Fixed-priority scheduling of dual-criticality systems

S.K. 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: burns@cs.york.ac.uk

***Abstract*—In modern embedded platforms, safety-critical functionalities that must be certified correct to very high levels of assurance may co-exist with less critical software that is not subject to certification requirements. One seeks to satisfy two, sometimes contradictory, goals upon such mixed-criticality platforms: (i) certify the safety-critical functionalities under very conservative assumptions, and (ii) achieve high resource utilization during run-time, when actual behavior does not live up to the pessimistic assumptions under which certification was made. This paper makes two contributions: (i) it surveys different fixed-priority scheduling algorithms that have been proposed, that seek to balance these two requirements, and (ii) it completes prior work that performs a comparative evaluation of these different fixed-priority scheduling algorithms. It particularly focuses upon the period transformation technique for dual-criticality scheduling, since this technique has received relatively less attention in prior work.**

I. Introduction

In *mixed-criticality (MC)* systems, functionalities of different degrees of importance (or *criticalities*) are implemented upon a common platform. Such MC implementations are becoming increasingly common in embedded systems – consider, for example, Integrated Modular Avionics (IMA) in aviation [15] and the AUTOSAR initiative (`www.autosar.org`) for automotive systems.

The safety-critical functionalities of systems in many safety-critical application domains (including the aerospace and automotive domains) are subject to *certification* by statutory certification authorities (CAs). However, this increasing trend towards platform integration means that even in highly safety-critical systems it is typically the case that only a relatively small fraction of the overall system is actually of high criticality and subject to certification. The remainder of the system — the non safety-critical parts — do not need to pass certification by the CAs, although the system designer would nevertheless like to validate that all functionalities, including the non safety-critical ones, will perform correctly. Dealing with such mixed criticalities upon shared platforms have been identified as a central requirement for the emerging domain of cyber-physical systems (CPS), and solutions to the problems associated with validating and certifying the high-criticality parts of such systems are gaining wide recognition as being foundational enabling technologies for CPS.

Recurrent task models may be used to model event-driven phenomena that occur repeatedly: each event gives rise to a *job* that needs to be executed. The *sporadic tasks* model [14] is widely used for modeling such recurrent tasks. In this model, each task τ_i is characterized by a *worst-case execution time* (WCET) C_i, a *relative deadline* D_i, and a *minimum inter-arrival separation* T_i (for historical reasons, T_i is often called the *period* of the task). Such a task is assumed to generate an unbounded sequence of jobs at run-time, with successive job-arrivals separated by at least T_i time units and each job needing to execute for at most C_i time units by a deadline that is D_i time units after its arrival time.

Analysis of a system during design time is made under certain assumptions about the run-time behavior of the system. Certification Authorities (CAs) tend to be very conservative, and hence it is often the case that the assumptions demanded by the CA for obtaining certification are far more pessimistic than those the system designer would use during the system design process if certification was not required. In particular, Vestal [17, page 239] suggested that "the more confidence one needs in a task execution time bound [...] the larger and more conservative that bound tends to become." Accordingly, mixed-criticality systems were modeled in [17] as collections of sporadic tasks, with each task designated as being either a LO-criticality task or a HI-criticality one[1]. Each task was characterized by *two* WCET parameters rather than just one: a larger value denoting the CA's assessment as to what the task's WCET should be and a smaller one representing the system designer's less pessimistic estimate. It was not known prior to run-time which set of WCET estimates were the correct ones: this information could only be gleaned by actually executing the system and observing when the jobs complete execution. The challenge was to design a scheduling policy possessing the property that all jobs of all tasks would meet their deadlines during run-time if the system designers' WCET estimates turned out to be the correct ones, whereas all jobs of HI-criticality tasks would meet their deadlines (although jobs of LO-criticality tasks may miss theirs) if the system designers' WCET estimates turned out to be incorrect (some job executed for more than its WCET as estimated by the system designer) but the CA's estimates were respected (no job executed for more than its WCET as estimated by the CA).

Vestal [17] studied preemptive uniprocessor fixed-priority (FP) scheduling algorithms for scheduling mixed-criticality

[1]Vestal [17] considered more than two criticality levels, and did not use the terms LO and HI — for pedantic reasons, we restrict our discussion for now to two criticality levels and in general, we are not using the notation and terminology used in [17]. We explain later how the ideas presented in this paper extend to more than two criticality levels.

systems of such sporadic tasks. Among his results, he showed that the deadline-monotonic priority assignment strategy [12] is not optimal for mixed-criticality systems: this stands in contrast to "regular" (non mixed-criticality) systems where deadline-monotonic priorities are known to be optimal for constrained-deadline sporadic task systems. One of the reasons for this non-optimality is that the deadline-monotonic priority assignment is criticality-agnostic: in assigning priorities to tasks, the criticality of the tasks – whether they are HI-criticality or LO-criticality – is not taken into consideration. This can lead to less critical tasks that are assigned higher priority causing more critical tasks to miss their deadlines. To avoid such possibilities, Vestal explored the use of the technique of *period transformation* (PT) [16], which uses time-slicing to schedule the more critical task as though it had a smaller period and deadline (thereby enhancing its priority under deadline-monotonic priority assignment). Vestal also proposed, and evaluated, another FP scheme that was based on an application of Audsley's technique for priority assignment [3], [2], and that did not require period transformation.

