Towards A More Practical Model for Mixed Criticality Systems

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

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

***Abstract*—Mixed Criticality Systems (MCSs) have been the focus of considerable study over the last six years. This work has lead to the definition of a standard model that allows processors to be shared efficiently between tasks of different criticality levels. Key aspects of this model are that a system is deemed to execute in one of a small number of criticality modes; initially the system is in the lowest criticality mode, but if any task executes for more than its predefined budget for this criticality level then a mode change is made to a higher criticality mode and all tasks of the lowest criticality level are abandoned (aborted). The initial criticality level is never revisited. This model has been useful in defining key properties of MCSs, but it does not form a useful basis for an actual implementation of a MCS. In this paper we consider the tradeoffs stemming from a consideration of what systems engineers require at run-time and the actual properties of the model that scheduling analysis guarantees. Alternative models are defined that allow low criticality tasks to continue to execute after a criticality mode change. The paper also addresses robust priority assignment.**

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 [24], a standard model has emerged (see for example [4], [5], [13], [14], [19]). For dual criticality systems this standard model has the following properties:

- A mixed criticality system is defined to execute in either of two modes: a HI-crit mode and a LO-crit mode.
- Each task is characterised by 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 MCS model is that $C(HI) \geq C(LO)$.
- The system starts in the LO-crit mode, and remains in that mode as long as all jobs execute within their low 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-crit mode.
- As the system moves to the HI-crit mode all LO-crit tasks are abandoned. No further LO-crit jobs are executed.
- The system remains in the HI-crit 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 met with some criticism from systems engineers[1] that it does not match their expectations. In particular:

- In the HI-crit mode LO-crit tasks should not be abandoned but be allowed to make some progress, as long as they are not interfering with HI-crit tasks.
- For systems which operate for long periods of time it should be possible for the system to return to the LO-crit mode when the conditions are appropriate.
- Whereas a HI-crit job executing for more than its $C(LO)$ execution time must induce a mode change, a LO-crit job should be constrained so that it cannot execute for more than $C(LO)$ (so no a mode change).

Some of these criticisms are partly misplaced as any high integrity system should remain in the LO-crit mode for its entire execution: the transition to HI-crit mode is only a theoretical possibility that the scheduling analysis can exploit [4]. Nevertheless, in less critical applications (such as those envisaged in the automotive industry) actual criticality mode changes may be experienced during operation and the above criticisms should be addressed.

Our contributions. In this paper we address all of the concerns listed above. First, we present alternative implementation models that (a) do not abandon LO-crit tasks upon transitioning to HI-crit mode; and (b) define conditions for the system to transition back to LO-crit mode. And second, we propose priority-assignment techniques for fixed-priority mixed-crit schemes that are more *robust* than previously-proposed techniques in the sense that systems assigned priorities according to these robust priority-assignment schemes are less likely to undergo a mode change to HI-crit mode. In other work [9] we have discussed (and removed) a further criticism of the standard model – that tasks are independent. This was done by revisiting and adapting the original priority ceiling protocol. Other proposals comes from Lakshmanan et al. [15] They define two protocols: PCIP (Priority and Criticality Inheritance Protocol) and PCCP (Priority and Criticality Ceiling Protocol). Both of these contain the notion of criticality inheritance. This

[1]For example at the tutorial presented at the 2012 Embedded System Week http://www.esweek.org/ and at the workshop that was part of the 2013 HiPEAC conference http://www.hipeac.net/conference/berlin/workshop/integration-mixed-criticality-subsystems-multi-core-processors.

notion is also used by Zhao et al. [25] in their HLC-PCP (Highest-Locker Criticality Priority Ceiling Protocol).

Related work. Background material on MCS research can be obtained from the following papers [2], [4], [5], [12]–[14], [24]). A survey on MCS research is available from the MCC (Mixed Criticality Systems on Many-core Platforms) project's web site[2] Santy et al. [19] attempt to remove some of the strictness of the standard model; however, they largely focus on a different set of issues.

