)$ for all but $L1$ tasks. Each level is addressed in this way until eventually $R(L4)$ is computed for the highest criticality tasks. At each step the interference from higher priority, but lower criticality tasks is capped. So for task τ_i from L4, the interval over which $L1$ tasks can interfere is capped at $R_i(L1)$, for $L2$ tasks the cap is $R_i(L2)$ and for $L3$ tasks the cap is $R_i(L3)$. Applying this approach to AMC-NPR, the basic equations are replaced by the framework defined by equations (12) to (16), with multiple levels of busy periods examined.

The parameter allocation algorithm is a straightforward generalisation of the FNR-PA algorithm described previously. For a task to be assigned a specific priority level it must, with the help of deferred preemption, be schedulable at each criticality level up to and including its own criticality level. Again we attempt to have the same sized FNPR for each criticality level, but it cannot be larger than the available computation time. The algorithm is therefore:

1) For each unassigned task, compute a minimum F (if one exists) between the values 1 and $C(Lm)$ such that the task is schedulable with $F(L1)$ set to the minimum of $C(L1)$ and F; and for each criticality level (Li) from $L2$ up to the level of the task, $F(Li)$ is set to the minimum of $C(Li) - C(Li-1)$ and F.
2) If no such task exists, declare the task set *unschedulable*.
3) Of the schedulable tasks found in the first step, choose the one with the smallest F and assign it to this priority level; if there is more than one task with the same minimum F, break ties arbitrarily.

VI. Empirical Evaluations

In this section, we present an empirical investigation, examining the effectiveness of our analysis techniques and the AMC-NPR scheme itself. We report on a set of experiments undertaken for two criticality levels. These are sufficient to provide a clear evaluation. For high numbers of criticality levels the parameter space grows considerably and it becomes harder to provide a comprehensive study.

A. Task set parameter generation

The task set parameters used in our experiments were randomly generated as follows:

- Task utilisations ($U_i = C_i/T_i$) were generated using the UUnifast algorithm [8], giving an unbiased distribution of utilisation values.
- Task periods were generated according to a log-uniform distribution. By default, the range of task periods was one order of magnitude $r = 1$.
- Task deadlines were set equal to their periods.
- The LO-criticality execution time of each task was set based on the utilisation and period: $C_i(LO) = U_i/T_i$.
- The HI-criticality execution time of each task was a fixed multiplier of the LO-criticality execution time, $C_i(HI) = CF \cdot C_i(LO)$ (e.g., $CF = 2.0$).
- The probability that a generated task was a HI-criticality task was given by the parameter CP (e.g. $CP = 0.5$).

B. Schedulability tests investigated

We investigated the performance of the following techniques and associated schedulability tests.

- AMC-NPR: the scheme introduced in this paper.
- AMC-rtb: published in [5] and described in Section III-A.
- SMC: the approach published in [3].
- SMC-NO: the SMC scheme without run-time monitoring (i.e. the approach published by Vestal [31]).
- CrMPO: Criticality Monotonic Priority Ordering. Task priorities were ordered first according to criticality (HI-criticality first) and then according to deadline (shortest deadline first). Response time analysis was then used to determine if the task set was schedulable with HI-criticality tasks assumed to execute for $C(HI)$ and LO-criticality tasks for $C(LO)$.

In addition, in the Figures we include two further lines:

- Valid: Task sets pass this 'test' if the sum of their LO-criticality utilisation and the sum of their HI-criticality utilisation are both no greater than one.
- UB-NPR: A composite upper bound on any fixed priority deferred preemption mixed-criticality scheduling technique. Task sets pass this 'test' if they are independently schedulable in each of the LO-criticality and HI-criticality modes according to the optimal priority and FNPR assignment algorithm given by Davis and Bertogna [15]; but no check on the mode change is made. Note that UB-NPR is a necessary schedulability condition for any valid fixed priority scheduling algorithm with or without non-preemptive regions. It is included to illustrate the effectiveness, or otherwise, of the other schemes.

