# Response Time Analysis for Mixed Criticality Systems with Arbitrary Deadlines

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

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

***Abstract*—This paper extends analysis of the Adaptive Mixed Criticality (AMC) scheme for fixed-priority preemptive scheduling of mixed-criticality systems to include tasks with arbitrary deadlines. Both of the previously published schedulability tests, AMC-rtb and AMC-max are extended to cater for tasks with deadlines that may be greater than their periods. Evaluations show that the simpler method, AMC-rtb-Arb, remains a viable approach that performs almost as well as the more complex alternative, AMC-max-Arb, when tasks with arbitrary deadlines are considered.**

## I. Introduction

Since the publication of Vestal's model [14] there has been a significant number of papers on the scheduling of Mixed Criticality Systems (MCS) (see [8] for a survey). Somewhat surprisingly none of the papers on fixed priority scheduling of MCS have addressed tasks with so called *arbitrary deadlines* that may be greater than their periods. In this paper we consider this issue within the context of fixed-priority preemptive systems scheduled according to the Static Mixed Criticality (SMC) [4] and Adaptive Mixed Criticality (AMC) [5], [1] schemes.

Arbitrary-deadline tasks cater for situations where there is some leeway in when a task must execute. For example, a consumer task that reads items from a buffer must, over a long time interval, consume items at the same rate as they are produced; however, the task can have response times that are longer than its period or some multiple of its period, provided that buffer has sufficient space to store unread items. A task set may be unschedulable if its deadlines are constrained (i.e. less than or equal to their periods), but may meet all of its time constraints if a few tasks are allowed to have arbitrary deadlines.

The remainder of the paper is organized as follows. In Section II, we outline the standard MCS model. In Section III, we recap on the extended form of response-time analysis for arbitrary-deadline tasks. Section V then reviews existing schedulability analysis for SMC and AMC, assuming tasks with constrained deadlines. The main contribution of the paper is the new response-time analyses derived in Section VI for the AMC scheme, catering for arbitrary-deadline task sets. Section VII discusses priority assignment, while Section VIII evaluates the performance of the various schedulability tests. Section IX concludes.

## II. System Model, Terminology and Notation

In this paper, we are interested in the Fixed Priority Preemptive Scheduling (FPPS) of a single processor MCS comprising a static set of $n$ sporadic tasks. Each task, $\tau_i$, is defined by its period (or minimum inter-arrival time), relative deadline, worst-case execution time (WCET), criticality level, and unique priority: ($T_i$, $D_i$, $C_i$, $L_i$, $P_i$). Task deadlines may be arbitrary, i.e. less than, equal to or greater than their periods.

We assume that each task $\tau_i$ gives rise to a potentially unbounded sequence of jobs, with the release of each job separated by at least the minimum inter-arrival time from the release of the previous job of the same task. The worst-case response time of task $\tau_i$ is denoted by $R_i$ and corresponds to the longest response time from release to completion for any of its jobs. For tasks with arbitrary deadlines, more than one job of the same task may be active at any given time. Among jobs of the same task, those released earlier are executed first.

The system is assumed to be defined over two criticality levels ($HI$ and $LO$). Each LO-criticality task $\tau_j$ is assumed to have a single estimate of its WCET: $C_j(LO)$, while each HI-criticality task $\tau_k$ has two estimates: $C_k(HI)$ and $C_k(LO)$, with $C_k(HI) \geq C_k(LO)$. (Note we drop the task index when using these and other terms in a generic way, but include the index when referring to the parameters of a specific task). Most scheduling approaches for MCS identify different modes of behavior. In the LO-criticality (or normal) mode, all tasks execute within their $C(LO)$ bounds and all deadlines are required to be met. At all times LO-criticality tasks are constrained by run-time monitoring to execute for no more than their $C(LO)$ bound. In contrast, if a HI-criticality task executes for $C(LO)$ without signaling completion then the system enters HI-criticality mode. In this mode only HI-criticality tasks are required to meet their deadlines. HI-criticality tasks are assumed to execute for no more than $C(HI)$. The response time of a task $\tau_i$ in LO-criticality mode is denoted by $R_i(LO)$ and in HI-criticality mode by $R_i(HI)$.

