Response-time analysis of mixed criticality systems with pessimistic frequency specification

Sanjoy Baruah Bipasa Chattopadhyay
The University of North Carolina

***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 are 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 describes efforts at designing fixed-priority scheduling algorithms that balance these two requirements, when scheduling recurrent tasks that are triggered by external events of unknown exact frequency.**

I. Introduction

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). The current trend in embedded systems towards integrating multiple functionalities upon a shared platform, as exemplified by Integrated Modular Avionics [17] in aerospace, and the AUTOSAR (AUTomotive Open System ARchitecture — www.autosar.org) specifications for the automotive industry, 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. Such task systems in which different tasks have different "importances" or *criticalities* and must therefore be validated correct to different levels of assurance, are commonly known as *mixed criticality* systems. 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. If these events occur arbitrarily frequently then it is not possible to offer deterministic guarantees as to when these jobs will be serviced. Often, however, *lower bounds* can be estimated beforehand on the minimum amount of time that must elapse between successive occurrences of these events (equivalently, upper bounds can be obtained on the frequency of occurrence), and performance guarantees can be made under the assumption that these bounds are correct. The *3-parameter sporadic tasks* model [16] is widely used for modeling such recurrent tasks. In the 3-parameter sporadic tasks model (henceforth simply called the sporadic tasks 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.

As we had stated above, system analysis 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. For instance, for the purposes of analysis by the CA a sporadic task may be assigned a larger WCET (the CA expects the task to execute for longer), and/ or a smaller period parameter (the CA expects the triggering events to occur more frequently), than would be assumed by the system designer. Most previous papers dealing with the certification-cognizant scheduling of mixed-criticality real-time systems have dealt with increased pessimism in WCET with increasing criticality levels. That is, they have considered the case where the period parameters are exactly known but the WCETs are not, and the system designer uses a less pessimistic (i.e., smaller) WCET estimate than the CA. The possibility of having pessimism in the period parameter was mentioned in [5], [10]; to our knowledge, though, [3] was the first paper to analyze the effects of such pessimism, in the context of preemptive uniprocessor scheduling. The current paper extends the work in [3] to ***fixed-priority (FP)*** preemptive uniprocessor scheduling.

Organization. The mixed-criticality model studied in this paper is formally defined in Section II; the problem considered is described in Section III. Some related work is described in Section IV. Different priority-assignment algorithms are presented and evaluated in Sections V-VII; their run-time computational complexity is discussed in Section VIII. These different algorithms are compared both theoretically (in Section IX) and experimentally, via simulation (in Section X).

Section XI describes how these results may be extended in several directions, to deal with more general workload models.

II. Model

Although the discussion in Section I only considered systems with two distinct criticality levels – tasks needing certification and tasks not needing to be certified – a single system may in general contain more criticality levels. For instance, the RTCA DO-178B software standard (widely used in the avionics industry) specifies five different criticality levels, with the system designer expected to assign one of these criticality levels to each task. Similar standards are in use in the automotive industry, and for factory automation – these standards, too, specify more than two criticality levels. However in this paper we will, for ease of exposition, restrict our attention to *dual-criticality* systems – systems with just two criticality levels, which we will denote as LO and HI.

We will characterize a *mixed-criticality (MC) sporadic task* by a 5-tuple of parameters:

$$\tau_i = (\chi_i, C_i, D_i, T_i(\text{LO}), T_i(\text{HI})),$$

where

- $\chi_i \in \{\text{LO}, \text{HI}\}$ denotes the criticality of the task; we assume that a τ_i with $\chi_i = \text{HI}$ must be certified correct by the CA while a τ_i with $\chi_i = \text{LO}$ is not subject to certification;
- C_i denotes the WCET of the task, and D_i its relative deadline;
- $T_i(\text{LO})$ denotes the minimum amount of time that the system designer expects will elapse between the arrival times of successive jobs of τ_i; while
- $T_i(\text{HI})$ denotes the minimum amount of time that the CA expects will elapse between the arrival times of successive jobs of τ_i.

We make the assumption that $T_i(\text{HI}) \leq T_i(\text{LO})$ for all tasks τ_i; i.e., the CA is *more pessimistic* than the system designer in that they assume that jobs of a task may arrive more frequently. We also restrict our attention in this paper to *constrained-deadline* sporadic task systems [7]; in the context of mixed-criticality sporadic tasks of the form considered here, this translates to requiring that $D_i \leq T_i(\text{HI})$ for each task τ_i.

A *MC sporadic task system* τ is a finite collection of MC tasks: $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. We study the problem of scheduling such task systems on a fully preemptive uniprocessor using fixed-priority scheduling (see Section III).

We close this section by formally defining what we expect of a scheduling algorithm for mixed-criticality systems.

