An Extended Fixed Priority Scheme for Mixed Criticality Systems

S. Baruah
Department of Computer Science,
University of North Carolina, US.
Email: baruah@cs.unc.edu

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*—This paper considers a novel implementation scheme for fixed priority (FP) uniprocessor scheduling of mixed criticality systems. The scheme requires that jobs have their execution times monitored. If system behavior inconsistent with lower criticality levels is detected during run-time via such monitoring, (i) tasks of lower criticalities are discarded (this is already done by current FP mixed-criticality scheduling algorithms); *and* (ii) the priorities of the remaining tasks may be re-ordered. Evaluations illustrate the benefits of this scheme.**

I. Introduction

Although the formal study of mixed criticality systems (MCSs) is a relatively new endeavor, starting with the paper by Vestal (of Honeywell Aerospace) in 2007 [11], a standard execution model for mixed-criticality recurrent real-time systems is emerging (see for example [8]). For dual criticality systems this standard model may be described as follows:

- Each task τ_i is characterized by its criticality $\chi_i \in \{\text{LO}, \text{HI}\}$, the minimum inter-arrival time of its jobs (period denoted by T_i), deadline (relative to the release of each job, denoted by D_i), and worst-case execution times (one per criticality level), denoted by $C_i(\text{HI})$ and $C_i(\text{LO})$. It is assumed that $C_i(\text{HI}) \geq C_i(\text{LO})$.
- A mixed criticality system is defined to execute in either of two "modes": a HI-criticality mode and a LO-criticality mode. The system starts in the LO–criticality mode, and remains in that mode as long as all jobs execute within their low criticality execution times ($C_i(\text{LO})$). If any job executes for its $C_i(\text{LO})$ execution time without completing then the system immediately moves to the HI-criticality mode.
- The correctness criterion for the system is as follows: all job deadlines should be met in LO-criticality mode, but in HI-criticality mode it is only required that jobs of HI-criticality tasks meet their deadlines[1].

In this paper, we are concerned with the *fixed-priority (FP)* scheduling of such dual criticality task systems. There is a general consensus concerning the implementation model of FP scheduling for "regular" (i.e. not mixed-criticality) systems: which is that at any instant during run-time, the highest priority active job is selected for execution. In mixed-criticality systems, however, matters are not quite as straightforward:

- The original work of Vestal [11] assumes exactly the same standard execution model that is used for FP scheduling of regular task systems.
- The model subsequently considered in [5] (and many other works dealing with non-FP mixed-criticality scheduling), however, incorporates a more general execution model, that allows for *the aborting and subsequent discarding of* LO*-criticality tasks once the system moves to* HI*-criticality mode*. It was shown [5] that this generalization results in a significant improvement in schedulability, in the sense that many task systems that cannot be scheduled correctly under the standard execution model can be scheduled correctly if LO-criticality tasks may be discarded upon a mode-change.
- By allowing that certain tasks be discarded upon the occurrence of a mode-change, the model of [5] and others requires a considerable re-organization of the run-time structure of the system when a mode-change occurs. In this paper, we therefore explore whether further improvements in schedulability can be achieved by allowing for one more change upon the system transiting to HI-criticality mode. Specifically, we consider the effect of allowing *a re-ordering of the priorities assigned to tasks upon a transition to* HI*-criticality mode*.

We will see (Section V) that in our proposed approach both sets of priorities, those used initially and those to be used in the event of a mode-change to HI-criticality mode, are precomputed prior to run-time. Since implementing the algorithms in [5] already requires that the run-time queue of ready jobs be reorganized upon a mode-change, the additional changes to the run-time execution model that we require over and above the changes required in order to implement the algorithms in [5] are slight and do not, in our opinion, impose significant additional challenges or overheads.

Organization. The remainder of this paper is organized as follows. In Section II we briefly review some prior work from [5] that is most closely related to the algorithms we will derive here. In Section III we explore an approach different from the one considered in [11], [5], for determining priorities for the FP scheduling of mixed-criticality sporadic task systems.