## III. Existing Analysis for FPPS

For constrained-deadline tasks ($D_i \leq T_i$) in a single criticality level system, the response time of task $\tau_i$ can be computed as follows (see [11], [3] for a full derivation):

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

where $\mathbf{hp(i)}$ is the set of tasks with higher priority than $\tau_i$. This and all subsequent response time equations can be solved via fixed point iteration. Iteration starts with a value of $R_i = 0$ and continues either until convergence or until the computed response time exceeds the task's deadline.

For tasks with arbitrary deadlines ($D_i \geq T_i$) then it is possible in a schedulable system that $R_i \geq T_i$. It follows that there can be more than one job of task $\tau_i$ active within the same priority level-$i$ busy period; any of these jobs can give rise to the worst-case response time for the task. We use $q$ as an index to denote each job within the busy period, with $q = 0$ indicating the first job. The completion time of each job of the task, $r_i(q)$ $(0 \leq q \leq p)$, as measured from the start of the busy period, can be computed as follows (see [13] for a full derivation):

$$r_i(q) \ = \ (q+1)C_i \ + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{r_i(q)}{T_j} \right\rceil C_j \tag{2}$$

The last job in the busy period is denoted by $p$, which is the first value where completion of the job occurs before the next release of the task, i.e. where $r_i(p) \leq (p+1)T_i$. The response-time of each job $q$ is calculated as follows:

$$\forall q_{(0 \leq q \leq p)} : \quad R_i(q) = r_i(q) - qT_i \tag{3}$$

With the worst-case response time of the task given by:

$$R_i \ = \max_{\forall q_{(0 \leq q \leq p)}} \{R_i(q)\} \tag{4}$$

This analysis for FPPS scheduling of arbitrary-deadline tasks can be applied to MCS by simply assuming that all HI-criticality tasks have a single execution time of $C(HI)$ and all LO-criticality tasks have a single execution time of $C(LO)$.

## IV. Existing Analysis for SMC

Static Mixed Criticality (SMC) [4] is a simple scheme that extends Vestal's original approach [14] with run-time monitoring. Under SMC, LO-criticality tasks continue to be released and to execute in HI-criticality mode; however, they are not required to meet their deadlines in that mode. (Recall that all tasks are required to meet their deadlines in LO-criticality mode).

Under SMC, the worst-case response times for constrained-deadline tasks may be computed as follows:

$$\begin{aligned} R_i(L_i) \ &= \ C_i(L_i) \\ &+ \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i(L_i)}{T_j} \right\rceil C_j(\min(L_i, L_j)) \end{aligned} \tag{5}$$

We note that the arbitrary-deadline analysis for FPPS given by (2), (3), and (4) can be adapted to SMC by changing $C_i$ to $C_i(L_i)$ and $C_j$ to $C_j(\min(L_i, L_j))$.

## V. Existing Analyses for AMC

With Adaptive Mixed Criticality (AMC) [5], if a HI-criticality task executes for $C(LO)$ without completing, then the system enters HI-criticality mode. AMC differs from SMC in that in HI-criticality mode, previously released jobs of LO-criticality tasks may be completed, but subsequent releases of LO-criticality tasks are not started. Similar to SMC, only HI-criticality tasks are required to be schedulable in HI-criticality mode. Also, in common with SMC, LO-criticality mode may be re-entered when the processor becomes idle.

In the original paper [5] two sufficient schedulability tests were developed for AMC. The first approach, called AMC-rtb (where rtb denotes 'response time bound') takes account of a bound on the duration over which LO-criticality tasks can interfere. The second, more complex approach is called AMC-max; it determines the maximum response time assuming all possible times at which the criticality mode change could occur.

### A. *AMC-rtb Analysis*

The AMC-rtb analysis first computes the worst-case response time for each task $\tau_i$ in the LO-criticality mode:

$$R_i(LO) = C_i(LO) + \sum_{\tau_j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \tag{6}$$

During the mode change and subsequent HI-criticality mode only HI-criticality tasks are required to be schedulable. The worst-case response time of a HI-criticality task $\tau_i$ can be computed as follows:

$$\begin{aligned} R_i(HI) \ = \ C_i(HI) \ + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \ + \\ \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \end{aligned} \tag{7}$$

where $\mathbf{hpH(i)}$ is the set of HI-criticality tasks with higher priority than $\tau_i$, and $\mathbf{hpL(i)}$ is the set of LO-criticality tasks with higher priority than $\tau_i$. Note that higher priority LO-criticality tasks released after $R_i(LO)$ cannot cause interference, since if task $\tau_i$ has not completed by $R_i(LO)$ then the system must have switched to HI-criticality mode.