Subsequent research gave rise to several non-FP algorithms for scheduling mixed-criticality systems of such sporadic tasks (see, e.g., [6], [8], [9], [4] — this list is not meant to be exhaustive). Huang et al. [10] have conducted a very thorough and detailed comparison of these different approaches, from various perspectives including their relative effectiveness, ease of implementation, overhead costs, etc.

The fixed priority scheduling of mixed-criticality sporadic task systems was revisited in [5], and the schemes studied by Vestal [17] classified, generalized, and extended. A new priority-assignment algorithm called AMC for *Adaptive Mixed Criticality* was also proposed. (The scheme based on the Audsley technique that Vestal proposed in [17] is referred to here as SMC, for *Static* Mixed Criticality.) It was shown in [5] that on preemptive uniprocessors, AMC strictly dominates SMC in the sense that any task system that is correctly scheduled by SMC is also correctly scheduled by AMC while the converse is not true: there are task systems that can be scheduled by a FP scheduler using AMC priorities but not by SMC priorities. As is routine in real-time scheduling theory his result was proved assuming an idealized scheduling model (e.g., no scheduling or preemption overhead); in addition, results of simulation studies that incorporated overhead considerations were reported in [5], that provide further validation of this conclusion even when the idealized scheduling assumptions do not hold.

Much of the comparison between the different approaches that was presented in [10] was based on extensive simulations. We believe that such implementation and simulation comparisons provide a wealth of useful information, and applaud the authors of [10] for having done that work. In addition, however, we believe that such simulation-based comparisons should be complemented by analyses based upon an idealized scheduling model and relatively simple example task systems, that serve to highlight the various advantages and disadvantages of the different approaches. It has been our experience that such studies provide great insight into the inner workings and inherent characteristics of the studied algorithms, and may lead to breakthroughs in the form of new and improved algorithms.

In this paper, we seek to provide such a comparison of the period transformation (PT) approach of [17] and the AMC scheduler of [5]. If implementation overheads are ignored, we show that the two schemes are *incomparable*: there are mixed-criticality sporadic task systems that can be FP-scheduled on a preemptive uniprocessor by each scheme, that the other scheme cannot schedule. However, it is well known that implementing PT can incur severe overhead, since the number of "jobs" that are being scheduled may increase significantly. (It is this fact that is primarily responsible for PT not becoming widely adopted in FP scheduling of non mixed-criticality real-time systems in practice, despite its pleasing theoretical property of offering 100% processor utilization.)

Organization. The mixed-criticality model studied in this paper is formally defined in Section II; a framework for fixed-priority (FP) scheduling of mixed-criticality systems is described in Section III. Three different FP scheduling approaches —Criticality Monotonic, Period Transformation, and Adaptive Mixed Criticality (AMC)— are described in Sections IV–VI. Section VII compares the different approaches.

II. System Model and Definitions

As described in Section I above, certification authorities (CAs) and system designers typically make different assumptions about the worst-case behavior of a system: the CA's assumptions are usually more conservative than those of the system designer. The system model that we assume is cognizant of the differences in the assumptions of the CA and the system designer and incorporates these differences.

We now formally define the mixed-criticality (henceforth often referred to as **MC**) workload model that is used in this paper, and explain terms and concepts used throughout the remainder of this document. As with traditional (i.e., non MC) real-time systems, we will model a MC real-time system τ as consisting of a finite specified collection of MC sporadic tasks, each of which will generate a potentially unbounded sequence of MC jobs.

MC jobs. Each job is characterized by a 5-tuple of parameters: $J_i = (a_i, d_i, \chi_i, c_i(\text{LO}), c_i(\text{HI}))$, where

- $a_i \in R^+$ is the release time.
- $d_i \in R^+$ is the deadline. We assume that $d_i \geq a_i$.
- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the job. A HI-criticality job (a J_i with $\chi_i = \text{HI}$) is one that is subject to certification, whereas a LO-criticality job (a J_i with $\chi_i = \text{LO}$) is one that does not need to be certified.
- $c_i(\text{LO})$ specifies the worst case execution time (WCET) estimate of J_i that is used by the system designer (i.e., the WCET estimate at the LO-criticality level).
- $c_i(\text{HI})$ specifies the worst case execution time (WCET) estimate of J_i that is used by the CA (i.e., the WCET estimate at the HI-criticality level). We assume that $c_i(\text{HI}) \geq c_i(\text{LO})$ (i.e., the WCET estimate used by the

system designer is never more pessimistic than the one used by the CA).