With non-mixed criticality systems there is no need to consider validity as there is a clear utilisation upper bound of 1; however, with mixed criticality $\sum(C(LO)/T)$ for LO-criticality tasks + $\sum(C(HI)/T)$ for HI-criticality tasks can be greater than 1 for schedulable systems [19], hence to correctly interpret the results presented it is necessary to know what proportion of the considered task sets could possibly be schedulable. We do this by checking that $\sum(C(LO)/T) \leq 1$ for *all* tasks and $\sum(C(HI)/T) \leq 1$ for all HI-criticality tasks. For extreme parameter values (see the results below) schedulability drops towards zero, but so does the number of valid task sets. It becomes increasingly difficult to generate task sets at these extremes and so rather than scale the results (e.g. to give the percentage of valid task sets that are schedulable) it is more informative to show explicitly on the figures the proportion of generated task sets that are valid.

The conclusion from these considerations is that a task set will only be considered in the experimental evaluation if it is feasible in the LO-criticality mode, i.e. $\sum(C(LO)/T) \leq 1$ for all tasks. This is a necessary but not sufficient requirement for the task set to be deemed valid. Also because the LO-criticality utilisation bound is unambiguous we present all results, in the figures, using LO-criticality utilisation on the x-axis.

The dominance relationships between the algorithms and tests implies that in all figures there is a strict ordering to the lines; from best to worst: Valid, UB-NPR, AMC-NPR, AMC-rtb, SMC, SMC-NO and finally CrMPO. The role of the experiments is to examine the relative performance of the different schemes.

C. Experiments

In our experiments, the task set LO-criticality utilisation was varied from 0.025 to 0.975. For each utilisation value, 1000 task sets were generated and the schedulability of those task sets determined for the different schemes. (Note, the graphs are best viewed online in colour).

Figure 5 plots the percentage of task sets generated that were deemed schedulable for a system of 20 tasks, with on average 50% of those tasks having HI criticality (CP = 0.5) and each task having a HI-criticality execution time that is 2.0 times its LO-criticality execution time (CF = 2.0).

We observe that AMC-NPR has a much improved performance over the original AMC method (AMC-rtb).

In the following figures we show the weighted schedulability measure $W_y(p)$ [7] for schedulability test y as a function of parameter p. For each value of p, this measure combines results for the task sets τ generated for all of a set of equally spaced utilization levels (0.025 to 0.975 in steps of 0.025).

Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for a task set τ with parameter value p:

$$W_y(p) = (\sum_{\forall\tau} u(\tau) \cdot S_y(\tau, p)) / \sum_{\forall\tau} u(\tau) \tag{17}$$

where $u(\tau)$ is the utilization of task set τ.

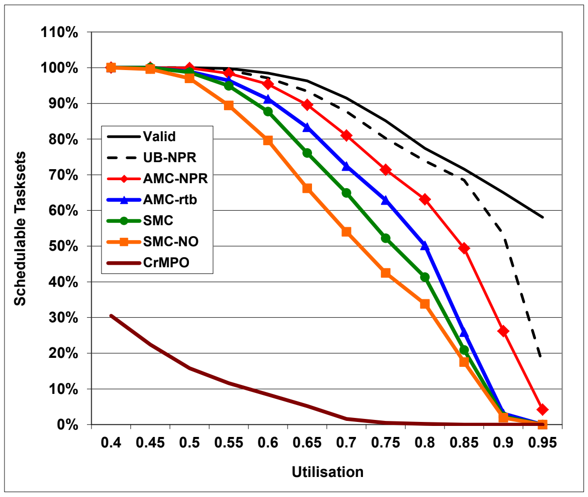

Fig. 5. Percentage of Schedulable Tasksets

The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [7]. Weighting the results by task set utilization reflects the higher value placed on being able to schedule higher utilization tasksets.

Figure 6 varies the criticality factor, Figure 7 varies the percentage of tasks with HI criticality, Figure 8 varies the size of the task set and Figure 9 varies the range of task periods. A number of points are illustrated by these figures:

- CrMPO preforms very badly as the priority ordering it uses is far from optimal.
- Figure 5 shows the differences between AMC-rtb, SMC and SMC-NO (as previously shown by Baruah et al. [5]) and clearly shows the further improvement that AMC-NPR delivers.
- All the weighted schedulability figures show that AMC-NPR out performs AMC over a wide range of parameters. The only exception is when the ratio of task periods is high, AMC-NPR then approaches the behaviour of AMC (as shown in Figure 9). This is due to short period high priority tasks being unable to cope with even small non-preemptive regions in the long period lower priority tasks.
- When the Criticality Factor is close to 1, or the percentage of HI-criticality tasks is either very high or very low then the system behaves similarly to a single criticality system. In these situations Figures 6 and 7 show the inherent advantage of deferred preemption.
- Figures 6 and 7 also show that as the number and/or size of the HI-criticality tasks increases it is hard to generate valid task sets with high LO-criticality utilisation. For example, when the percentage of HI-criticality tasks is 95% only 30% of the generated task sets were valid. (The remainder had a HI-criticality utilisation that exceeded 1).