### B. *AMC-max analysis*

The AMC-rtb analysis given in (7) is potentially pessimistic in that it assumes all jobs of higher priority HI-criticality tasks contribute $C(HI)$, and yet also assumes that the mode change takes place as late as possible at $R(LO)$ allowing jobs of higher priority LO-criticality tasks to contribute interference of $C(LO)$ until that point. This pessimism is avoided by the AMC-max analysis, which assumes that a mode change takes place at some time $s$, measured from the start of the busy period for task $\tau_i$. All jobs of higher priority tasks that complete before that time can cause interference equating to at most their $C(LO)$ values, and after that time, no new jobs of LO-criticality tasks can be released and contribute interference. (Note that the AMC-max analysis for tasks in LO-criticality mode is the same as that for AMC-rtb, i.e. (6)).

AMC-max computes the worst-case response time $R_i^s(HI)$ of HI-criticality task $\tau_i$, assuming a mode change at time $s$, and then takes the maximum of these values over all possible values of $s$.

$$R_i^s(HI) = C_i(HI) + I_L(s) + I_H(s) \tag{8}$$

where $I_L(s)$ is the interference from higher priority LO-criticality tasks, and $I_H(s)$ is the interference from higher priority HI-criticality tasks.

As jobs of higher priority LO-criticality tasks are prevented from being released after the mode change at time $s$, their worst-case interference is upper bounded by:

$$I_L(s) = \sum_{j \in \mathbf{hpL}(i)} \left( \left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) \tag{9}$$

$I_H(s)$ is defined by considering the number of jobs of each higher priority HI-criticality task $\tau_k$ that can execute in a busy period of length $t$, with the mode change taking place at time $s < t$. Only those jobs that may have some part of their execution after time $s$ can contribute interference of $C(HI)$, with the remainder contributing $C(LO)$.

The maximum number of jobs of $\tau_k$ with $D_k \leq T_k$ that can fit into an interval of length $t - s$ is bounded by:

$$\left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1 \tag{10}$$

Equation (10) can be pessimistic; including more jobs than can actually be present in an interval of length $t$. This is taken into account by defining:

$$M(k, s, t) = \min \left\{ \left\lceil \frac{t - s - (T_k - D_k)}{T_k} \right\rceil + 1, \left\lceil \frac{t}{T_k} \right\rceil \right\} \tag{11}$$

where $M(k, s, t)$ is the maximum number of jobs of $\tau_k$ that can exhibit HI-criticality behavior in a busy period of length $t$ with a transition to HI-criticality mode at time $s$. The interference term for higher priority HI-criticality tasks thus becomes:

$$I_H(s) = \sum_{k \in \mathbf{hpH}(i)} \Big\{ M(k, s, t)\ C_k(HI) + \left( \left\lceil \frac{t}{T_k} \right\rceil - M(k, s, t) \right) C_k(LO) \Big\} \tag{12}$$

Hence the worst-case response time with a mode change at time $s$ is given by:

$$R_i^s(HI) = C_i(HI) + \sum_{j \in \mathbf{hpL}(i)} \left( \left\lfloor \frac{s}{T_j} \right\rfloor + 1 \right) C_j(LO) + \sum_{k \in \mathbf{hpH}(i)} \Big\{ M(k, s, R_i^s) C_k(HI) + \left( \left\lceil \frac{t}{T_k} \right\rceil - M(k, s, R_i^s) \right) C_k(LO) \Big\} \tag{13}$$

The worst-case response time of the task is then the maximum over all possible values of $s$:

$$R_i^*(HI) = \max_{\forall s} (R_i^s(HI)) \tag{14}$$

Finally, it is necessary to limit the number of values of $s$ that that are considered from the range of possible values $[0, R_i^{LO}]$. In (13), the $\mathbf{hpL}$ term increases as a step function with increasing values of $s$, while the $\mathbf{hpH}$ term decreases. It follows that $R_i^s(HI)$ can only increase at values of $s$ corresponding to multiples of the periods of LO-criticality tasks – hence these are the only values of $s$ that need to be considered. Note that a LO-criticality job released at exactly $R_i^{LO}$ will not be allowed to execute, and hence $s$ is restricted to the interval $[0, R_i^{LO})$.