Behaviors. Any execution of the system satisfying the property that for each task τ_i, successive jobs of τ_i arrive[1] at least $T_i(\text{LO})$ time units apart, is said to exhibit a LO-*criticality behavior*. Any execution of the system that does not exhibit LO-criticality behavior but which satisfies the property that for each task τ_i, successive jobs of τ_i arrive[1] at least $T_i(\text{HI})$ time units apart is said to exhibit a HI-*criticality behavior*. Any execution that does not exhibit HI-criticality or LO-criticality behavior is said to be *erroneous*.

[1]Equivalently, the external events that trigger the arrival of jobs of τ_i occur

Correctness. An algorithm for scheduling MC task systems is said to be *correct* if and only if it satisfies the property that

- Each job of each task τ_i completes execution within D_i units of arrival, in all LO-criticality behaviors of the system; and
- Each job of each HI-criticality task τ_i (i.e., all τ_i with $\chi_i = \text{HI}$) completes execution within D_i units of arrival, in all HI-criticality behaviors of the system.

III. Fixed-priority scheduling of MC systems

In this paper we consider the schedulability analysis of a mixed criticality (MC) system 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. In MC systems, however, some additional details must be spelled out: in particular, *does the run-time dispatcher monitor arrivals in order to detect changes in behavior?* That is, does the run-time dispatcher keep track of arrival times of successive jobs of individual tasks in order to determine whether arrival patterns are consistent with the expectations of the system designer (in which case the behavior is a LO-criticality one) or not (in which case the behavior is HI-criticality or erroneous)? In order to make such a determination, the implementation platform must support the ability to *monitor* the arrivals of jobs of individual tasks. Some platforms may provide an additional ability to police job arrivals in order to *enforce* minimum inter-arrival separations. 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 [6] for the scheduling of MC task systems in which the CA's pessimism is expressed by having larger WCET values. In this paper we adapt these same three FP scheduling schemes —***Partitioned Criticality*** (a standard scheme, sometimes called *criticality monotonic* (CM) priority assignment, that is widely used in industrial practice); ***Static Mixed Criticality*** (SMC); and ***Adaptive Mixed Criticality*** (AMC)— for the scheduling of MC task systems in which pessimism is expressed in the period parameter.

In *Partitioned Criticality*, 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 – we will see that this

improves schedulability. One variant of the *static* scheme (SMC) polices job arrivals: it does not allow successive jobs of any task τ_i to arrive sooner than $T_i(\text{LO})$ time apart. The *adaptive* scheme (AMC) goes further and does not allow LO-criticality jobs to execute at all if successive jobs of any task τ_i arrive sooner than $T_i(\text{LO})$ time apart.

IV. Related work

Current industrial practice for ensuring non-interference on safety-critical components from non-critical ones is primarily based on strict *isolation* between components of different criticalities. For the model considered in this paper, such isolation would require that each task $\tau_i = (\chi_i, C_i, D_i, T_i(\text{LO}), T_i(\text{HI}))$ be modeled by the "regular" (i.e., not mixed-criticality) sporadic task $(C_i, D_i, T_i(\chi_i))$. That is, each LO-criticality task is assumed to arrive according to its LO-criticality period while each HI-criticality task is assumed to arrive according to its HI-criticality period. Although such isolation does indeed guarantee correctness, it is overly pessimistic in that some systems that could be scheduled correctly using other approaches are not schedulable using such an isolation-based approach.

To our knowledge, all prior research on mixed-criticality certification-cognizant scheduling (other than [3], which considered non-FP scheduling) has only looked at pessimism with regards to WCET — i.e., when the CA's WCET estimate is more pessimistic (larger) than the system designer's. Vestal [18] initiated this field of study: he proposed a fixed-priority (FP) algorithm for assigning priorities optimally to the tasks in such a system. Vestal's algorithm was further analyzed in [12]; generalizations were proposed in [6]. The non-FP scheduling of such sporadic task systems was considered in [15], [13], [4].

The non-FP scheduling of systems of sporadic mixed-criticality tasks with pessimism expressed along the frequency dimension was studied in [3]: a correct scheduling strategy, based on EDF, was presented and proved correct, and a utilization bound derived.

Several other real-time scheduling papers have dealt with mixed-criticality systems, although not from our perspective of scheduling for certification. They are therefore not directly relevant to the work we describe here.

V. 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 assigned according to the *deadline monotonic (DM)* priority assignment scheme [14], which is known to be optimal for constrained-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 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	D_i	$T_i(\text{LO})$	$T_i(\text{HI})$
τ_1	LO	1	10	10	10
τ_2	HI	10	200	250	200

CM priority-assignment would require that τ_2 be assigned higher priority, and τ_1's jobs could miss deadlines (consider, e.g., the scenario in which jobs of both tasks arrive simultaneously). It may be verified that assigning τ_1 higher priority yields a correct scheduling strategy.

The efficacy of CM scheduling may be shown to be arbitrarily pessimistic by increasing the periods and deadline parameters of τ_2 in this example.