VII. Conclusions

In this paper we presented an extension to the AMC scheduling policy for mixed criticality systems executing on

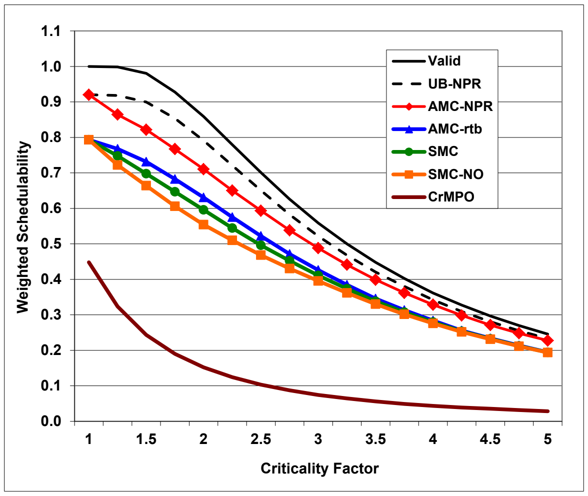

Fig. 6. Varying the Criticality Factor

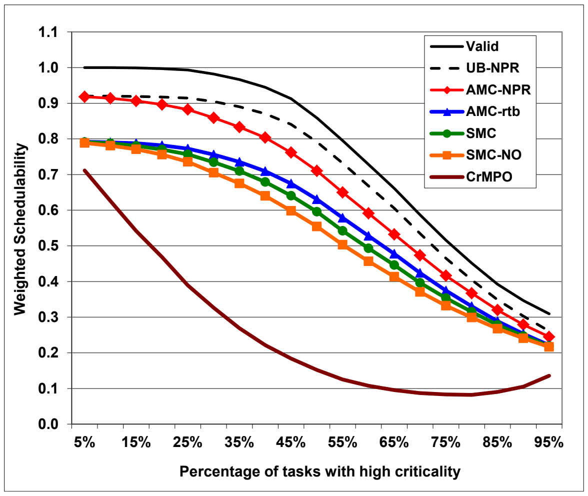

Fig. 7. Varying the Number of HI-criticality Tasks

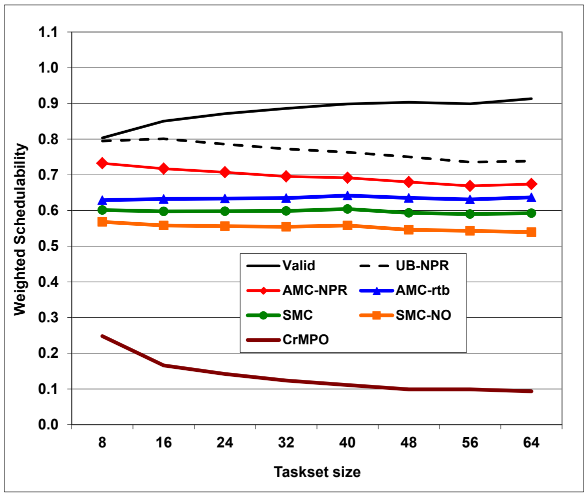

Fig. 8. Varying the Size of the Task set

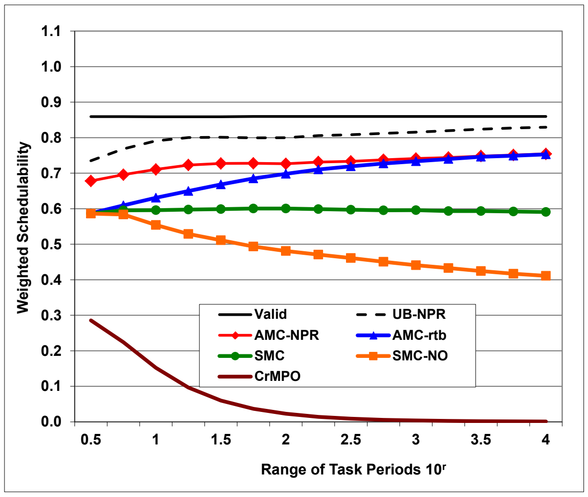

Fig. 9. Varying the Range of Task Periods

a single processor. The extension, which we denote as AMC-NPR, allows tasks to have non-preemptive regions positioned at the ends of the estimated execution times for each criticality level, with this deferred preemption behaviour controlled by the RTOS.