## VI. Arbitrary-Deadline Analysis for AMC

In this section, we present new analysis for arbitrary-deadline tasks scheduled according to AMC. The two new methods are referred to as AMC-rtb-Arb and AMC-max-Arb ("Arb" meaning arbitrary deadline). They build on the existing analysis for AMC-rtb, AMC-max [5] and the classical arbitrary-deadline analysis for FPPS [13].

### A. AMC-rtb-Arb Analysis

The strategy behind AMC-rtb is to count interference from each job of a higher priority HI-criticality task $\tau_k$ as $C_k(HI)$, and to assume that each job of a higher priority LO-criticality task $\tau_j$ released up to the LO-criticality response time of the task under analysis $\tau_i$ causes interference of $C_j(LO)$. We adapt this strategy to the case of arbitrary deadlines.

First we consider each task $\tau_i$ in LO-criticality mode. The length $r_i^L(q)$ of the busy period in LO-criticality mode up to completion of job $q$ of $\tau_i$ is given by:

$$r_i^L(q) \;=\; (q+1)C_i(LO) \;+ \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{r_i^L(q)}{T_j} \right\rceil C_j(LO) \tag{15}$$

The worst-case response time of each job is therefore:

$$\forall q_{(0 \leq q \leq p)} : \quad R_i(LO)(q) = r_i^L(q) - qT_i \tag{16}$$

and the worst-case response time of the task in LO-criticality mode is given by:

$$R_i(LO) \;=\; \max_{\forall q_{(0 \leq q \leq p)}} \{R_i(LO)(q)\} \tag{17}$$

As with the classical analysis for FPPS, iteration over the values of $q$ ends at $p$, the smallest value such that $r_i^L(p) \leq (p+1)T_i$, indicating that $r_i^L(p)$ corresponds to the end of the priority level-i busy period in LO-criticality mode.

Analysis for each HI-criticality task $\tau_i$ considering the mode switch and HI-criticality behavior proceeds in a similar fashion. Compared to LO-criticality mode, due to the larger amounts of interference, it is possible that the busy period extends over more releases of jobs of task $\tau_i$. Let $v$ be the index of the last job of $\tau_i$ in this longer priority level-$i$ busy period. Note that $v \geq p$, where $p$ is the last release considered in LO-criticality mode.

When analyzing HI-criticality behavior, for each job $q$, we consider the longest time for which the system can remain in LO-criticality mode, and so LO-criticality tasks can be released. This is given by $r_i^L(q)$ provided that $q \leq p$. In the case that $q > p$ then it is not possible for LO-criticality mode to extend as far as the release of job $q$, since releases of LO-criticality jobs cannot take place beyond $r_i^L(p)$. The relevant equations become:

$$r_i^H(q) \;=\; (q+1)C_i(HI) \;+ \sum_{j \in \mathbf{hpH}(i)} \left\lceil \frac{r_i^H(q)}{T_j} \right\rceil C_j(HI) \;+ \sum_{k \in \mathbf{hpL}(i)} \left\lceil \frac{r_i^L(\min(q,p))}{T_k} \right\rceil C_k(LO) \tag{18}$$

$$\forall q(0 \leq q \leq v) : \quad R_i(HI)(q) = r_i^H(q) - qT_i \tag{19}$$

$$R_i(HI) \;=\; \max_{\forall q_{(0 \leq q \leq v)}} \{R_i(HI)(q)\} \tag{20}$$

This formulation shares similar pessimism to the constrained-deadline analysis on which it is based. As with AMC-rtb, it assumes that jobs of higher priority HI-criticality tasks cause interference of $C(HI)$ over the entire busy period, while higher priority LO-criticality tasks can interfere over the duration of a maximum length LO-criticality busy period. Often these two cases cannot occur together leading to pessimism. The following analysis, building on AMC-max, seeks to remove this pessimism.

## B. AMC-max-Arb analysis for arbitrary-deadline MC tasks

We now extend the ARM-max analysis to arbitrary-deadline tasks. The analysis for all tasks in LO-criticality mode is the same as AMC-rtb-Arb, i.e. (15), (16), and (17). We therefore only consider the schedulability of HI-criticality tasks across the mode switch and in the subsequent HI-criticality mode. It turns out that the derivation, described in Section V-B for the constrained deadline case, applies with simple adaptations.