■

VI. Static Mixed Criticality - SMC

We consider two variants here: one in which no run-time monitoring or policing is done, and a second in which admission control is performed to prevent successive jobs of any LO-criticality task τ_i (i.e., with $\chi_i = \text{LO}$) from arriving sooner than $T_i(\text{LO})$ apart[2]. In both variants, SMC assigns priorities to the tasks in the MC sporadic task system by applying the *Audsley optimal priority assignment* strategy [2]. Priority assignment according to this strategy proceeds by identifying some task that would meet its timing constraints if it were assigned lowest priority; once this task is identified, it is assigned the lowest priority and removed from consideration and the process repeated on the remaining tasks. (That is, some remaining task is identified that would meet its timing constraints if it were assigned lowest priority amongst the remaining tasks –second-lowest priority in the overall system, and so on.)

To specify the two priority assignment strategies, it remains to specify how we determine whether a particular task will meet its timing constraints if it were assigned lowest priority; this is done below in Section VI-A for the variant with no run-time monitoring or policing, and in Section VI-B for the variant that requires admission control.

A. SMC with no admission control

SMC with no admission control requires no additional run-time support (over and above that expected for supporting FP scheduling of "regular –non-MC– task systems) for handling the mixed-criticality nature of the workload. That is, the system design is subject to analysis prior to implementation (using the technique we discuss below); once it passes analysis, run-time scheduling is performed using a regular FP scheduler.

We specify how it is determined whether a particular job may be assigned lowest priority. Suppose we are seeking to

[2]The rationale for the use of the adjective "static" in naming these priority assignment schemes will become clear when we discuss an *adaptive* priority assignment scheme in Section VII.

determine whether τ_i, with criticality level $\chi_i \in \{\text{LO}, \text{HI}\}$, can be the lowest-priority task. Mixed-criticality semantics specify that jobs of τ_i should meet their deadlines, provided arrival patterns of all the tasks in the system respect their χ_i period parameters (i.e., jobs of each task τ_j arrive with a minimum inter-arrival separation of $T_j(\chi_i)$). It then follows from response-time analysis (RTA) [19] that the maximum response time of τ_i's jobs is given by the smallest fixed-point solution of the following recurrence:

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

This recurrence can be solved using standard techniques from RTA; if the solution is no larger than D_i then τ_i may indeed be assigned lowest priority.

B. SMC with admission control

With this scheme *admission control* is done during run-time in order to ensure that successive jobs of any task τ_i arrive no sooner than $T_i(\chi_i)$ time apart. (How earlier arrivals are dealt with depends on application semantics – jobs that arrive sooner than permitted may either be discarded, or queued and admitted later, at the permitted rate.) In particular, this means that jobs of a task τ_j with $\chi_j = \text{LO}$ are only admitted with a minimum inter-arrival separation of $T_j(\text{LO})$. Once again applying RTA, we conclude that the maximum response time of τ_i's jobs is given by the smallest fixed-point solution of the following recurrence:

$$t = \sum_{\forall\ j} \left\lceil \frac{t}{T_j(\min(\chi_i, \chi_j))} \right\rceil C_j \tag{2}$$

where the $\min$ operator when applied to criticalities has the interpretation that

$$\min(\chi_1, \chi_2) = \begin{cases} \text{LO}, & \text{if } \chi_1 = \text{LO} \ \vee \ \chi_2 = \text{LO} \\ \text{HI}, & \text{otherwise} \end{cases}$$

This recurrence, too, can be solved using standard techniques from RTA; τ_i may only be assigned lowest priority if the solution is no larger than D_i.

Example 2: This task system $\{\tau_1, \tau_2\}$ illustrates that SMC with admission control may be able to schedule task systems that cannot be scheduled in the absence of admission control.

τ_i	χ_i	C_i	D_i	$T_i(\text{LO})$	$T_i(\text{HI})$
τ_1	LO	5	5	15	10
τ_2	HI	10	15	15	15

Since $C_1 = D_1$, it is evident that τ_1 cannot be guaranteed to meet its deadline in any behavior (including LO-criticality ones) unless it is assigned highest priority. Hence τ_2 must be assigned lowest priority.

- Under SMC with no admission control, the Recurrence (1) for τ_2 is

$$\begin{aligned} t &= \lceil t/T_1(\chi_2) \rceil\, C_1 + \lceil t/T_2(\chi_2) \rceil\, C_2 \\ &= \lceil t/T_1(\text{HI}) \rceil\, C_1 + \lceil t/T_2(\text{HI}) \rceil\, C_2 \\ &= \lceil t/10 \rceil \times 5 + \lceil t/15 \rceil \times 10 \end{aligned}$$

  for which there is no non-negative solution $\leq D_2$. Hence the system is not FP-schedulable in the absence of admission control.
- If run-time support for admission control is available, then successive jobs of τ_1 can be prevented from arriving sooner than $T_1(\text{LO}) = 15$ time units apart. Under SMC in the presence of admission control, the Recurrence (2) for τ_2 is therefore

$$\begin{aligned} t &= \left\lceil \frac{t}{T_1(\min(\chi_1, \chi_2))} \right\rceil C_1 + \left\lceil \frac{t}{T_2(\min(\chi_2, \chi_2))} \right\rceil C_2 \\ &= \lceil t/T_1(\text{LO}) \rceil\, C_1 + \lceil t/T_2(\text{HI}) \rceil\, C_2 \\ &= \lceil t/15 \rceil \times 5 + \lceil t/15 \rceil \times 10 \end{aligned}$$

  for which the smallest solution is 15. Since $D_2 = 15$, the system is thus deemed FP-schedulable if admission control can be performed.