[1]This is a bit of an over-simplification: strictly speaking, the model does not require deadlines of LO-criticality jobs to be met even in LO-criticality mode if the system is definitely going to move to HI-criticality mode in the future.

This approach is based on a recently-proposed EDF-based algorithm called EDF-VD [3] for scheduling mixed-criticality sporadic systems. In Section IV we show there is indeed a benefit to re-ordering priorities upon transiting to HI-criticality mode. This motivates us to improve the algorithm derived in Section III by incorporating priority-changing; the improved algorithm is presented in Section V. We provide an empirical evaluation of this improved algorithm in Section VI.

II. Related work

We are concerned here with the schedulability analysis of mixed criticality (MC) constrained-deadline sporadic task systems that are being scheduled by the standard fixed priority (FP) preemptive dispatcher on a single processor. This topic has previously been considered in [11], [5]; a priority-assignment algorithm called *Adaptive Mixed Criticality (AMC)* was introduced in [5], and shown to strictly dominate the algorithm in [11] (which was referred to as Static Mixed Criticality, or SMC, in [5]). We now provide a brief summary of AMC.

Under AMC[2], priority assignment is done according to Audsley's Optimal Priority Assignment algorithm [1]. That is, some task is identified that is schedulable if assigned the lowest priority; this task is assigned lowest priority and removed from consideration; and the process is recursively applied to the remaining tasks.

To completely specify the priority assignment strategy, it remains to describe how the lowest-priority task is identified.

It was shown in [5], using results from [9] that within each criticality level (i.e., when comparing different LO-criticality tasks or different HI-criticality tasks), tasks can be assigned priorities in deadline-monotonic (DM) partial order. Hence in seeking to determine the lowest-priority task it suffices to consider only the LO-criticality task with largest relative deadline, and the HI-criticality task with largest relative deadline. This is done by determining (an upper bound on) the worst-case response time of a task τ_k if it were assigned the lowest priority, and checking whether this is no larger than the relative deadline parameter of the task. If τ_k is a LO-criticality task (i.e., $\chi_k = \text{LO}$), then we seek to bound its worst-case response time if all jobs execute for no more than their own LO-criticality WCETs; if τ_k is a HI-criticality task ($\chi_k = \text{HI}$), then we must bound its worst-case response time when some jobs may execute for up to their HI-criticality WCETs.

In somewhat greater detail, let L_{LO} (L_{HI}, respectively) denote an upper bound on the length of the longest busy interval during any LO-criticality (HI-criticality, resp.) behavior of τ. It is evident that any LO-criticality task τ_i satisfying $D_i \geq L_{\text{LO}}$ may be assigned the lowest priority: since no LO-criticality behavior can span the entire interval between the release of any job of τ_i and its deadline, no such job will miss its deadline if τ_i is assigned the lowest priority. Similarly, any HI-criticality task τ_i satisfying $D_i \geq L_{\text{HI}}$ may be assigned the lowest priority.

[2]Two different schemes, AMC-rtb and AMC-max were described in [5]. We restrict our attention here to AMC-rtb.

So we first compute L_{LO}. Based on results from Response-Time Analysis (RTA) [2] it is straightforward to observe that L_{LO} can be set equal to the smallest positive value of t satisfying the response-time formula:

$$t = \sum_{\forall j} \lceil \frac{t}{T_j} \rceil C_j(\text{LO}). \tag{1}$$

We seek to determine L_{HI} next. Without loss of generality, let us suppose that the longest busy interval in any HI-criticality behavior occurs for a sequence of jobs of τ in which the first job arrives at time zero. Let t_1 denote the time-instant at which the mode-change occurs. That is, t_1 is the first instant at which a job of some task τ_i does not signal completion despite having received $C_i(\text{LO})$ units of execution. No job of any LO-criticality task will receive any execution after time-instant t_1. Hence for any τ_j with $\chi_j = \text{LO}$, at most $\lceil \frac{t_1}{T_j} \rceil$ jobs of τ_j may execute in this longest busy interval.