We compute the completion time of the $q$th job of task $\tau_i$ when the mode switch occurs at time $s$, as follows:

$$r_i^s(q) = XC_i(HI) + YC_i(LO) + I_L(s) + I_H(s) \tag{21}$$

where $X+Y = q+1$. (The values of $X$ and $Y$ are determined below). Note (9) still provides an upper bound $I_L(s)$ on the interference from LO-criticality tasks.

Following the same argument as the AMC-max analysis, we need to determine an upper bound $I_H(s)$ on the interference that HI-criticality tasks such as $\tau_k$ can cause in a busy period of length $t$ if the mode change occurs at time $s$, with $s < t$. To do so, we maximize the number of jobs of $\tau_k$ still potentially active at time $s$ as all of these jobs can contribute interference of $C_k(HI)$, while all other jobs of $\tau_k$ contribute $C_k(LO)$. In an interval of length $t - s$ there can be at most

$$\left\lceil \frac{t - s + (D_k - T_k)}{T_k} \right\rceil + 1 \tag{22}$$

active jobs of $\tau_k$.

Equation (22) is identical to (10) that caters for $D \leq T$; we observe that it is also applicable when $D > T$. Since (11) and (12) remain unchanged, (21) can be used to compute $r_i^s(q)$, with $I_H(s)$ given by (12) and $I_L(s)$ given by (9).

The number $X$ of jobs of the task under analysis $\tau_i$ which contribute $C_i(HI)$ can be derived in a similar way to (22). Accounting for the fact that there are at most $q+1$ active jobs in total, we have:

$$X = \min\left( \left\lceil \frac{t - s + (D_i - T_i)}{T_i} \right\rceil + 1, q+1 \right)$$

The remaining steps are:

$$r_i^*(q) \ = \ \max_{\forall s} (r_i^s(q)) \tag{23}$$

where $s$ takes values corresponding to the release times of jobs of higher priority LO-criticality tasks in the interval $[0, r_i^L(q))$ with $r_i^L(q)$ given by (15).

$$\forall q_{(0 \leq q \leq p)} : \quad R_i(HI)(q) = r_i^*(q) - qT_i \tag{24}$$

where $p$ is the smallest value such that $r_i^*(p) \leq (p+1)T_i$. Finally, the worst-case response time is given by:

$$R_i(HI) \ = \max_{\forall q (0 \leq q \leq p)} \{R_i(HI)(q)\} \tag{25}$$

## VII. Priority Assignment

To maximize schedulability it is necessary to assign task priorities in an optimal way [10]. For arbitrary-deadline task sets scheduled under FPPS, and for constrained-deadline mixed-criticality task sets scheduled under SMC or AMC, an optimal priority ordering can be obtained via Audlsey's Optimal Priority Assignment (OPA) algorithm [2].

Davis and Burns [9] proved that it is both sufficient and necessary to show that a schedulability test meets three simple conditions in order for Audlsey's OPA algorithm to be applicable. These three conditions require that schedulability of a task according to the test is (i) independent of the relative priority order of higher priority tasks, (ii) independent of the relative priority order of lower priority tasks, (iii) cannot get worse if the task is moved up one place in the priority order (i.e. its priority is swapped with that of the task immediately above it in the priority order). We observe that these three conditions hold for all of the analyses derived in this paper, and thus Audsley's OPA algorithm is applicable.

## VIII. Evaluation

In this section, we present an empirical evaluation of the schedulability tests introduced for mixed-criticality tasks with arbitrary deadlines.

### 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 [7], giving an unbiased uniform distribution of values.
- Task periods $T_i$ were generated according to a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible period.
- Task deadlines $D_i$ were generated according to a log-uniform distribution in the range $[0.25, 4.0]T_i$.
- The LO-criticality execution time of each task was given by: $C_i(LO) = U_i \cdot T_i$.
- The HI-criticality execution time of each task was given by: $C_i(HI) = CF \cdot C_i(LO)$ where $CF = 2.0$.
- The probability that a generated task was of HI-criticality was given by the parameter $CP = 0.5$.

### B. Schedulability tests investigated

We investigated the performance of the following schedulability tests. In all cases, we made use of Audsley's Optimal Priority Assignment (OPA) algorithm [2].