II. Limitations Imposed by Scheduling Analysis

The standard model requires an immediate change to the HI-crit mode and the consequent abandonment of all active LO-crit jobs upon a change to HI-crit mode. Despite this simplifying assumption, it has been shown [2] that the mixed criticality schedulability problem is strongly NP-hard even if there are only two criticality levels. Hence only sufficient rather than exact analysis is computationally feasible. One of the consequences of this intractability is that a significant proportion of the available (sufficient rather than exact) analysis that has been produced for MCSs actually assumes that any LO-crit job that has been released at the time of the mode change will complete, rather than being aborted – this is the case with, e.g., the analysis in [2], [4], [24] (although not the analysis of the EDF-VD algorithm [3]).

For example, for constrained deadlines tasks, the Adaptive Mixed Criticality (AMC Method 1 or AMC-rtb) approach presented at RTSS in 2011 [4] first computes the worst-case response times for all tasks in the LO-crit mode (denoted by $R_i(LO)$). This is accomplished by solving, via fixed point iteration, the following response-time equation for each task:

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

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

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

$$R_i(HI) \;=\; C_i(HI) \;+\; \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \;+ \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(HI)}{T_k} \right\rceil C_k(LO) \quad (2)$$

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

This equation is conservative for AMC as it does not take into account the fact that LO-crit tasks cannot execute for the entire busy period of a high criticality task in the HI-crit mode. A change to HI-crit must occur before $R_i(LO)$ and hence (2) can be modified as follows:

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

which 'caps' the interference from LO-crit tasks as $R_i(HI)$ must be greater than $R_i(LO)$.

The cap is however at its maximum level. The maximum number of LO-crit jobs are assumed to interfere and each of these jobs is assumed to complete – each inducing the maximum interference of $C_k(LO)$.

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

III. Alternative Models

Notice that one of the consequences of the limitations imposed by the schedulability analysis that we described in Section II above is that any LO-crit job, once released, is assumed to complete. This "limitation" can be exploited in an implementation model by only allowing the status of LO-crit tasks to be changed when they are suspended between jobs.

To accommodate the requirement to allow some progress for LO-crit tasks after the move to HI-crit mode, and to allow a mode change back to LO-crit mode, LO-crit tasks must not be abandoned. Rather they must remain runnable but in a way that cannot impact on HI-crit jobs. For a fixed priority system this means: (i) changing the priority of these tasks to be below the lowest priority of any HI-crit task, or (ii) reducing the execution time requirements of these tasks so that both HI and LO criticality tasks can execute successfully in the HI-crit mode, or (iii) extending the period of the LO-crit tasks to achieve the same result.

In the rest of this paper we will concentrate on single processor systems scheduled using fixed priorities. We consider three complimentary schemes. In the first LO-crit tasks have their priorities reduced, in the second they have their execution-time requirements reduced, and in the third they have their periods extended.

A. Reducing LO-crit Tasks' Priorities

Although with fixed priority scheduling priorities are 'fixed', to accommodate the requirements identified above, it must be possible to dynamically change the priority of a LO-crit task. Such tasks will have two priorities, $P_i(LO)$ and $P_i(HI)$; with the constraints that $P_i(HI) \leq P_i(LO)$ and $P_i(HI) < \min_{j \in \mathbf{HI}}(P_j)$ where $\mathbf{HI}$ is the set of all HI-crit tasks. We note that most RTOSs and programming languages allow the base priority of a task to be altered. For optimal performance the relative ordering of the priorities of LO-crit tasks will be the same in both criticality modes.

[2]http://www.cs.york.ac.uk/research/research-groups/rts/mcc/.

At run-time the overhead cost of changing the priority of a runnable task can be relatively high as the task must be taken out of the run queue and then reinserted at the place appropriate for its new priority. Fortunately due to the limitations of the analysis, which not only assumes all LO-crit jobs complete but also assumes they do so at their current priority, it is acceptable to only modify the priority of a task (from $P_i(LO)$ to $P_i(HI)$) when the task is suspended.

To return the system from HI-crit to LO-crit requires that a further mode change is undertaken. This is a more extensive mode change as new work (the LO-crit tasks) needs to be reintegrated with the HI-crit work. Although there is analysis that attempts to deal with complex mode change protocols (eg. [17]), the most straightforward and easily verified protocol to use is one that simply waits for a system idle tick and then makes the mode change [23]. As the system is idle at that point there can be no behavioural impact on the new LO-crit mode from the previous HI-crit mode. This issue has been further investigated in two recent papers [16], [20].