Since L_{LO} is, by definition, an upper bound on the length of the largest busy interval in any LO-criticality behavior, it follows that $t_1 \leq L_{\text{LO}}$. Hence the total amount of execution by jobs of LO-criticality tasks in this longest busy interval of any HI-criticality behaviour is bounded from above by $\sum_{j:\chi_j=\text{LO}} \big(\lceil L_{\text{LO}}/T_j \rceil \cdot C_j(\text{LO}) \big)$. And for any value of t, the total amount of execution of HI-criticality jobs over the interval $[0, t)$ in any HI-criticality behaviour is bounded from above by $\sum_{j:\chi_j=\text{HI}} \big(\lceil t/T_j \rceil \cdot C_j(\text{HI}) \big)$. It therefore follows that L_{HI}, an upper bound on the length of the longest HI-criticality busy interval, can be set equal to the smallest value of t that is $\geq L(\text{LO})$, satisfying:

$$t = \sum_{j:\chi_j=\text{LO}} \lceil \frac{L_{\text{LO}}}{T_j} \rceil C_j(\text{LO}) + \sum_{j:\chi_j=\text{HI}} \lceil \frac{t}{T_j} \rceil C_j(\text{HI}). \tag{2}$$

Plugging the value for L_{LO} obtained by solving (1) into the recurrence relation given by (2), we can determine the value of L_{HI} by using standard techniques for determining fixed-points.

III. A new priority-assignment scheme

The AMC algorithm, described in Section II above, performs priority-assignment by concurrently considering both LO-criticality and HI-criticality modes: Equation (1) computes the response time in LO-criticality mode while (2) computes the response time in HI-criticality mode, and a priority assignment is made only if both values are acceptable (i.e., no larger than the relative deadline). We now propose a somewhat different two-step approach to priority assignment:

1) In the first step, we assign priorities by only considering LO-criticality mode and check all deadlines are met in LO-criticality mode.
2) In the second step, we then *check* whether the priority assignment so obtained will also meet all deadlines if the system behavior were to transit to HI-criticality mode.

In Section V we build upon this approach: given an execution model where priorities may be re-assigned in the event of a mode-change to HI-criticality mode, we describe how this second set of priorities may be computed.

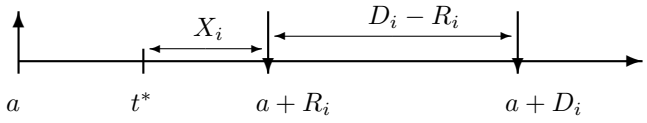

Fig. 1. A job of HI-criticality task τ_i arrives at time a, and has a LO-criticality response time bound $a + R_i$. Its deadline is at $a + T_i$. Criticality level change is triggered at t^*. We define X_i as follows: $X_i = a + R_i - t^*$.

In the ***first step***, we assign priorities to the tasks as we would in a regular (not mixed-criticality) task system, using the LO-criticality WCET of each task as its WCET for this purpose. Since deadline-monotonic (DM) priorities are known to be optimal, we may assign these priorities in DM order[3].

Some notation: for each i, $1 \leq i \leq n$, let R_i denote the worst-case response time of task τ_i in LO-criticality mode under this priority assignment. These R_i values are easily computed using the standard response-time analysis (RTA) recurrence relation.

In the ***second step***, we need to determine whether the priority-assignment determined above continues to meet all deadlines if some job executes beyond its LO-criticality WCET, thereby triggering a change to HI-criticality mode. Let t^* denote the time-instant at which this happens (see Figure 1). Let τ_i denote any HI-criticality task. Suppose that the *current job* of τ_i at time-instant t^* — the job (if any) of τ_i that arrives during the interval $(t^* - T_i, t^*]$ — arrives at time $a \leq t^*$. We note that if $t^* > a + R_i$, then this current job of τ_i must have signaled that it has completed execution prior to time-instant t^*, since τ_i's job is guaranteed to complete by time-instant $a + R_i$ in LO-criticality mode.