■

VII. Adaptive Mixed Criticality - AMC

We now discuss an adaptive dispatching scheme. This scheme is *adaptive* in the sense that it monitors the arrival of the jobs of each task during runtime; if successive jobs of any task arrive sooner than the task's LO-criticality period apart, it immediately *drops* all LO-criticality jobs and only executes HI-criticality ones. The following example illustrates that the ability of AMC to drop LO-criticality jobs enables it to schedule task systems that cannot be scheduled by SMC algorithms.

Example 3: Consider a task system comprised of three tasks $\{\tau_1, \tau_2, \tau_3\}$, with attributes as follows:

τ_i	χ_i	C_i	D_i	$T_i(\text{LO})$	$T_i(\text{HI})$
τ_1	LO	1	2	2	2
τ_2	HI	1	2	10	2
τ_3	HI	4	100	100	100

Since C_3 is larger than both D_1 and D_2, it is evident that τ_1 and τ_2 must be assigned greater priority than τ_3 in any fixed-priority scheduling scheme that schedules LO-criticality behaviors correctly.

It may be verified that assigning τ_1 and τ_2 priorities in either order, and both being assigned greater priority than τ_3, will result in τ_1 and τ_2 meeting all their deadlines in all LO-criticality and HI-criticality behaviors. It remains to determine whether τ_3 would meet its deadlines under either SMC scheme.

Under SMC with no admission control, the maximum response time of τ_3's jobs is given by the smallest fixed-point solution to Recurrence (1):

$$\begin{aligned} t &= \lceil t/T_1(\chi_3) \rceil\, C_1 + \lceil t/T_2(\chi_3) \rceil\, C_2 + \lceil t/T_3(\chi_3) \rceil\, C_3 \\ &= \lceil t/T_1(\text{HI}) \rceil\, C_1 + \lceil t/T_2(\text{HI}) \rceil\, C_2 + \lceil t/T_2(\text{HI}) \rceil\, C_3 \\ &= \lceil t/2 \rceil \times 1 + \lceil t/2 \rceil \times 1 + \lceil t/100 \rceil \times 4 \end{aligned}$$

for which there is no non-negative solution ≤ 100. Hence the system is not FP-schedulable in the absence of admission control.

Under SMC with admission control, the maximum response time of τ_3's jobs is given by the smallest fixed-point solution to Recurrence (2)

$$\begin{aligned} t &= \lceil t/T_1(\min\{\chi_1, \chi_3\}) \rceil\, C_1 + \lceil t/T_2(\min\{\chi_2, \chi_3\}) \rceil\, C_2 \\ &\quad + \lceil t/T_3(\min\{\chi_3, \chi_3\}) \rceil\, C_3 \\ &= \lceil t/T_1(\text{LO}) \rceil\, C_1 + \lceil t/T_2(\text{HI}) \rceil\, C_2 + \lceil t/T_3(\text{HI}) \rceil\, C_3 \\ &= \lceil t/2 \rceil \times 1 + \lceil t/2 \rceil \times 1 + \lceil t/100 \rceil \times 4 \end{aligned}$$

which is the same recurrence as the one above, and therefore also has no non-negative solution ≤ 100. Hence this system is not FP-schedulable under SMC even if admission control is enabled.

Under our adaptive scheme, however, the run-time dispatcher immediately discards the LO-criticality task τ_1 if τ_2's jobs arrive sooner than 10 time units apart. We will see later that as a consequence it can guarantee to complete τ_3's job well within its specified deadline of 100 time units.

■

In Section VII-A, we describe the algorithm used for determining which job should execute at each instant during run-time, assuming that priorities have been assigned to the tasks. The algorithm for assingining priorities is described in Section VII-B.

A. 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 arrives sooner than expected (i.e., a job of some task τ_i arrives prior to $T_i(\text{LO})$ time having elapsed since the arrival of $\tau_i's$ previous job), then $\Gamma \leftarrow$ HI.
4) Once $(\Gamma \equiv \text{HI})$, jobs with criticality level $\equiv$ LO will *not* execute. 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 $\Gamma \leftarrow$ LO any further in this paper.)