All but the simplest models of MCS require that the execution times of all jobs are monitored. Most RTOSs will allow this to be done for single processor systems (although the problem for multi-core platforms with shared buses remains an open issue). For LO-crit tasks, an adequate behaviour is for them to be prevented from executing for more than their $C(LO)$ budget. For HI-crit jobs there is no run-time benefit to be gained from capping their execution times, but the criticality mode switch must be made if any HI-crit job executes for its $C(LO)$ value without signaling completion.

B. Reducing LO-crit Tasks' Execution Time Budgets

One possible criticism of the above scheme is that if LO-crit tasks can have their priorities changed, this capability could be exploited in a security breach to undermine the assurances given to the HI criticality tasks. Another problem with the approach is that no guarantees can be given to short deadline LO-crit jobs executing after the mode change. In practice there is likely to be spare capacity available (once the HI-crit tasks have been scheduled in the high criticality mode) and this capacity could be used to guarantee at least some level of service to some of the LO-crit tasks. In this section we introduce a different mixed criticality model that explicitly retains (some) LO-crit work in the HI-crit mode.

We first introduce the model under the assumption that there is a specific level of LO-crit work that must be guaranteed; i.e. there is a schedulability test that will either accept, or not, a given task set. This test is then used, in the context of sensitivity analysis, to explore what levels of service are possible whilst retaining schedulability.

The *modified system model* is as follows. Each task, τ_i is defined by the parameters: T_i, D_i, L_i, $C_i(LO)$ and $C_i(HI)$. For HI-crit tasks $C_i(HI) \geq C_i(LO)$, for LO-crit tasks $C_i(LO) \geq C_i(HI)$. Note, for some LO-crit tasks $C_i(HI)$ may be zero meaning that no jobs of such tasks start their execution once the system is in the HI-crit mode. Static priorities (P_i) are assigned to the tasks according to Audsley's Optimal Priority Assignment algorithm [1] (as explained in [4], [24]). The system model is now defined by the following behaviours:

- The system starts in the LO-crit mode, and remains in that mode as long as all HI-crit tasks execute within their low criticality computation times ($C(LO)$).
- If any job of a HI-crit task τ_i executes for its $C_i(LO)$ value without completing then the system immediately moves to the HI-crit mode.
- The priorities of tasks are never modified.
- No job of a LO-crit task τ_k is allowed to execute for more than its $C_k(LO)$ parameter in the LO-crit mode or its $C_k(HI)$ parameter in the HI-crit mode; any attempt to do so will result in the task being suspended until its next release (at least T_k after its last release).
- There is no bound on the execution time of HI-crit tasks.
- If the system is in the HI-crit mode and there is an idle tick then the system can move back to the LO-crit mode and all LO-crit tasks can have their execution time budgets restored to their original $C(LO)$ values.

Note that a LO-crit job released in the LO-crit mode is not guaranteed to receive processing time of $C(LO)$ by its deadline, if there is a mode change during its execution and its budget is reduced. It is however guaranteed to receive processing time of $C(HI)$ by its deadline.

We can now give the schedulability test for this model, using the AMC-rtb (Method 1) approach. Equation (1) is again used to compute the worst-case response time of all tasks in the LO-crit mode. Considering the HI-crit mode, the starting point is (3). This equation assumes that no LO-crit job released after $R_i(LO)$ can interfere with a HI-crit task τ_i. Now we allow interference, but at a lower level. This is easily accommodated; firstly for HI-crit tasks:

$$R_i(HI) \;=\; C_i(HI) \;+ \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) \;+$$
$$\sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k(LO) \;+$$
$$\sum_{\tau_k \in \mathbf{hpL}(i)} \left(\left\lceil \frac{R_i(HI)}{T_k} \right\rceil - \left\lceil \frac{R_i(LO)}{T_k} \right\rceil \right) C_k(HI) \qquad (4)$$

Note that as $R_i(HI)$ is always greater than or equal to $R_i(LO)$ the final term in (4) is never negative. Equation (4) thus assumes the maximum possible number of releases of LO-crit tasks with the higher execution time.