Consider now the situation when $t^* \leq a + R_i$ (see Figure 1). Since the current job of τ_i would have completed by time-instant $a + R_i$ in a LO-criticality behavior, it must have executed for at least an amount $\max(0, C_i(\text{LO}) - X_i)$ prior to time-instant t^*. Therefore in the HI-criticality behavior, the amount of execution remaining to be done over the interval $[t^*, a + D_i]$ is

$$\begin{aligned} &\leq C_i(\text{HI}) - \max(0, C_i(\text{LO}) - X_i) \\ &= \min(C_i(\text{HI}), C_i(\text{HI}) - C_i(\text{LO}) + X_i) \end{aligned} \tag{3}$$

Hence, the *worst-case* behavior of this task after time-instant t^* is represented by this job with an execution requirement of at most $\min(C_i(\text{HI}), C_i(\text{HI}) - C_i(\text{LO}) + X_i)$ and relative deadline $D_i - R_i + X_i$, followed by arrivals of jobs with an execution requirement of $C_i(\text{HI})$ at all instants $a + kT_i$, $k \geq 1$. What value of X_i would maximize this worst-case behavior? We observe that

1) setting $X_i \leftarrow C_i(\text{LO})$ results in the first job having WCET $C_i(\text{HI})$ and relative deadline $D_i - R_i + C_i(\text{LO})$;
2) Making X_i larger than $C_i(\text{LO})$ increases the relative deadline without increasing the WCET; and
3) Making X_i smaller than $C_i(\text{LO})$ by an amount Δ reduces both the WCET and the relative deadline by this same amount Δ. This cannot lead to a "worse" scenario than setting $X_i \leftarrow C_i(\text{LO})$), since
   - τ_i's interference on lower-priority tasks *de*creases when this happens, whereas
   - the interference τ_i suffers from higher-priority tasks does not change (hence if the job of τ_i had received its WCET by its deadline, it must have received WCET-minus-Δ by deadline-minus-Δ).

Therefore, the worst-case situation is when $X_i \leftarrow C_i(\text{LO})$. As we saw above, in this situation the first job has WCET $C_i(\text{HI})$ and regular deadline $D_i - R_i + C_i(\text{LO})$; equivalently, we can think of this job as having a release jitter equal to $\big(R_i - C_i(\text{LO})\big)$ and WCET $C_i(\text{HI})$, relative deadline D_i and period T_i.

The execution requirement of the HI-criticality task τ_i after time-instant t^* may therefore be modeled by a sporadic-with-release-jitter task (denoted by the 4-tuple (J_i, C_i, D_i, T_i); with J_i the *release jitter* of the task) characterised by $\big(R_i - C_i(\text{LO}), C_i(\text{HI}), D_i, T_i\big)$. To determine if all HI-criticality tasks will meet all their deadlines after time-instant t^*, we must therefore check whether the regular (i.e., not mixed-criticality) sporadic task system

$$\bigcup_{\chi_i = \text{HI}} \Big\{ \big(R_i - C_i(\text{LO}), C_i(\text{HI}), D_i, T_i\big) \Big\},$$

is deemed FP-schedulable according to the assigned priorities.

The entire algorithm is listed in Figure 2.

Step 1: Assign priorities to the regular sporadic task system (and check schedulability)

$$\bigcup_{\forall i} \big\{ \big(C_i(\text{LO}), D_i, T_i\big) \big\}$$

Step 2: Determine whether the regular sporadic task system with release jitter

$$\bigcup_{\chi_i = \text{HI}} \Big\{ \big(R_i - C_i(\text{LO}), C_i(\text{HI}), D_i, T_i\big) \Big\}$$

is scheduled correctly according to the priorities determine in Step 1, where R_1 denotes the worst-case response time of τ_i in LO-criticality mode

Fig. 2. Priority-assignment algorithm of Section III

IV. Changing priorities

The fixed-priority scheduling model assumed in Sections II-III above requires that each task is assigned a single priority value, which determines its priority relative to other tasks during run-time. We now ask: can we obtain improved schedulability (in the sense of being able to make more systems

[3]In our experiments, reported in Section VI, we have chosen to instead use Audsley's Optimal Priority Assignment algorithm [1], and favor LO-criticality tasks with longer deadlines for assigning lower priorities whenever that approach reveals that there is a choice of multiple tasks that may be assigned the lowest priority. This decision was made based on experimental observations indicating that such an approach results in more systems being deemed schedulable when compared to the straightforward DM approach.