Behaviors. The MC job model has the following semantics. Job J_i is released at time a_i, has a deadline at d_i, and needs to execute for some amount of time γ_i. However, *the value of γ_i is not known beforehand, but only becomes revealed by actually executing the job until it* signals *that it has completed execution*. If J_i signals completion without exceeding $c_i(\text{LO})$ units of execution, we say that it has exhibited LO-criticality behavior; if it signals completion after executing for more than $c_i(\text{LO})$ but no more than $c_i(\text{HI})$ units of execution, we say that it has exhibited HI-criticality behavior. If it does not signal completion upon having executed for $c_i(\text{HI})$ units, we say that its behavior is erroneous.

MC sporadic tasks. Each sporadic task in the MC model is characterized by a 5-tuple of parameters: $\tau_k = (\chi_k, C_k(\text{LO}), C_k(\text{HI}), D_k, T_k)$, with the following interpretation. Task τ_k generates a potentially unbounded sequence of jobs, with successive jobs being released at least T_k time units apart. Each such job has a deadline that is D_k time units after its release. The criticality of each such job is χ_k, and it has LO-criticality and HI-criticality WCET's of $C_k(\text{LO})$ and $C_k(\text{HI})$ respectively.

A MC sporadic task system is specified as a finite collection of such sporadic tasks. As with traditional (non-MC) systems, such a MC sporadic task system can potentially generate infinitely many different MC instances (collections of jobs), each instance being obtained by taking the union of one sequence of jobs generated by each sporadic task.

An *implicit-deadline* MC sporadic task system satisfies the additional property that each task in the system has its relative deadline parameter equal in value to its period parameter: $D_k = T_k$ for all tasks τ_k in the systems. ***In the remainder of this paper we will, for ease of exposition, usually restrict our discussion to implicit-deadline sporadic task systems***, although our techniques and results are easily extended to apply to *constrained-deadline* sporadic task systems (systems in which $D_k \leq T_k$ for all tasks τ_k).

Scheduling MC sporadic task systems. A particular implicit-deadline sporadic task system may generate different instances of jobs during different runs. Furthermore, during any given run each job comprising the instance may exhibit LO-criticality, HI-criticality, or erroneous behavior. We define an algorithm for scheduling implicit-deadline sporadic task system τ to be *correct* if it is able to schedule every instance generated by τ such that

- If all jobs exhibit LO-criticality behavior, then all jobs receive enough execution between their release time and deadline to be able to signal completion; and
- If *any* job exhibits HI-criticality behavior, then all HI-criticality jobs receive enough execution between their release time and deadline to be able to signal completion.

Note that if any job exhibits HI-criticality behavior, we do not require any LO-criticality jobs (including those that may have arrived before this happened) to complete by their deadlines. This is an implication of the semantics of certification: informally speaking, the system designer fully expects that all jobs will exhibit LO-criticality behavior, and hence is only concerned that they behave as desired under these circumstances. The CA, on the other hand, allows for the possibility that some jobs may exhibit HI-criticality behavior, and requires that all HI-criticality jobs nevertheless meet their deadlines.

III. Fixed-priority scheduling of MC systems

We are concerned here with the schedulability analysis of mixed criticality (MC) implicit-deadline sporadic task systems that are being scheduled by the standard fixed priority (FP) preemptive dispatcher on a single processor.

The notion of FP dispatching is pretty well understood in the context of non-MC task systems: each task is assigned a distinct priority and during run-time at each instant the currently active job generated by the task with the greatest priority is chosen for execution. For non-MC implicit-deadline sporadic task systems, it is known [13] that the Rate-Monotonic (RM) priority assignment strategy is optimal in the sense that if priorities can be assigned to a given task system such that all jobs of all tasks will always meet their deadlines, then all jobs of all tasks will also always meet their deadlines if priorities are assigned to tasks in rate-monotonic order: tasks with smaller period are assigned higher scheduling priority.

In discussing the scheduling of MC systems, some additional implementation details must be made explicit: in particular, what kind of support for mixed-criticality implementation is provided by the platform upon which the system is being implemented. An important form of platform support is the ability to *monitor* the execution of individual jobs, i.e., being able to determine how long a particular job has been executing. If such support is available, then the run-time dispatcher is able to determine if and when the behavior of an executing job transits from LO-criticality to HI-criticality behavior. A strong case can be made for this ability to be part of the standard mechanisms for safety-critical applications. Such functionality is already commonly available on many real-time platforms and is widely assumed in, for example, many implementations of servers (e.g., [1]), or in real-time "open" environments that support the policing of individual jobs or of collections of jobs.

Depending on whether support for run-time monitoring and/or policing is available or not (and if available, what use is made of it), three different FP scheduling schemes were evaluated in [5]: *Criticality Monotonic* (CM), a standard scheme that is widely used in industrial practice; *Static Mixed Criticality* (SMC); and *Adaptive Mixed Criticality* (AMC). In Criticality Monotonic, all HI-criticality tasks are assigned higher priority than all LO-criticality tasks. The other two schemes, SMC and AMC, allow the priorities of different criticality tasks to be interleaved – it was shown in [17], [5] that such interleaving improves schedulability. One variant of the static scheme (SMC) polices job executions: it does not

allow jobs of any LO-criticality task τ_i more than $C_i(\text{LO})$ units of execution. The adaptive scheme (AMC) goes further and does not allow LO-criticality jobs to execute at all if some job of any task τ_i seeks to execute for more than $C_i(\text{LO})$ time units.