- UB-H&L-Arb: This is a *necessary* test which checks if all of the tasks are schedulable in LO-criticality mode and if the HI-criticality tasks are schedulable in HI-criticality mode (with no LO-criticality tasks executing). It ignores the mode switch. UB-H&L-Arb thus provides an upper bound on the performance of any fixed-priority fully-preemptive scheme for scheduling mixed-criticality, arbitrary-deadline tasks.
- AMC-max-Arb: described in Section VI-B.
- AMC-rtb-Arb: described in Section VI-A.
- SMC-Arb: Analysis for Static Mixed Criticality [4] using a straightforward adaptation of the standard arbitrary-deadline analysis for FPPS [13].
- FPPS-Arb: The standard arbitrary-deadline analysis for FPPS [13] described in Section III. This test requires that both LO- and HI-criticality tasks must be schedulable in both modes.

In addition to the above tests for arbitrary-deadline tasks, we also explored the performance that could be obtained by utilizing existing schedulability tests (e.g. AMC-max [5], AMC-rtb [5], SMC [4], FPPS [11], [3]) designed for constrained-deadline task sets. These methods can be used to provide sufficient tests for arbitrary-deadline task sets via the simple expedient of *constraining* any deadline that is greater than the task's period to be equal to that period. In the figures, these methods are denoted by "(Suff.)" indicating a sufficient test. They are shown using dotted lines, with the same markers and line colors as the equivalent arbitrary-deadline tests. Note that UB-H&L-Arb (Suff.) is an upper bound on the schedulability of all fixed-priority fully-preemptive methods for constrained-deadline task sets.

### *C. Experiments*

In our experiments, the task set utilization was varied from 0.025 to 0.975[1]. For each utilization value, 1000 task sets were generated (100 for weighted schedulability experiments) and the schedulability of those task sets determined using the schedulability tests listed above. The graphs are best viewed on-line in color.

Figure 1 shows the percentage of task sets generated that were deemed schedulable for a system of 20 tasks with the defaults parameters as described in section VIII-A. We observe that AMC-max-Arb outperforms AMC-rtb-Arb by a small but significant margin. The performance of both these tests is relatively close to the theoretical upper bound given by UB-H&L-Arb; closer than the equivalent tests for constrained deadlines (dotted lines) are to their bound. This is due to the fact that longer (arbitrary) deadlines can compensate for the effects of the overload that occurs on a criticality mode switch over a longer time interval.

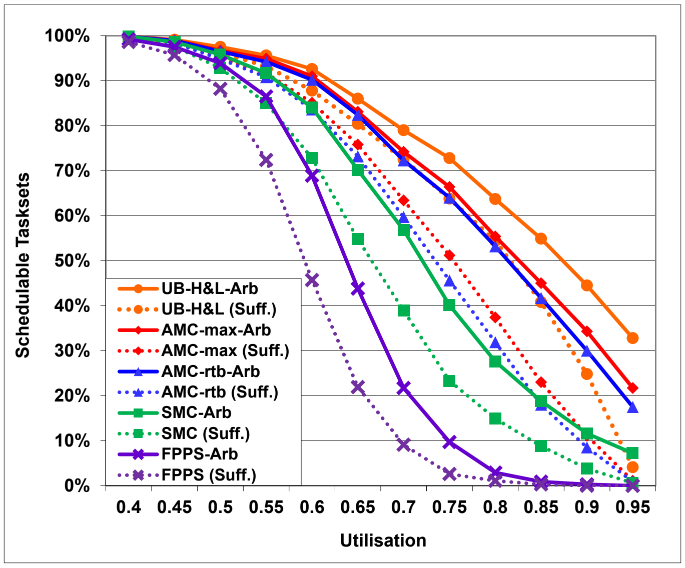

Fig. 1. Percentage of schedulable task sets

Figure 1 also illustrates that the performance of AMC-max-Arb and AMC-rtb-Arb is significantly better than that of SMC-Arb, which in turn is significantly better than FPPS-Arb. Further, each of these methods provides performance that is significantly better than the equivalent constrained-deadline test adapted to provide sufficient analysis for arbitrary-deadline tasks. Stated otherwise, for mixed criticality systems scheduled using AMC, SMC, and FPPS, increasing task deadlines beyond their periods can provide a substantial increase in guaranteed real-time performance when the schedulability tests derived in this paper are employed.