schedulable) if we are allowed to *re*-assign priorities upon the system's transition to the HI-criticality mode? We argue that this is a minor (and reasonable) generalization to the model considered in [5], and continues an evolving trend of minor generalizations. It also has some parallels with the EDF schemes that allow effective deadlines to change after the mode change [4], [10]:

- The original model of Vestal [11] is restricted to off-line analysis only; it does not allow changes to the run-time algorithm that is used in implementing FP scheduling.
- The model considered in [5] (and many other works dealing with non-FP mixed-criticality scheduling) allow a limited change to the run-time algorithm: they all allow for the aborting and subsequent discarding of LO-criticality tasks if the transition to HI-criticality mode occurs.
- The model we are now proposing allows both the aborting of LO-criticality tasks, *and* a re-ordering of the priorities of the remaining (HI-criticality) tasks, if the transition to HI-criticality mode occurs.

The following theorem asserts that this additional generalization enhances system schedulability:

Theorem 1: There are mixed-criticality sporadic task systems that can be scheduled correctly by a FP algorithm that can re-assign priorities upon a mode change, that cannot be scheduled by a FP algorithm that may never re-assign priorities.

Proof: Consider the three-task system that is given below:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	D_i	T_i
τ_1	HI	1	2	10	10
τ_2	HI	1	2	12	12
τ_3	LO	4	-	5	5

Suppose we had to use the *same* priorities in both modes. It is evident that τ_3 must be assigned one of the two highest priorities, in order to meet its deadline in LO-criticality mode. So, consider both possibilities; for each, consider the following behavior in a synchronous arrival sequence.

1) $\tau_1 > \tau_3 > \tau_2$: τ_1 executes over $[0, 1]$, τ_3 over $[1, 5]$ and then over $[5, 9]$. τ_2 executes over $[9, 10]$, but fails to signal completion – the system therefore transits to the HI-criticality mode. A job of τ_1 arrives and executes for its HI-criticality WCET of 2, thereby causing τ_2's first job to miss its deadline.
2) $\tau_2 > \tau_3 > \tau_1$: τ_2 executes over $[0, 1]$, τ_3 over $[1, 5]$ and then over $[5, 9]$. τ_1 executes over $[9, 10]$, but fails to signal completion – the system therefore transits to the HI-criticality mode. Since 10 is the deadline of τ_1's first job, this job has missed its deadline.

Now if we were to initially assign priorities $\tau_1 > \tau_3 > \tau_2$, but immediately switch to the ordering $\tau_2 > \tau_1$ upon determining the criticality change, we see that both τ_1 and τ_2 continue to meet their deadlines:

- $\tau_1 > \tau_3 > \tau_2$: τ_1 executes over $[0, 1]$, τ_3 over $[1, 5]$ and then over $[5, 9]$. τ_2 executes over $[9, 10]$, but fails to signal completion – the system therefore transits to the HI-criticality mode. Now τ_2 has the highest priority so it continues to run in $[10, 11]$ and meets its deadline. After that τ_1 needs two units of execution in every 10 and τ_2 needs two units of execution in every 12, which is easily accommodated.

■

We invite the reader to re-visit this example after reading Section V below, as the above example also illustrates the priority allocation scheme defined in the algorithm presented in that section. If we compute the response time of the HI-criticality tasks in the LO-criticality mode then we get (for priority ordering $\tau_1 > \tau_3 > \tau_2$): $R_1(\text{LO}) = 1$ and $R_2(\text{LO}) = 10$. The deadline-minus-jitter values for the two tasks are therefore (10-1+1) = 10 for τ_1 and (12-10+1) = 3 for τ_2 . So $D - J$ priority ordering results in $\tau_2 > \tau_1$ after the mode change – exactly as the proof above required.

V. Improved Algorithm: when Priorities May Change – PMC

If priorities are allowed to change upon the system transiting from LO-criticality mode to HI-criticality mode, we may modify the priority-assignment algorithm of Section III to compute *two* sets of priorities.