IV. Criticality Monotonic Scheduling

In Partitioned Criticality, commonly called Criticality Monotonic (CM), scheduling, all HI-criticality tasks are assigned higher priority than all LO-criticality tasks. Within each criticality, priorities may be assigned according to the *rate monotonic* (RM) priority assignment scheme [13], which is known to be optimal for implicit-deadline regular – non-MC – task systems.

This partitioned approach has the advantage that the scheduling of HI-criticality jobs is not impacted in any manner by LO-criticality jobs. No additional run-time support (beyond that needed by a standard fixed-priority dispatcher in order to implement a fixed-priority scheduler in "regular" – non-MC – systems) is required for implementing such a FP scheduling scheme. However, these advantages come at a significant cost in terms of schedulability, in the sense that CM fails to correctly schedule very many systems that can be correctly FP-scheduled using some other priority assignment scheme. Consider the following example:

Example 1: Let τ denote a task system with two tasks τ_1 and τ_2, with parameters as follows:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i
τ_1	HI	5	10	20
τ_2	LO	2	2	4

CM priority-assignment would require that τ_1 be assigned higher priority, and τ_2's jobs could miss deadlines even if no jobs execute beyond their LO-criticality WCET's (consider, e.g., the scenario in which jobs of both tasks arrive simultaneously). It may be verified that assigning τ_2 higher priority yields a correct scheduling strategy.

The efficacy of CM scheduling may be shown to be arbitrarily pessimistic by increasing the period of τ_1 in this example.

V. Period Transformation

The technique of period transformation (PT) [16] may be applied to avoid the kind of schedulability loss observed in Example 1 when other factors (in this case, criticalities) prohibit the assignment of priorities in rate-monotonic (RM) order. Below we first illustrate the use of PT by applying it to Example 1. We then describe the technique as presented in [16], and its application to mixed-criticality systems as described in [17]. We conclude this section with a discussion regarding the benefits and drawbacks of PT that have been identified in the scheduling of regular (not mixed-criticality) task systems.

To obtain a priority ordering for the task system of Example 1 that is compliant with both periods and criticalities, the PT technique would reduce task τ_1's period to 4, by scaling down by a factor $20/4 = 5$; the WCET's would similarly be scaled by a factor of 5 so that $C_1(\text{LO}) = 1$ and $C_1(\text{HI}) = 2$:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i
τ'_1	HI	1	2	4
τ_2	LO	2	2	4

The CM priority assignment for this system is now consistent with the rate-monotonic priority assignment.

PT for mixed-criticality tasks. The general technique presented in [17] is as follows. Let T_j denote the smallest period of any LO-criticality task. Each HI-criticality task τ_i with $T_j > T_i$ is 'transformed" by having its parameters scaled down by a factor $\lceil T_i/T_j \rceil$. Since $\lceil T_i/T_j \rceil \leq (T_i/T_j)$, the period T'_i of the transformed task satisfies

$$T'_i = \frac{T_i}{\lceil T_i/T_j \rceil} \leq \frac{T_i}{T_i/T_j} \leq T_j$$

and a rate-monotonic priority assignment would assign τ_i greater priority than τ_j.

And how would a mixed-criticality FP run-time dispatcher schedule the transformed task system during run-time? Let us revisit Example 1 and consider a job of τ_1 that arrives at time-instant 0; in the transformed system this is represented by five jobs of τ'_1 arriving at time-instants $0, 4, 8, 12$, and 16 (see Figure 1).

Now according to the semantics of the mixed-criticality model as described in Section II above, this job of τ_1 must execute for more than five units of execution before we can determine that its behavior is a HI-criticality one (which would mean that jobs of the LO-criticality task τ_2 need not be executed correctly). With regards to the *transformed* task system, we cannot however conclude that the behavior is a HI-criticality one simply if the first job of τ'_1 executes for more than one time unit without signaling completion – indeed, even the LO-criticality assumption is that this job will execute for up to five time units. Hence, the first few jobs of τ'_1 must each be allowed to execute for up to their HI-criticality WCET's; it is only if the *third* job – the one released at time-instant 8 – executes beyond one time unit without signaling completion that we will know that the behavior is a HI-criticality one. Hence the first few jobs of the transformed task τ'_1 will be executed for a duration equal to their HI-criticality WCETs even if the overall behavior of the system is a LO-criticality one. (On the other hand, the last few jobs of τ'_1 will require no execution at all in a LO-criticality behavior.)

Discussion. The technique of period transformations (PT) had originally been introduced [16] with the goal of enhancing schedulability by making a (regular – i.e., not mixed-criticality) task system *harmonic*[2]. The benefit of applying such a transformation derives from the result [11] that any harmonic implicit-deadline sporadic task system with total utilization ≤ 1 is scheduled correctly by RM, where the *utilization* $U(\tau)$ of an implicit-deadline sporadic task system τ is defined to be the sum over all the tasks in the system of

[2]A task system is said to be harmonic if it is the case that for any pair of tasks the period of one is an integer multiple of the other.