B. AMC priority assignment

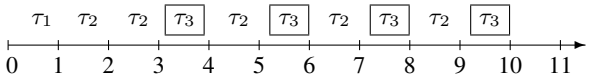
We now describe our AMC priority-assignment scheme for assigning the priorities that are used by the adaptive dispatcher described in Section VII-A above. Priorities assignment is again done according to the Audsley Optimal Priority Assignment strategy [2]. That is, some task is identified that may be 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. At a high level, the approach is similar to the one used in both variants of SMC (Sections VI-A and VI-B): we determine the worst-case response time of a task τ_i if it were assigned lowest priority, and check whether this is $\leq$ the relative deadline of the task. However, there is a new problem with realizing this approach for AMC scheduling: unlike in the SMC variants, for which it was known that the response time of the lowest-priority task is maximized when all other tasks arrive in the synchronous arrival sequence[3], for AMC *we do not know how to determine the maximum response time of the lowest-priority task*. The following example illustrates that the synchronous arrival sequence does not result in the maximum response time:

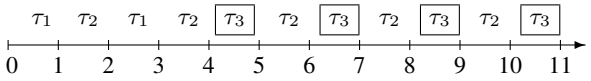
Example 4: Consider once again the task system of Example 3, and suppose that the tasks are prioritized as $\tau_1 > \tau_2 > \tau_3$. We seek to determine the worst-case response time of task τ_3's job.

Task τ_3's worst-case response time in any LO-criticality behavior is easily seen to be equal to 10. Let us now compute its worst-case response time in HI-criticality behaviors.

- In a HI-criticality behavior, successive jobs of τ_2 may arrive as few as 2 time units apart. Suppose then that all three tasks had jobs arrive simultaneously at time-instant zero, and τ_2 had its second job arrive at time-instant two. The arrival of τ_2''s job at time-instant two would result in all current and future jobs of τ_1 being discarded; as a consequence, the schedule would look like this:



  The response time of τ_3 is therefore once again 10.
- Suppose instead that τ_2 had its second job arrive at time-instant three (rather than its earliest time of two). The second job of τ_1 was scheduled before the dispatcher had determined that the behavior is a HI-criticality one and jobs of τ_1 may be discarded; as a consequence, the schedule would now look like this:



  The response time of τ_3 in this scenario is seen to equal 11, which is larger than the value of 10 identified in the scenario above in which all jobs arrive as soon as legally entitled to do so.

■

We do not yet know how to determine the exact worst-case response time of tasks scheduled under the adaptive dispatching scheme discussed in Section VII-A above. In the remainder of this section, we describe how we can compute an *upper bound* on this response time. We will then use these upper bounds to determine safe priority assignments.

We introduce some notation: 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 lowest priority: since no LO-criticality behavior can span the entire interval between the

[3]The *synchronous arrival sequence* for a set of tasks occurs when each task has a job arrive at the same instant, and subsequent jobs all arrive as soon as legally permitted to do so.

- Determine L_{LO} as the smallest positive value of t satisfying Equation 3. If there is some LO-criticality task τ_i with $D_i \geq L_{\text{LO}}$, assign it lowest priority.
- **Else** determine L_{HI} as the smallest positive value of t satisfying Equation 4. If there is some HI-criticality task τ_i with $D_i \geq L_{\text{HI}}$, assign it lowest priority.
- **Else** declare failure.

Fig. 1. AMC: Determining the lowest-priority task.

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 $D_i \geq L_{\text{HI}}$ may be assigned lowest priority.

Based on results from Response-Time Analysis (RTA) [19] it is straightforward to observe that L_{LO} can be set equal to the smallest positive value of t satisfying

$$t = \sum_{\forall j} \left\lceil \frac{t}{T_j(\text{LO})} \right\rceil C_j \tag{3}$$

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 arrives, despite strictly less than $T_i(\text{LO})$ time having elapsed since the arrival of the *previous* job –if any– of τ_i.) 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 $\lceil t_1/T_j(\text{LO}) \rceil$ 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 $\sum_{j:\chi_j=\text{LO}} \left(\lceil L_{\text{LO}}/T_j(\text{LO}) \rceil \cdot C_j \right)$. 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 $\sum_{j:\chi_j=\text{HI}} \left(\lceil t/T_j(\text{HI}) \rceil \cdot C_j \right)$. 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

$$t = \sum_{j:\chi_j=\text{LO}} \left\lceil \frac{L_{\text{LO}}}{T_j(\text{LO})} \right\rceil C_j + \sum_{j:\chi_j=\text{HI}} \left\lceil \frac{t}{T_j(\text{HI})} \right\rceil C_j \tag{4}$$

Plugging the value for L_{LO} obtained by solving Equation 3 into recurrence Equation 4, 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 1.

Example 5: Consider again the task system of Example 3. We seek to determine an upper bound on τ_3's response time.