1) The first set of priorities are computed as in the first step of the algorithm described in Section III. These are the priorities that are initially used by the run-time dispatcher, and continue to be used while the system is in LO-criticality mode.
2) The second set of priorities are designed to be used if the system transits to HI-criticality mode. As shown during the discussion of the second step of the algorithm in Section III, the HI-criticality tasks that need to be scheduled upon the system transiting to HI-criticality mode may be modeled as a collection of regular tasks with release jitter. Since we may assign these HI-criticality tasks priorities different from those they were assigned in the first step, we will assign priorities according to the *Deadline minus Jitter (D-J)*-monotonic priority assignment scheme [12]; as proved in [12], this priority assignment is optimal for the fixed-priority scheduled constrained-deadline sporadic task systems with release jitter. The regular (i.e., not mixed-criticality) sporadic task system

$$\bigcup_{\chi_i = \text{HI}} \Big\{ \big(R_i - C_i(\text{LO}), C_i(\text{HI}), D_i, T_i\big) \Big\}$$

should be deemed FP-schedulable according to the test of [12].

VI. Evaluation

In this section, we present an empirical investigation, examining the effectiveness of our PMC scheme.

A. Taskset parameter generation

The taskset 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 distribution of utilisation values.
- Task periods were generated according to a log-uniform distribution with a factor of 100 difference between the minimum and maximum possible task period.
- Task deadlines were set equal to their periods.
- $C_i(LO) = U_i/T_i$.
- $C_i(HI) = CF \cdot C_i(LO)$ where the criticality factor CF is a fixed multiplier e.g., $CF = 2.0$.
- The probability that a generated task was a high 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.

- UB-H&L: A composite upper bound on any fixed priority mixed-criticality scheduling technique described in [5].
- AMC-max: Method 2 described in [5].
- AMC-rtb: Method 1 described in [5].
- SMC (Static Mixed Criticality): the approach described in [5].
- SMC-NO: the SMC approach without run-time monitoring (i.e., the approach published by Vestal [11]).
- CrMPO: Criticality Monotonic Priority Ordering described in [5].
- PMC: The method developed in this paper, and discussed in Section V.

Note, UB-H&L is not a schedulability test, it is an upper bound on any such test. It is included to illustrate the effectiveness, or not, of the other schemes.

C. Experiments

In our experiments, the taskset utilisation was varied from 0.025 to 0.975[4]. For each utilisation value, 1000 tasksets were generated and the schedulability of those tasksets determined using the seven algorithms / schedulability tests. The graphs are best viewed online in colour.

Fig 3 plots the percentage of tasksets generated that were deemed schedulable for a system of 20 tasks, with on average 50% of those tasks having high criticality (CP = 0.5) and each task having a high criticality execution time that is 2.0 times its low criticality execution time (CF = 2.0).

We observe that the PMC schedulability test has similar performance to the AMC-rtb test, yet the two tests are incomparable.

In the following figures we show the weighted schedulability measure $W_y(p)$ [6] for schedulability test y as a function of parameter p. For each value of p, this measure combines results for all of the tasksets τ generated for all of a set of

[4]Utilisation here is computed from the $C(LO)$ values only.

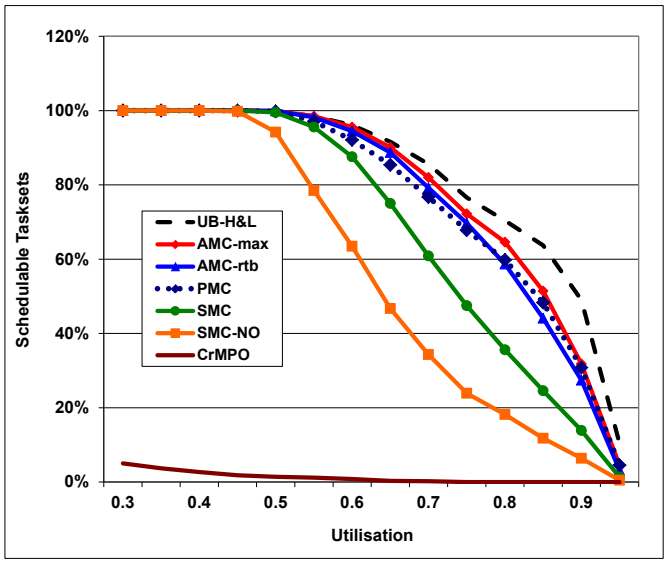

Fig. 3. Percentage of Schedulable Tasksets

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 taskset τ with parameter value p:

$$W_y(p) = (\sum_{\forall \tau} u(\tau) \cdot S_y(\tau, p)) / \sum_{\forall \tau} u(\tau) \qquad (4)$$

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

The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [6]. Weighting the individual schedulability results by taskset utilization reflects the higher value placed on being able to schedule higher utilization tasksets.

Fig 4 varies the criticality factor, Fig 5 varies the percentage of tasks with high criticality and Fig 6 varies the size of the task set. A number of points are illustrated by these figures:

- CrMPO preforms very badly as priority ordering it uses is far from optimal.
- Figs 4 and 5 show a decline in schedulability for high criticality factors or high percentage of HI-critical tasks; but this is due to utilisation being calculated via the $C(LO)$ values only. In effect utilisation goes up (and hence schedulability goes down) as the parameter of the figure increases.
- PMC and AMC-rtb have similar but incomparable performance; however, PMC has slightly better performance as the number of tasks increases giving more scope for the change in priorities to make a difference.

It is interesting to note that the new method PMC has weak performance, worse than SMC-NO, when there is a large proportion of high criticality tasks. This can easily be seen by considering its behavior on a taskset that is composed entirely of high criticality tasks. Here, PMC effectively adds

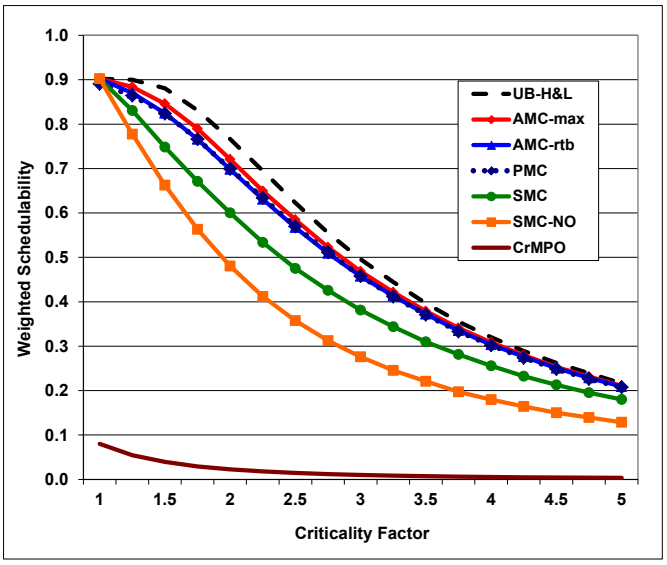

Fig. 4. Varying the Criticality Factor

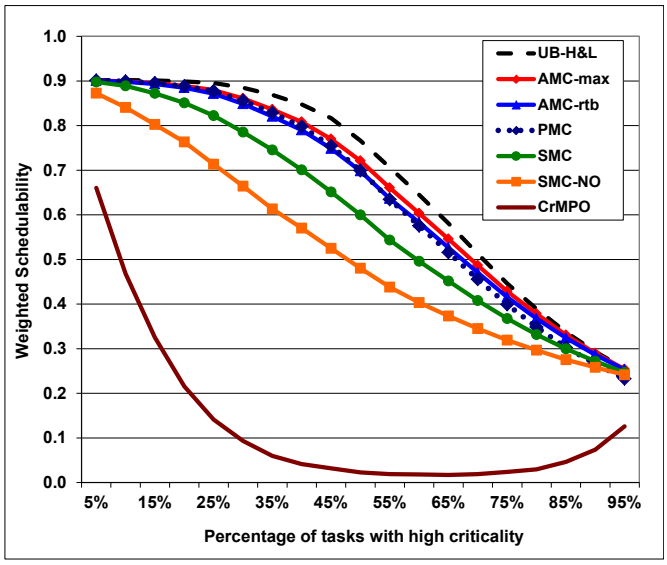

Fig. 5. Varying the Criticality Mix

in some (in this case unnecessary) jitter and then checks the schedulability of the high criticality tasks, obtaining worse results than AMC-rtb which would include no additional jitter. On the other hand, if there are a large number of tasks of high priority, but low criticality, then the AMC-rtb test (see Equation (7) in [5]) can potentially become markedly inferior to the PMC method, which excludes any contribution from these low criticality tasks other than as jitter on the high critical tasks (the jitter could in some cases have no effect).

Overall the key observation of these figures, representing an extensive set of experiments, is that the relationship between the six lines on the graphs remains relatively stable, and that both AMC and PMC are very effective yet incomparable means of scheduling mixed criticality systems.