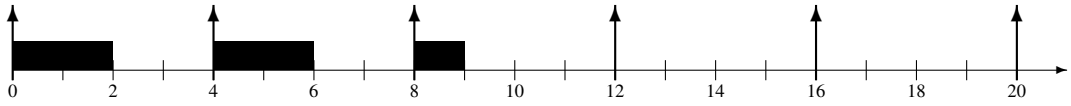

Fig. 1. PT schedule for the task system of Example 1. A job of τ_1, with scheduling window $[0, 20)$ is modeled as five jobs of τ'_1, with scheduling windows $[0, 4)$, $[4, 8)$, $[8, 12)$, $[12, 16)$, and $[16, 20)$ respectively. The original job would signal a transition to HI-criticality behavior by executing for more than five time units; the third job of the transformed task must execute for more than one time unit for the same information to be revealed. During a LO-criticality behavior, therefore, the jobs released at time-instants 12 and 16 have an execution requirement equal to zero.

the ratio of the task's WCET to its period parameter:

$$U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \frac{C_i}{T_i}$$

Since $U(\tau) \leq 1$ is a necessary condition for τ to be schedulable on a unit-speed processor by any scheduling algorithm, the result of [11] implies that RM is an optimal scheduling algorithm for task systems that happen to be harmonic.

Now, PT can be used to make *any* sporadic task system a harmonic one, by simply reducing the periods of all the tasks to the greatest common divisor (gcd) of the original tasks' periods. Why then is it not widely used? Why is it generally accepted that while RM scheduling has benefits, it has substantial drawbacks as well,[3] perhaps the most significant one being this utilization loss? After all, PT could be applied to render the task system harmonic and thereby ensure that RM is optimal from the utilization perspective. The reason is *overheads*: by splitting a single job up into multiple smaller ones, PT can significantly increase the scheduling overhead by increasing the number of scheduling decisions that must be made during run-time, the number of preemptions and context switches that occur, etc.

To summarize: if overheads are ignored then PT can be applied to make RM an optimal scheduling discipline. However, overheads exist in real systems, which explains why, despite its nice theoretical properties, the use of PT has not really become wide-spread in practice.

Due to this overheads issue, the application of PT to dual criticality systems (as presented in [17]) only transforms the HI-criticality tasks, and only so that their periods become equal to the shortest LO-criticality task. So LO-criticality tasks are not transformed, and the shorter periods of HI-criticality task are not used to determine the greatest common divisor for the transformations. As a result, the utilisation bound of 1 is no longer attained, but there are less transformations and hence less overhead.

VI. Adaptive Mixed Criticality - AMC

We now describe the adaptive dispatching scheme introduced in [5]. This scheme is *adaptive* in the sense that it monitors the execution of jobs during runtime; if any job executes for a duration greater than the LO-criticality WCET of the task that generated it, it immediately *drops* all LO-criticality jobs and only executes HI-criticality ones.

[3]See [7] for an interesting discussion regarding the relative benefits and drawbacks of RM and EDF scheduling.

An adaptive dispatcher. The algorithm used for run-time dispatching of jobs is provided with a mixed-criticality sporadic task system along with an assignment of unique distinct priorities to the tasks in the system. Dispatching of jobs for execution occurs according to the following rules:

1) The dispatcher maintains a *criticality level indicator* Γ, initialized to LO.
2) While $(\Gamma \equiv \text{LO})$, at each instant the waiting job generated by the task with highest priority is selected for execution.
3) If a job executes for more than its LO-criticality WCET without signaling that it has completed execution, then $\Gamma \leftarrow \text{HI}$.
4) Once $(\Gamma \equiv \text{HI})$, LO-criticality jobs will *not* be executed. Henceforth, therefore, at each instant the waiting job generated by the HI-criticality task with the highest priority is selected for execution.
5) An additional rule could specify the circumstances when Γ gets reset to LO. This could happen, for instance, if no HI-criticality jobs are active at some instant in time. (We will not discuss the process of resetting $\Gamma \leftarrow \text{LO}$ any further in this paper.)

Priority assignment. We now describe the AMC priority-assignment scheme for assigning the priorities that are used by the adaptive dispatcher described above. This priority assignment is done according to the Audsley Optimal Priority Assignment strategy [2]. That is, some task is identified that may be 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.[4] It was shown in [5] that within each criticality level (i.e., when comparing different LO-criticality tasks or different HI-criticality tasks), tasks can be assigned priorities in RM order. Hence in seeking to determine the lowest-priority task it suffices to consider only the largest-period LO-criticality task, and the largest-period HI-criticality one. This is done by essentially determining (an upper bound on) the worst-case response time of a task τ_k if it were assigned lowest priority, and checking whether this is no larger than the period 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 then their own LO-criticality WCETs; if τ_k is a HI-criticality task ($\chi_k = \text{HI}$), then we must bound its worst-case

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

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 $T_i \geq L_{\text{LO}}$ may be assigned 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 lowest priority. Similarly, any HI-criticality task τ_i satisfying $T_i \geq L_{\text{HI}}$ may be assigned lowest priority.