An alternative form for (4) is available by simply rearranging the terms:

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

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

to give:

$$R_i(HI) = C_i(HI) + \sum_{\tau_j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil (C_k(LO) - C_k(HI)). \quad (5)$$

For a LO-crit task τ_i, which now continues after the mode change, but with a reduced execution time requirement of $C_i(HI)$, the above equation also applies; however, a tighter formulation is also possible. We note that if the task has already executed for $C_i(HI)$ in the LO-crit mode, then it has trivially met its requirements in the HI-crit mode. Therefore we need only consider the case where the mode change occurs before it has executed for $C_i(HI)$. Equation (1) computes the worst-case response-time for LO-crit tasks assuming that the task's own requirement is $C_i(LO)$, but here the only scenario of interest is when the mode change occurs before it has executed for $C_i(HI)$ which must be earlier as $C_i(HI) \leq C_i(LO)$. Hence (1) can be modified for LO-crit tasks to give:

$$R_i^*(LO) = C_i(HI) + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j(LO) \quad (6)$$

and (4) then becomes:

$$R_i(HI) = C_i(HI) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i^*(LO)}{T_k} \right\rceil C_k(LO) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left(\left\lceil \frac{R_i(HI)}{T_k} \right\rceil - \left\lceil \frac{R_i^*(LO)}{T_k} \right\rceil \right) C_k(HI) \quad (7)$$

again this can be rearranged to give:

$$R_i(HI) = C_i(HI) + \sum_{\tau_j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j(HI) + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i^*(LO)}{T_k} \right\rceil (C_k(LO) - C_k(HI)). \quad (8)$$

As indicated earlier, (4) and (7) (or (5) and (8)) could be used to test a specific application's requirements. More practically, they would be used to explore the design space – how much guaranteed capacity is available once all HI-crit tasks are validated? Sensitivity analysis [8], [18] could be used to explore this space. Possible questions to consider are:

- How many LO-crit tasks can have their full capacity (i.e. $C(HI) = C(LO)$) with the rest having $C(HI) = 0$?
- By how much must all the $C(LO)$ of LO-crit tasks be reduced (i.e. $C(HI) = \alpha \cdot C(LO)$) with $\alpha < 1$ to give a schedulable system?
- Can some or all LO-crit tasks employ alternative versions that take less resources?

With this implementation model the mode change back to LO-crit is straightforward. Again a low priority 'background' task can be used to implement the change back to LO-crit mode; however, now the only action of this task is to return the budgets for each LO-crit task to their larger LO-crit values ($C(LO)$ rather than $C(HI)$). This should be done atomically to avoid any potential race condition.

C. Increasing LO-crit Tasks' Periods

The elastic scheduling model [10] has been applied [22] to EDF scheduled mixed criticality systems to allow a LO-crit task to have its period extended after a mode change. We can apply this idea to fixed priority systems by allowing a LO-crit task to have two period values: $T_i(LO)$ and $T_i(HI)$ with $T_i(LO) \leq T_i(HI)$. After a mode change the task can be released again but with an extended period (and perhaps also a reduced budget). The equations of the previous section can be extended to include a revised period after the mode change. But as space is restricted this is left as an exercise for the reader.

D. Capacity Inheritance with the Budget Reduction Scheme

The previous subsections have introduced three different schemes for dealing with LO-crit tasks following a mode change to HI-crit. One reduces LO-crit tasks to, in effect, the background level; here they can utilise all available spare capacity, but the schedulability of LO-crit tasks after the criticality mode change is seriously undermined. The other schemes guarantees some level of service, but does not utilise spare capacity. This is a significant drawback as there is likely to be considerable spare capacity in the HI-crit mode. A HI-crit job may execute for more than $C(LO)$, but will most likely complete well before it has used its full $C(HI)$ budget – most of $\big(C(HI) - C(LO)\big)$ could be available for LO-crit jobs.

To improve the effectiveness of the second scheme two strategies are possible:

1) Use the schemes together – once a low criticality job has used up its $C(HI)$ budget its priority is lowered to its background level ($P_i(HI)$) where it can continue to execute.
2) The spare capacity from HI-crit jobs is directly assigned to LO-crit jobs.

The second strategy can exploit previously published techniques such as Extended Priority Exchange [21], Capacity Sharing [7] and History Rewriting [6] that were developed for combined hard and soft real-time fixed priority task sets. Within the context of mixed criticality systems these techniques would work as follows. Assume the system is in the HI-crit mode.