The main contribution of the paper is the introduction of the AMC-NPR scheme and its schedulability analysis. In addition, we were also able to provide an effective means of deriving the lengths of the non-preemptive regions, along with the task priorities. In theory, AMC-NPR strictly dominates AMC, previously the best performing scheme based on fixed priorities. Our evaluations show that in practice, the performance of AMC-NPR is significantly better than that of AMC for a wide range of parameter settings.

Acknowledgements

The research described in this paper is funded, in part, by the ESPRC grant, MCC (EP/K011626/1) and the EU FP7 Integrated Project PROXIMA (611085).

References

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

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

[3] S.K. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky, editor, *Proc. of Reliable Software Technologies - Ada-Europe 2011*, pages 174–188. Springer, 2011.

[4] S.K. Baruah and A. Burns. Fixed-priority scheduling of dual-criticality systems. In *Proc. RTNS*, pages 173–182. ACM, 2013.

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

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

[7] A. Bastoni, B. Brandenburg, and J. Anderson. Cache-related preemption and migration delays: Empirical approximation and impact on schedulability. In *Proc. of Sixth International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*, pages 33–44, 2010.

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

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

[10] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *ECRTS '07: Proc. of the 19th Euromicro Conference on Real-Time Systems*, pages 269–279. IEEE Computer Society, 2007.

[11] R.J. Bril, J.J. Lukkien, and W.F.J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption. *Real-Time Systems*, 42(1-3):63–119, 2009.

[12] A. Burns. Preemptive priority based scheduling: An appropriate engineering approach. In S.H. Son, editor, *Advances in Real-Time Systems*, pages 225–248. Prentice-Hall, 1994.

[13] A. Burns. The application of the original priority ceiling protocol to mixed criticality systems. In L. George and G. Lipari, editors, *Proc. ReTiMiCS, RTCSA*, pages 7–11, 2013.

[14] A. Burns and S. Baruah. Towards a more practical model for mixed criticality systems. In *Proc. WMC, RTSS*, pages 1–6, 2013.

[15] R.I. Davis and M. Bertogna. Optimal fixed priority scheduling with deferred pre-emption. In *Proc. IEEE Real-Time Systems Symposium*, pages 39–50, 2012.

[16] R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Journal of Real-Time Systems*, 35(3):239–272, 2007.

[17] R.I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority pre-emptive systems. *IEEE Transaction on Computers*, 57(9):1261–1276, 2008.

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

[19] T. Fleming and A. Burns. Extending mixed criticality scheduling. In *Proc. WMC, RTSS*, pages 7–12, 2013.

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

[21] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. Technical Report 350, ETH Zurich, Laboratory TIK, 2013.

[22] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele. Service adaptions for mixed-criticality systems. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Singapore, Jan 2014.

[23] M. Jan, L. Zaourar, and M. Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. In *Proc. WMC, RTSS*, pages 43–48, 2013.

[24] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Computer Journal*, 29(5):390–395, 1986.

[25] K. Lakshmanan, D. de Niz, and R. Rajkumar. Mixed-criticality task synchronization in zero-slack scheduling. In *IEEE RTAS*, pages 47–56, 2011.

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

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

[28] L. Sha, J.P. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritizing preemptive scheduling. In *Proc. 7th IEEE Real-Time Sytems Symposium*, 1986.

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

[30] H. Su, D. Zhu, and D. Mosse. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems. In *Proc. RTCSA*, 2013.

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

[32] Q. Zhao, Z. Gu, and H. Zeng. Integration of resource synchronization and preemption-thresholds into EDF-based mixed-criticality scheduling algorithm. In *Proc. RTCSA*, 2013.

[33] Q. Zhao, Z. Gu, and H. Zeng. PT-AMC: Integrating preemption thresholds into mixed-criticality scheduling. In *Proc. DATE*, pages 141–146, 2013.