So we first compute L_{LO}. Based on results from Response-Time Analysis (RTA) [18] it is straightforward to observe that L_{LO} can be set equal to the smallest positive value of t satisfying the response-time formula

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

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 on 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 criticality level indicator Γ sees its value changed from LO to HI. (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 $\lceil t_1/T_j \rceil$ 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 $\sum_{j:\chi_j=\text{LO}} \left(\lceil L_{\text{LO}}/T_j \rceil \cdot C_j(\text{LO}) \right)$. 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 $\sum_{j:\chi_j=\text{HI}} \left(\lceil t/T_j \rceil \cdot C_j(\text{HI}) \right)$. 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

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

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

The algorithm for determining a lowest-priority task is summarized in Figure 2.

VII. Comparing AMC and PT

We now show that, unlike in the case of regular (non-MC) implicit-deadline sporadic task systems, for which the 100% utilization bound of PT implies optimality, there are

- Determine L_{LO} as the smallest positive value of t satisfying Equation 1. If there is some LO-criticality task τ_i with $T_i \geq L_{\text{LO}}$, assign it lowest priority.
- **Else** determine L_{HI} as the smallest positive value of t satisfying Equation 2. If there is some HI-criticality task τ_i with $T_i \geq L_{\text{HI}}$, assign it lowest priority.
- **Else** declare failure.

Fig. 2. AMC: Determining the lowest-priority task.

MC implicit-deadline sporadic task systems that are FP-schedulable under AMC but are not FP-schedulable with PT. And since this result runs contrary to that for non-MC systems, we will only be able to show this result by exploiting those unique attributes of mixed-criticality systems that are not found in regular systems.

Consider once again the task system of Example 1, a PT schedule for which is illustrated in Figure 1. Modify this task system just a little bit, by *increasing* the WCETs of τ_2 by an arbitrarily small amount $\epsilon > 0$:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i
τ_1	HI	5	10	20
τ_2	LO	$2+\epsilon$	$2+\epsilon$	4

Since the periods of the tasks are the same, PT would result in exactly the same transformation as described in Section V; the resulting schedule for the HI-criticality task τ_1 would again be as shown in Figure 1. It is evident that if a job of τ_2 were to arrive concurrently with one of τ_1 then τ_2's job would miss its deadline in such a schedule; this system is therefore *not* FP schedulable under PT.

In attempting to assign priority under AMC (see the pseudocode in Figure 2), we would first determine L_{LO} as the smallest positive value of t satisfying

$$t = \left\lceil \frac{t}{4} \right\rceil (2+\epsilon) + \left\lceil \frac{t}{20} \right\rceil 5.$$

It is evident that under the assumption that $\epsilon < 1/3$, the smallest positive value of t satisfying this recurrence is $(11+3\epsilon)$. Since this is larger than the period of the LO-criticality task τ_2, we conclude that τ_2 may not be assigned lowest scheduling priority.

We next seek to determine L_{HI}; this is the smallest positive value of t satisfying

$$t = \left\lceil \frac{11+3\epsilon}{4} \right\rceil (2+\epsilon) + \left\lceil \frac{t}{20} \right\rceil 10,$$

which, it may be verified, is $(16+3\epsilon)$. Since $(16+3\epsilon) < 20$, we conclude that τ_1 may indeed be assigned lowest scheduling priority; accordingly under AMC τ_2 is assigned higher scheduling priority than τ_1.

This example proves the following result:

Lemma 1: There are MC implicit-deadline sporadic task systems that are FP-schedulable under AMC that are not schedulable under PT. ■

What about the other direction? — are there implicit-deadline sporadic task systems that are FP-schedulable with PT but that cannot be scheduled using AMC? Recall from Section V that the technique of period transformations can be used to make RM-schedulable systems that would not be fixed-priority schedulable if PT were not used; indeed, if run-time overheads are ignored we had seen that the application of PT results in RM becoming an optimal algorithm for scheduling regular (non mixed-criticality) task systems.

Vestal [17] had proposed the use of PT in mixed-criticality scheduling only for the purpose of making the criticality-monotonic priority ordering consistent with the rate-monotonic one. However, under an idealized scheduling model (in which overheads are ignored) there is no particular reason why we could not go further and use PT to also enhance schedulability. That is, rather than using PT only to reduce the periods of HI-criticality tasks and thereby make a rate-monotonic priority assignment consistent with a criticality-monotonic one, one could also apply additional period transformations to reduce all the tasks' periods to the greatest common divisor of their original periods, and thereby make the system harmonic — doing so increases the utilization bound of the system in both LO-criticality and HI-criticality behaviors.

By making use of this aspect of PT, it is possible to construct MC task systems that are schedulable using PT but not by AMC: consider the following example.