- All HI-crit tasks have a budget equal to their maximum, guaranteed, execution time $C(HI)$.
- At run-time the actual execution time of each HI-crit job is monitored;
- When a job with release time s and absolute deadline d (with $d = s + D$) completes, its actual execution time is noted (e) and its *gain* time (g) is computed ($g = C(HI) - e$); g is assumed to be non-negative.
- The gain time is available to be allocated to lower priority jobs.
- The gain time must be used by d (its expiry time).

The three techniques referenced above allocate the gain time in different ways. For Extended Priority Exchange [21]:

- The gain time is allocated to the budget of the next highest priority task that is ready to execute. (Note if there is no ready task, then the gain time is lost).

For Capacity Sharing [7]:

- An executing job first uses its own budget of $C(HI)$.
- When this budget is exhausted it 'pulls down' extra capacity from any available higher priority gain time.
- The LO-crit job is said to be *plugged* to the budget of the *host* HI-crit job.
- The plug is broken when the expiry time is reached, the capacity is exhausted or the job completes.

If there are a number of higher priority gain times available then any can be chosen and indeed more than one can be utilised, though only one at a time. Useful heuristics to use are: use the biggest g first, or use the earliest d first. An analysis of these heuristics and the implementation efficiency of the Capacity Sharing scheme is described in [7].

For History Rewriting [6] a retrospective reallocation of budgets is undertaken. The problem with Capacity Sharing is that the gain time has to be used before its expiry time or it is lost. This is not the case with History Rewriting which has the following characteristics:

- At the deadline (d) of a job the gain time is noted (g).
- A lower priority task that has executed for e before d is chosen and its budget is increased by $\max(g, e - g)$, g is reduced by this amount.
- Any remaining gain time is further allocated to lower priority tasks.

In effect, a job that was executing from its own budget is deemed to have been executing from the gain time of a higher priority job, and hence its own budget is intact and can be used to further the execution of the job. Further details of this approach are given in [6].

Job	Crit	C(LO)	C(HI)	s	d	e	g
τ_1	HI	1	4	0	8	2	2
τ_2	LO	6	4	4	12	-	-

TABLE I
TWO JOB EXAMPLE

For an example of History Rewriting consider the simple system of two jobs as defined in Table I. In the HI-crit mode both jobs are guaranteed 4 units of execution (although the LO-crit job would prefer 6). As τ_1 only executes for 2 units there is a gain time of 2 at time 2. But τ_2 is not active at time 2 and so the gain time cannot be utilised with Extended Priority Exchange or Capacity Sharing. With History Rewriting, however, the gain time of 2 becomes available at the deadline of τ_1 at time 8 after τ_2 has executed for 4 units. Two of these units can now be considered to be gain time and hence τ_2 can execute at time 8 for 2 more units thereby satisfying its full requirement.

IV. ROBUST PRIORITY ASSIGNMENT

For a dual-crit system $C(LO)$ values must, of course, be known. Once schedulability has been established however, it is possible to derive [19], using sensitivity analysis, a scaling factor f ($f > 1$) such that the system remains schedulable with all $C(LO)$ values replaced by $f \cdot C(LO)$. Using these scaled values at run-time will increase the robustness of the system, as LO-crit tasks will be able to execute for longer before they are suspended. Further, HI-crit tasks will be able to execute for longer without inducing a mode change.

Although the work of Santy et al. [19] allows the computation time of tasks to be increased it does this without modification to the priority ordering of the tasks. Here, we note that as the use of Audsley's Optimal Priority Assignment algorithm takes into account task computation times, a scheme that looks to increase robustness by extending the allowed execution times, will perform better if it also considers priority assignment.

In their work on Robust Priority Assignment algorithms, Davis and Burns [11] showed that for a general class of additional interference functions, Deadline Monotonic (DM) is both an optimal and a robust partial ordering for any subset of tasks that would on their own have DM as their optimal priority ordering. In this way, HI-crit tasks may be viewed as the subset of tasks which have DM as their optimal partial order with the LO-crit tasks constituting additional interference, and also vice-versa (assuming that all of the tasks have constrained deadlines). Hence an overall optimal and robust priority ordering can be achieved via a merge of the DM partial order of HI-crit tasks with the DM partial order of LO-crit tasks as described in [4]. This requires at most $2n-1$ task schedulability tests for a system of n tasks.