In the following figures we show the weighted schedulability measure [6] for each schedulability test as a function of some other parameter $w$ which is varied. For each value of $w$, this measure combines results for the task sets generated for all of a set of equally spaced utilization levels (0.025 to 0.975 in steps of 0.025) weighted by utilisation level. (See [6] for further details of the weighted schedulability measure).

Figure 2 shows how the weighted schedulability measure for each schedulability test changes as the range of task deadlines is varied from $[0.25, 0.25]T_i$ to $[0.25, 5.66]T_i$ (each step on the x-axis increases the upper limit of the range by a factor of $\sqrt[4]{2}$, and hence every 4 steps it increases by a factor of 2). Note in each case the deadlines are chosen at random according to a log-uniform distribution. As expected, in all cases schedulability improves as the range of possible deadlines is expanded.

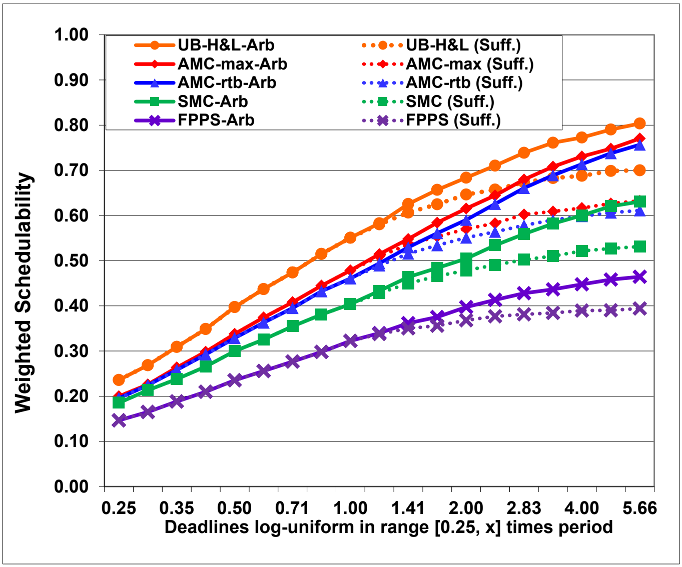

Fig. 2. Varying the range of task deadlines

We observe that while the range is no greater than $[0.25, 1.0]T_i$, then each of the arbitrary-deadline methods provides exactly the same results as its constrained-deadline counterpart (i.e. the solid and the dotted lines precisely overlap). Beyond that point, the arbitrary-deadline analysis confers increasingly superior performance. We note that the relative performance of the various schemes (AMC, SMC, and FPPS) remains broadly similar to that shown in the baseline experiment.

Figure 3 shows how the weighted schedulability measure changes as the range of task periods (ratio of max/min possible task period) is expanded from $10^{0.5} \approx 3$ to $10^4 = 10,000$. (In this experiment, task deadlines $D_i$ were chosen according to the default settings from the range $[0.25, 4.0]T_i$). When the range of tasks periods is small, then the performance advantage of AMC-max-Arb over AMC-rtb-Arb is negated. This happens because when the task periods are similar, AMC-max-Arb counts, via (22), almost all of the jobs of each higher priority HI-criticality task as contributing interference of $C(HI)$. In this case, the analysis effectively reduces to that of AMC-rtb-Arb; all jobs of higher priority HI-criticality tasks contribute $C(HI)$, and the maximum response time occurs when the transition to HI-criticality mode is as late as possible allowing the maximum interference from LO-criticality tasks. As the

[1]Utilization here is computed from the $C(LO)$ values only.

range of task periods increases, so AMC-max-Arb begins to confer an advantage.