Example 2: Let τ be comprised of the following two tasks:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i
τ_1	HI	5	5	10
τ_2	LO	2	2	4

(Note that since each task's LO-criticality and HI-criticality WCET's are equal, i.e., $C_i(\text{LO}) = C_i(\text{HI})$ for all τ_i, this task system happens to be a non-MC system being represented using mixed-criticality notation.)

PT would reduce the periods of both tasks to $\gcd(4, 10) = 2$, and the resulting harmonic system has utilization equal to 1 and is therefore RM-schedulable. However, no FP scheduling algorithm (and hence, not AMC) can schedule this system without transforming the periods: this can be seen according to the following argument.

- Since RM is known to be an optimal priority assignment [13], if any FP scheduling algorithm can schedule this system then so can RM.
- Under RM, task τ_1 would have lower priority.
- The worst-case response time of τ_1's jobs is then given by the smallest value of t satisfying

$$t = 5 + \left\lceil \frac{t}{4} \right\rceil 2$$

which is 11 (see Figure 3).

This example of a task system that is FP schedulable only after period transformation serves to prove the following result:

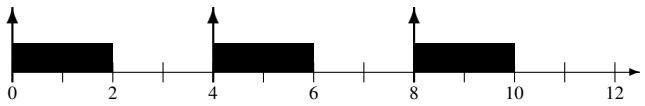

Fig. 3. Illustrating the response time of τ_1.

Lemma 2: There are MC implicit-deadline sporadic task systems that are FP-schedulable under PT that are not schedulable without PT (and hence, not under AMC). ■

Lemma 1 an Lemma 2 together prove the following theorem.

Theorem 1: The two implementation schemes, PT and AMC, are incomparable. Neither dominates the other. ■

We would like to highlight that the enhanced capabilities of PT in FP scheduling that was illustrated in Example 2 above holds for regular task systems, not just mixed-criticality ones (indeed, as pointed out above, the task system in Example 2 is not really a mixed-criticality one). Hence this benefit of period transformation is orthogonal to mixed criticalities — it is a direct consequence of the fact that PT has resulted in a harmonic system and has consequently raised the utilization bound in both LO-criticality and HI-criticality behaviors.

This gives rise to the question: does PT help when it does not result in an increase in utilization bound? The answer appears to be "no", in the sense that if the system is already harmonic then if PT can schedule the system then so can AMC - there is no further advantage that PT provides. We prove this conjecture for a two task systems.

Theorem 2: If a harmonic two task system is schedulable under PT then it is also schedulable under AMC.

Proof: The two tasks are defined using the usual symbols:

τ_i	χ_i	$C_i(\text{LO})$	$C_i(\text{HI})$	T_i
τ_1	HI	$C_1(LO)$	$C_1(HI)$	T_1
τ_2	LO	$C_2(LO)$	-	T_2

with $T_1 = nT_2$ (n is a positive integer).

For schedulability in the LO mode:

$$C_1(LO)/T_1 + C_2(LO)/T_2 \leq 1,$$

i.e.

$$C_1(LO)/n + C_2(LO) \leq T_2.$$

So if $C_2(LO)$ is fixed then

$$C_1(LO) \leq n(T_2 - C_2(LO)) \tag{3}$$

For period transformation define $\tau_1^{'}$ to have period T_2 and computation time $C_1(HI)/n$. This task now has highest priority and same period as τ_2 and hence the maximum possible value of $C_2(LO)$ is $T_2 - C_1(HI)/n$ ie

$$C_2(LO) \leq T_2 - C_1(HI)/n \tag{4}$$

Under AMC assume τ_2 has to have highest priority for its schedulability. It makes available to τ_1: $T_2 - C_2(LO)$ every T_2. To satisfy its LO-crit requirement, τ_1 will require I slots:

$$I = \left\lceil \frac{C_1(LO)}{T_2 - C_2(LO)} \right\rceil$$

So during a mode change there will be I jobs of τ_2 before the mode change is signalled. The response time of τ_1 during the mode change is therefore given by:

$$R_1 = IC_2(LO) + C_1(HI)$$

For schedulability $R_1 \leq T_1 = nT_2$. So

$$R_1/n \;=\; \frac{IC_2(LO)}{n} + \frac{C_1(HI)}{n} \;\leq\; T_2$$

Substituting for $C_1(HI)/n$ from eq (4) gives

$$\frac{IC_2(LO)}{n} + (T_2 - C_2(LO)) \leq T_2$$

ie

$$\frac{IC_2(LO)}{n} \leq C_2(LO)$$

and hence

$$I/n \leq 1 \tag{5}$$

The maximum value of I comes when $C_1(LO)$ is maximum, so from definition of I and equation (3):

$$I^{max} = \left\lceil \frac{n(T_2 - C_2(LO))}{T_2 - C_2(LO)} \right\rceil = \lceil n \rceil = n$$

Inequality (5) becomes:

$$I/n \leq I^{max}/n = 1 \leq 1$$

which is clearly true. So $R_1 \leq T_1$ and hence the system is schedulable by AMC.