L_{LO} is determined by finding the smallest positive solution to Recurrence 3:

$$\begin{aligned} t &= \lceil t/T_1(\text{LO}) \rceil C_1 + \lceil t/T_2(\text{LO}) \rceil C_2 + \lceil t/T_3(\text{LO}) \rceil C_3 \\ &= \lceil t/2 \rceil + \lceil t/10 \rceil \times 1 + \lceil t/100 \rceil \times 4 \end{aligned}$$

for which the smallest positive solution is $t = 10$. We therefore have $\mathbf{L_{\text{LO}} = 10}$ for our example system.

To compute L_{HI}, we must obtain the smallest positive solution to Recurrence 4:

$$\begin{aligned} t &= \lceil L_{\text{LO}}/T_1(\text{LO}) \rceil C_1 + \lceil t/T_2(\text{HI}) \rceil C_2 + \lceil t/T_3(\text{HI}) \rceil C_3 \\ &= \lceil 10/2 \rceil + \lceil t/2 \rceil + \lceil t/100 \rceil \times 4 \\ &= 5 + \lceil t/2 \rceil + 4 \end{aligned}$$

for which the smallest positive solution is $t = 18$. We therefore have $\mathbf{L_{\text{HI}} = 18}$ for our example system. This is the desired upper bound on the worst-case response time of τ_3; since it is $\leq D_3$, we conclude that τ_3 may indeed be assigned lowest priority.

■

VIII. Run-time complexity

Both variants of the SMC priority assignment scheme in Section VI, and the AMC priority assignment scheme of Section VII, make use of the Audsley optimal priority assignment strategy [2]. The word *optimal* needs to be qualified in this context: as stated above, the algorithm in Section VII is *not* an optimal AMC priority assignment scheme. Instead, the optimality holds in the following narrower sense: observe that the algorithm of Figure 1 is non-deterministic in the sense that it leaves unspecified which τ_i to choose if there are multiple candidates satisfying the desired constraint. This algorithm is *optimal* in the sense that ties may be broken arbitrarily: if some choice of tie-breaks at each step would have resulted in a successful priority assignment, then any choice of tie-breaks will result in a successful priority assignment. Similarly for both variants of SMC: in choosing some task with deadline parameter $\geq$ the solution to Recurrence (1) for SMC with no admission control, or Recurrence (2) for SMC with admission control, if some choice of tie-breaks at each step would have resulted in a successful priority assignment, then any choice of tie-breaks will result in a successful priority assignment.

Priority assignment according to the Audsley optimal priority assignment strategy [2] requires that some task be identified that may be assigned lowest priority; once this task is identified, it is assigned the lowest priority and removed from consideration, after which some remaining task is identified that may be assigned second-lowest priority, and so on. For an n-task system there are n tasks that may potentially need to be considered as being eligible to be assigned lowest priority; $(n-1)$ tasks that may potentially need to be considered as being eligible to be assigned second-lowest priority; etc. Hence in the worst case $n + (n-1) + (n-2) + \cdots + 2 + 1 = \Theta(n^2)$ recurrences of the form (1)-(4) may need to be solved. The following lemma tells us that in fact far fewer recurrences need to be solved for constrained-deadline mixed criticality sporadic task systems.

Lemma 1: Suppose that the priority-assignment algorithms described in Section VI-A (SMC with no admission control), Section VI-B (SMC with admission control), or Section VII (AMC) is successful in determining a correct priority ordering. Then a correct priority ordering exists that has all tasks with the same criticality assigned priorities in deadline monotonic priority order.

Proof: The proof follows the standard method of proving that deadline monotonic priority ordering is optimal for tasks with constrained deadlines (as described in any standard textbook such as [11]). Assume τ_i is deemed schedulable at the lowest priority (i.e., its computed response time bound R_i is $\leq D_i$). Let τ_k be any task with the same criticality level as τ_i but a larger deadline: $D_k > D_i$. If τ_i is exchanged with τ_k (so it becomes the lowest priority task) then τ_k will suffer exactly the same interference from LO and HI tasks and hence the busy period for τ_k will be the same as that for τ_i when it was assigned the lowest priority (both will contain a single C_i and a single C_k term as $D_i \leq T_i(\text{HI}) \leq T_i(\text{LO})$ for all tasks). Hence the computed response time bound R_k of τ_k is $= R_i \leq D_i \leq D_k$ (where R_i is the value computed when τ_i was assigned the lowest priority). It follows that τ_k is schedulable at the lowest priority and that the task with the longest deadline (but identical criticality) is also eligible to be assigned lowest priority. ■

Hence in seeking to determine whether some task may be assigned lowest priority, there are only two potential candidates to consider: the LO-criticality task with the largest relative deadline and the HI-criticality task with the largest relative deadline. This implies that a total of at most $\mathcal{O}(|\tau|)$ recurrences need to be solved in order to assign priorities to the tasks in the MC sporadic task system τ (where $|\tau|$ denotes the number of tasks in τ).

IX. Comparing the AMC and SMC algorithms

We first compare the two SMC schemes; we show that if the platform upon which the task system is being implemented allows for admission control, it is always beneficial (from the schedulability perspective) to exploit this feature and implement the SMC version that incorporates admission control.