We note that changes to execution times will not affect the separate DM partial orderings of the two groups of tasks, but will potentially change the merge and hence the overall priority ordering. For example, if all $C(LO)$ values are close to zero but all $C(HI)$ values for HI-crit tasks are at their maximum level for schedulability then a total priority ordering in which all HI-crit tasks have higher priorities than all LO-crit tasks is optimal; however, if we increase the budgets for LO-crit execution and make $C(LO)$ equal to $C(HI)$ for all tasks then the optimal and robust priority ordering has the complete set of tasks in DM order.

Given the benefits that priority reassignment provides, we recommend this straightforward extension to Santy et al.'s approach [19].

V. Conclusions

This paper has addressed some of the issues that have been raised with what has become the standard analysis model for mixed criticality systems. As the tightest available analysis often assumes any released LO-crit job completes, there is no benefit for the actual run-time behaviour to require that LO-crit jobs are immediately abandoned. There is, however, a need for a real implementation to incorporate a means of returning the mode of the system to the initial mode if conditions are acceptable for this to occur. These needs are satisfied by the model presented in this paper. Other topics covered in this paper include allowing reduced but still guaranteed behaviour for low criticality tasks after a criticality mode change, and the use of robust priority assignment to reduce the likelihood of such mode changes.

This paper, like many on mixed criticality, is limited to only addressing dual criticality systems. This restriction helps to clarify descriptions. However, it is important that protocols do generalise to a realistic number of criticality levels. Perhaps up to five levels may be needed (see, for example, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards). In the models presented in this paper, HI-crit tasks have the particular property that they are not themselves abandoned if they execute for more than their budgets. As they have the highest criticality level the best run-time behaviour is always to allow them to continue to execute. When more than two levels are present then only the highest criticality level retains this ‘privilege’. All the others must be monitored and criticality mode changes affected where necessary. The models developed in this paper are easily extended to a small number of levels. With, say, five levels of criticality it is unlikely that a task will have five levels of service defined. Nevertheless the models defined do have the capability to be used in this way.

Taken together, the contributions of this paper aim to define a model for mixed criticality systems that has practical utility. It aims to ease the movement of the wealth of theoretical results that have appeared since 2007 into industrial practice.

Acknowledgements

This research has been supported in part by NSF grants CNS 0834270, CNS 0834132, and CNS 1016954; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; AFRL grant FA8750-11-1-0033 and EPSRC(UK) grant MCC (EP/K01 1626/1). We thank Rob Davis for this assistance.

References

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

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

[3] S.K. 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.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.

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

[6] G. Bernat, I. Broster, and A. Burns. Rewriting history to exploit gain time. In *Proc. Real-time Systems Symposium*, pages 328–335, Lisbon, Portugal, 2004. Computer Society, IEEE.

[7] G. Bernat and A. Burns. Multiple servers and capacity sharing for implementing flexible scheduling. *Real-Time Systems Journal*, 22:49–75, 2002.

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

[9] 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.

[10] G. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *IEEE Real-Time Systems Symposium*, pages 286–295, 1998.

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

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

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

[14] 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.

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

[16] M. Neukirchner, S. Quinton, and K. Lampka. Multi-mode monitoring for mixed-criticality real-time systems. In *Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2013.

[17] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *10th Euromicro Workshop on Real-Time Systems*, pages 172–179. IEEE Computer Society, 1998.

[18] 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.

[19] 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.

[20] F. Santy, G. Raravi, G. Nelissen, V. Nelis, P. Kumar, J. Goossens, and E. Tovar. Two protocols to reduce the criticality level of multiprocessor mixed-criticality systems. In *Proc. RTNS*, pages 183–192. ACM, 2013.

[21] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proc. 9th IEEE Real-Time Systems Symposium*, pages 251–258, 1988.

[22] 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.

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

[24] 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.

[25] Q. Zhao, Z. Gu, and H. Zeng. HLC-PCP: A resource synchronization protocol for certifiable mixed criticality scheduling. *Embedded Systems Letters, IEEE*, PP(99), 2013.