■

In future work we will attempt to extend this proof to an arbitrary number of tasks with mixed levels of criticality. However, the theorem does seem to imply that the advantage PT has comes entirely from its ability to generate harmonic tasks sets. Which, of course, it can only do optimally if overheads are ignored.

VIII. Conclusion

Obtaining certification for safety-critical applications in mixed-criticality systems is particularly challenging due to potential interference by lower-criticality applications. Approaches based on preventing such interference by enforcing strict isolation amongst applications of different criticalities can lead to poor resource utilization, thereby defeating one of the major objectives of platform-sharing.

Recently, some exciting research has been done on designing scheduling strategies that are easy to certify and that simultaneously ensure efficient use of platform resources. In this paper we focus our attention upon a subset of these strategies: those that seek to do fixed-priority scheduling on preemptive uniprocessors. First, we survey the different approaches to fixed-priority scheduling that have previously been proposed. Next, we continue the comparative evaluation of these different fixed-priority scheduling algorithms that has been reported in [10], [5]. We particularly focus upon the period transformation (PT) technique, since this technique seems to have received relatively less attention in prior work. We compare this technique with AMC - Adaptive Mixed Criticality.

We have demonstrated that these two approaches are incomparable; neither techniques dominates the other. We also show that the benefits that come from period transformation are due to its ability to turn any task set into one that has harmonic periods and hence has a utilization bound of 1. This advantage is however largely undermined by the overheads involved in implementing period transformation. Certainly for a large number of high criticality tasks with arbitrary periods the greatest common divisor of the periods is likely to be very small (perhaps 1/100 of some periods) and hence would require an excessive number of context switches (in addition to the necessary task monitoring and enforcement of these switches). The other technique, AMC, is not without its overheads (as low criticality task computation times need to be monitored) but this overhead is more likely to scale with the number of tasks.

Although PT seems to be of limited practical application, it does have the useful property that the high criticality tasks are given the high priorities. And as a result no wayward behaviour of low criticality tasks can ever impact on the performance of high criticality tasks. This support for isolation is an important asset for implementing mixed criticality systems.

Our results are easily generalized in several directions; we briefly list a couple of these extensions below.

Constrained-deadline sporadic task systems. To keep the discussion simple, we have chosen to restrict our attention here to implicit-deadline sporadic task systems: task systems in which the relative-deadline and period parameters of a task are equal, for all the tasks in the system. Most of our results generalize for constrained-deadline systems (the relative deadline of a task is no larger than its period). We have not considered arbitrary task systems (those that are not constrained-deadline).

More than two criticality levels. In this paper we have assumed two criticality levels, denoted LO and HI. In many safety-critical application domains, there may be more than two levels specified. For instance, the DO-178B standard specifies five criticality levels, while the IEC 61508 international standard for industrial use recommends four different *Safety Integrity Levels (SILS)*. Our techniques may be extended to deal with multiple criticality levels, by separately considering the transition between each adjacent pair of criticality levels.

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).

REFERENCES

[1] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 3–13, Madrid, Spain, December 1998. IEEE Computer Society Press.

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

[3] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, England, 1991.

[4] 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 *Proceedings of the 2012 24th Euromicro Conference on Real-Time Systems*, ECRTS '12, Pisa (Italy), 2012. IEEE Computer Society.

[5] S. Baruah, A. Burns, and R. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[6] S. Baruah and S. Vestal. Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Prague, Czech Republic, July 2008. IEEE Computer Society Press.

[7] G. Buttazzo. Rate-monotonic vs. EDF: Judgement day. *Real-Time Systems: The International Journal of Time-Critical Computing*, 29(1):5–26, 2005.

[8] D. de Niz, K. Lakshmanan, and R. R. Rajkumar. On the scheduling of mixed-criticality real-time task sets. In *Proceedings of the Real-Time Systems Symposium*, pages 291–300, Washington, DC, 2009. IEEE Computer Society Press.

[9] N. Guan, P. Ekberg, M. Stigge, and W. Yi. Effective and efficient scheduling for certifiable mixed criticality sporadic task systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, Vienna, Austria, 2011. IEEE Computer Society Press.

[10] H.-M. Huang, C. Gill, and C. Lu. Implementation and evaluation of mixed-criticality scheduling algorithms for periodic tasks. In *Proceedings of the 2012 IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '12, Beijing (China), 2012. IEEE Computer Society.

[11] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 160–171, 1991.

[12] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

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

[14] A. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[15] P. J. Prisaznuk. Integrated modular avionics. In *Proceedings of the IEEE 1992 National Aerospace and Electronics Conference (NAECON 1992)*, volume 1, pages 39–45, May 1992.

[16] L. Sha, J. Lehoczky, and R. Rajkumar. Solutions for some practical problems in prioritized preemptive scheduling. In *Proceedings of the Real-Time Systems Symposium*. IEEE Computer Society Press, Dec. 1986.

[17] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proceedings of the Real-Time Systems Symposium*, pages 239–243, Tucson, AZ, December 2007. IEEE Computer Society Press.

[18] A. Wellings, M. Richardson, A. Burns, N. Audsley, and K. Tindell. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8:284–292, 1993.