Lemma 2: If SMC with no admission control successfully assigns priorities to the tasks in a task system τ, then so does SMC with admission control.

Proof Sketch: Suppose that SMC with no admission control determines that some task $\tau_i \in \tau$ may be assigned lowest priority. This implies that the smallest positive solution of Equation 1 is $\leq D_i$. Since the denominator of each term in the summation on the RHS of Equation 2 is at least as large as the denominator of the corresponding term in the summation on the RHS of Equation 1, it is evident that the smallest positive solution of Equation 2 is no larger than the smallest positive solution of Equation 1, and is hence also $\leq D_i$. ■

Next, we show below (Thm. 1) that AMC strictly *dominates* the SMC algorithm: by showing that (i) every task system that can be scheduled in a certifiably correct manner by SMC can also be scheduled in a certifiably correct manner by AMC (Lemma 3); and (ii) there are task systems schedulable by AMC, that SMC cannot schedule in a certifiably correct manner. Given the result of Lemma 2 above, it suffices to demonstrate this for SMC with admission control alone.

Lemma 3: If SMC with admission control successfully assigns priorities to the tasks in task system τ, then so does AMC.

Proof Sketch: Suppose that SMC with admission control determines that some task $\tau_i \in \tau$ may be assigned lowest priority. This implies that the smallest positive solution of Equation 2 is $\leq D_i$. We will show that this task may also be assigned lowest priority under AMC. We consider separately the cases when this lowest-priority task τ_i is a LO-criticality task ($\chi_i = \text{LO}$) and when it is a HI-criticality task ($\chi_i = \text{HI}$).

- In the case when $\chi_i = \text{LO}$, notice that Equation 3 is identical to Equation 2. Recall that L_{LO} is, by definition, equal to the smallest positive solution to Equation 3. Hence, the fact that the smallest positive solution of Equation 2 is $\leq D_i$ implies that L_{LO} is also $\leq D_i$. Since τ_i is a LO-criticality task, it is consequently possible under AMC to assign it the lowest priority.
- It remains to consider the case when $\chi_i = \text{HI}$. Since we assumed that τ_i has been assigned lowest priority by the technique of Section VI, it must be the case that the smallest positive solution to Equation 2 is $\leq D_i$. Equation 2 can be rewritten as

$$\begin{aligned} t &= \sum_{\forall j} \left\lceil \frac{t}{T_j(\min(\text{HI}, \chi_j))} \right\rceil C_j \\ &= \sum_{j:\chi_j=\text{LO}} \left\lceil \frac{t}{T_j(\text{LO})} \right\rceil C_j + \sum_{j:\chi_j=\text{HI}} \left\lceil \frac{t}{T_j(\text{HI})} \right\rceil C_j \\ &\geq \sum_{j:\chi_j=\text{LO}} \left\lceil \frac{L_{\text{LO}}}{T_j(\text{LO})} \right\rceil C_j + \sum_{j:\chi_j=\text{HI}} \left\lceil \frac{t}{T_j(\text{HI})} \right\rceil C_j \end{aligned} \quad (5)$$

  Upon comparing Inequality 5 above to Equation 4, it becomes evident that the smallest positive solution to Equation 5 (and hence, to Equation 2) is no smaller than the smallest positive solution to Equation 4. Recall that L_{HI} is defined to be equal to the smallest positive solution to Equation 4. We may therefore conclude that $L_{\text{HI}} \leq D_i$, which in turn implies that AMC may assign lowest priority to τ_i.

We thus see that for any task system for which SMC identifies a lowest-priority job, AMC does likewise. The lemma proves, by repeated applications of this argument. ■

Examples 3 and 5 bear witness to the fact that there are MC sporadic task systems to which SMC cannot assign priorities in a certifiably correct manner, whereas AMC can. It follows from Lemma 3 that every task system that can be scheduled in a certifiably correct manner by SMC can also be scheduled in a certifiably correct manner by AMC, allowing us to conclude that at least from the perspective of certifiable correctness,

Theorem 1: AMC dominates the priority assignment technique of SMC. ■

X. Experimental Evaluation

In this section we present the results we obtained from our experimental analysis. We randomly generated task sets and determined the percentage of task sets that are schedulable

under different priority assignment techniques that we considered. In some cases we computed the weighted schedulability [8].

The method we used to generate the task sets is inspired by, and hence very similar to, that used in [6].

- The UUnifast algorithm [9] was used to generate the task utilizations ($U_i = C_i/T_i$).
- The low criticality period, $T_i(\text{LO})$, for each task was generated according to a log-uniform distribution in the range 10ms to 1000ms. The high criticality period, $T_i(\text{HI})$, was a fixed multiplier of the low criticality period, $T_i(\text{HI}) = CF \times T_i(\text{LO})$ and $CF \leq 1$. All task periods were set to integer values, by rounding down from any non-integer value.
- Task deadlines, D_i, were set equal to $T_i(\text{HI})$.
- The worst-case execution times, C_i, were set equal to $U_i \times T_i(\text{LO})$.
- The probability that a generated task was a high criticality task was given by the parameter CP. For example, if $CP = 0.5$ then on an average 50% of the generated tasks were expected to have high criticality.