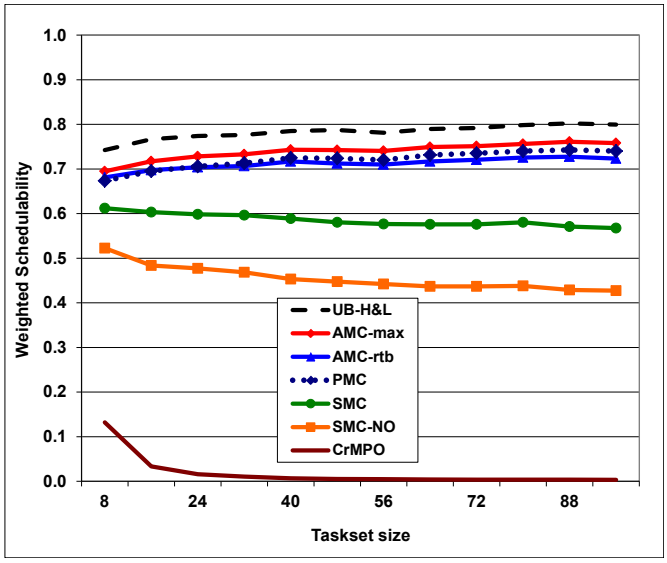

Fig. 6. Varying the Number of Tasks

VII. Conclusions

In this paper we have presented an extension to the 'standard' model that has emerged for fixed priority scheduling of mixed criticality systems. Rather than have a single priority per task that is used in all criticality modes, higher priority tasks are assigned a new priority when there is a criticality mode change. This new priority reflects the impact that the previous lower criticality level has on the new higher criticality mode by recognising that the task has, in effect, experienced release jitter. We are therefore able to exploit an existing optimal priority ordering policy for tasks with release jitter.

Evaluations show that the new scheme performs better than many previously published approaches, and has similar performance to the AMC scheme. Neither approach dominate the other, and hence having both approaches available to systems engineers increases the likelihood of a specific task set being deemed schedulable (by AMC or PMC).

This paper, like many on Mixed Criticality has adopted a dual-criticality assumption. Extension of the PMC approach to multiple criticality levels is an area for future work.

References

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

[2] 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.

[3] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *Proc. of ECRTS, Pisa*, pages 145–154, 2012.

[4] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. In *Proc. of the 19th Annual European Symposium on Algorithms (ESA 2011) LNCS 6942, Saarbruecken, Germany*, pages 555–566, 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. Davis. Mixed criticality systems: A review. Technical Report MCC-1(b), available at http://www-users.cs.york.ac.uk/burns/review.pdf, Department of Computer Science, University of York, 2013.

[9] R. Davis and A. Burns. Robust priority assignment for fixed priority real-time systems. In *Proc. of IEEE Real-Time Systems Symposium (RTSS)*, 2007.

[10] P. Ekberg and W. Yi. Bounding and shaping the demand of generalized mixed-criticality sporadic sprodic tasks. *Journal of Real-Time Journal*, 2013.

[11] 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.

[12] A. Zuhily and A. Burns. Optimal (D-J)-monotonic priority assignment. *Information Processing Letters*, 103(6):247–250, 2007.