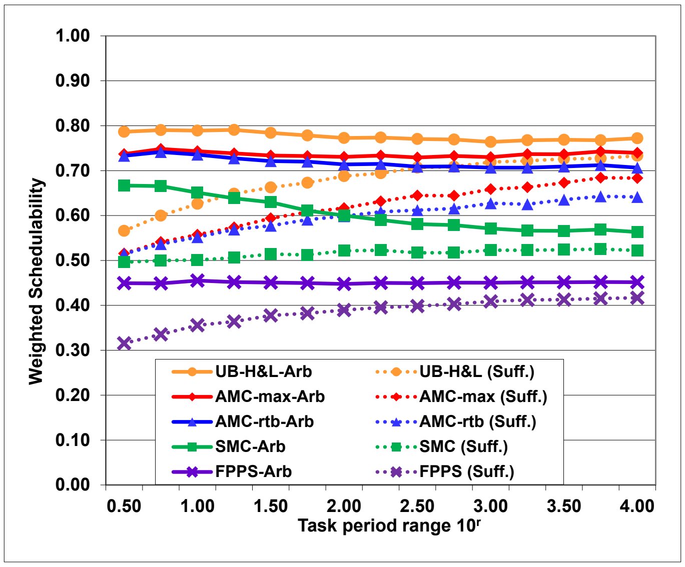

Fig. 3. Varying the range of task periods

The performance of SMC-Arb is substantially better, weighted schedulability $\approx 0.67$, when the range of task periods is small, declining to $\approx 0.56$ as the range of task periods increases. This effect is due to the influence of optimal priority assignment. If Deadline Monotonic Priority Order is assumed for SMC-Arb, then the result is a nearly horizontal line, with weighted schedulability $\approx 0.47$ (this is not shown to avoid cluttering the graph). With arbitrary deadlines and a small range of task periods, the performance of SMC-Arb can be enhanced by the OPA algorithm placing LO-criticality tasks at low priority levels. These tasks are schedulable in LO-criticality mode, assuming interference of $C(LO)$ from higher priority HI-criticality tasks. (Note such an arrangement would not be feasible under FPPS-Arb where LO-criticality tasks also have to be schedulable in HI-criticality mode, where they are subject to interference of $C(HI)$ due to higher priority HI-criticality tasks). As a consequence, HI-criticality tasks can be placed at higher priority levels and thus have fewer higher priority LO-criticality tasks that contribute interference when HI-criticality mode is considered, improving overall schedulability. The amount of flexibility available via priority assignment reduces as the range of task periods becomes larger and the deadlines of the tasks become more widely separated. This reduces the performance of SMC-Arb.

Finally, observe that in the case of the constrained-deadline methods, when the range of task periods is small, then once constrained to the range $[0.25, 1.0]T_i$ all of the task deadlines are fairly similar and hence schedulability is low compared to the situation with a much larger range of periods (and deadlines). This is a well-known property of FPPS. It happens when the total interference from higher priority tasks in a given interval is considerably higher than that implied by their utilization [12]. With longer, arbitrary deadlines or with a larger range of task periods, this excess interference reduces and so schedulability improves.

We also explored the effects of varying the number of tasks, the criticality factor (CF), and the criticality percentage (CP). These results were broadly in line with those reported for the baseline experiment. The graphs are not shown here due to space limitations.

## IX. Conclusions

In this paper, we considered the problem of scheduling mixed-criticality systems on a single processor. We studied the Adaptive Mixed Criticality (AMC) scheme that requires additional run-time support, but is able to provide superior schedulability guarantees to Static Mixed Criticality (SMC) or Fixed Priority Preemptive Scheduling (FPPS). Previous studies of AMC have restricted the task model to constrained-deadline tasks. In this paper we lifted this restriction, allowing tasks to have arbitrary deadlines.

There are two main published schedulability tests for the AMC scheme: AMC-rtb and AMC-max. In this paper we extended both tests to allow tasks to have deadlines greater than their periods. Our evaluation demonstrates that the simpler AMC-rtb-Arb form of analysis remains effective for tasks with arbitrary deadlines. The AMC-max-Arb analysis delivers improved schedulability, but at a cost in terms of increased complexity of the method.

*Acknowledgements*

The research in this paper is partially funded by the ESPRC grant, MCCps (EP/K011626/1). EPSRC Research Data Management: No new primary data was created during this study.

## References

[1] S. Asyaban and M. Kargahi. An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems. *Real-Time Systems*, Aug 2017.

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

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

[4] S. 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.

[5] S. 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] 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.

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

[8] A. Burns and R. I. Davis. A survey of research into mixed criticality systems. *ACM Computing Surveys*, To appear, 2017.

[9] R. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. In *Real-Time Systems, Volume 47, Issue 1*, pages 1–40, 2010.

[10] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns. A review of priority assignment in real-time systems. *Journal of systems architecture*, 65:64–82, 2016.

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

[12] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM*, 20(1):46–61, 1973.

[13] K. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analysing fixed priority hard real-time tasks. *Journal of Real-Time Systems*, 6(2):133–151, 1994.

[14] 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.