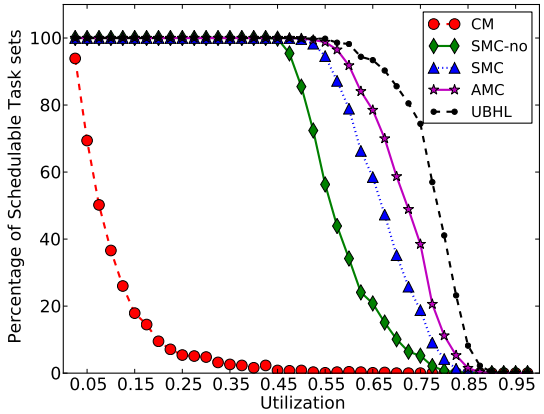
Our task set parameter generation method is different from the one used in [6] only in the following way: In [6] each task had a HI-criticality worst case execution time which was CF ($CF \geq 1$) times the LO-criticality worst-case execution time. However, in our case, each task has a HI-criticality time period, $T_i(\text{HI})$ which is CF ($CF \leq 1$) times the LO-criticality time period, $T_i(\text{LO})$.

The different priority assignment techniques that we considered are:

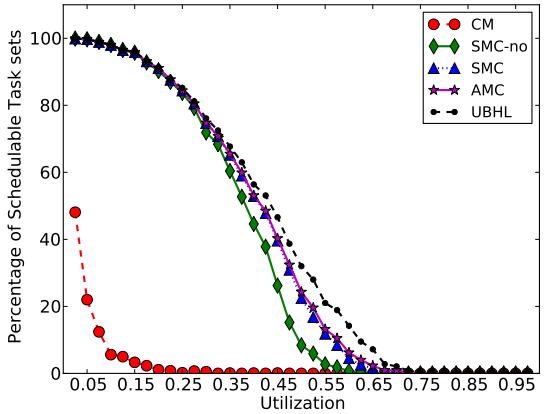
- CM: Criticality monotonic priority ordering described in Section V.
- SMC-no: SMC with no admission control described in Section VI-A.
- SMC: SMC with admission control described in Section VI-B.
- AMC: The AMC priority assignment scheme described in Section VII.
- UBHL: UBHL, described in [6], gives the upper bound on the schedulability of a task set: if a task set is not schedulable under UBHL then it is not schedulable according to any priority assignment (UBHL is a mnemonic for "upper bound – HI and LO"). To pass this test the task set must be shown to be schedulable in both Steps 1 and 2 described below:
  - Step 1 - All tasks in the task set are arranged as per deadline monotonic priority ordering and response time analysis is done assuming that all tasks are executing with low criticality time period.
  - Step 2 - Only the high criticality tasks are arranged as per deadline monotonic priority ordering and response time analysis is done assuming that only high criticality tasks are executing with high criticality time period.

For the experiments reported in Figure 2 task sets were generated with utilizations in the range 0.025 to 0.975 in increments of 0.025. For each utilization value, 1000 task sets were generated such that each task set had 20 tasks ($N = 20$), and on average 50% of these tasks had HI criticality ($CP = 50\%$). For all tasks the HI-criticality time period was 0.5 times its LO-criticality time period ($CF = 0.5$), and the deadline was set equal to its HI- criticality time period, $D_i = T_i(\text{HI})$. For each of the task sets priorities were assigned according to the different priority assignment schemes. The graph in Figure 2(a) shows that percentage of schedulable task sets under the respective priority assignment schemes.

In Figure 2(b) we repeated the experiment with the parameters described above and set the deadline of each task, D_i, to a random value chosen uniformly between C_i and $T_i(\text{HI})$.



(a) Deadline equal to Period, $T_i(\text{HI})$

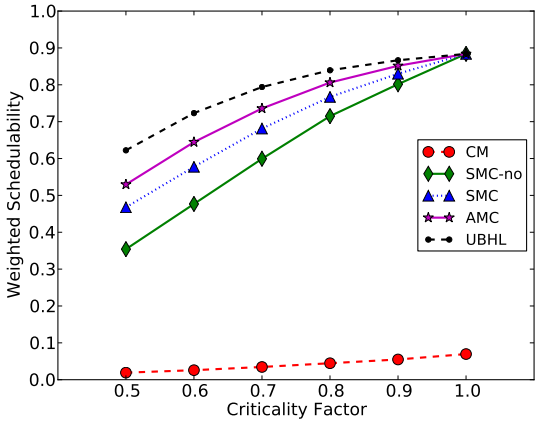


(b) Deadline less than Period, $T_i(\text{HI})$

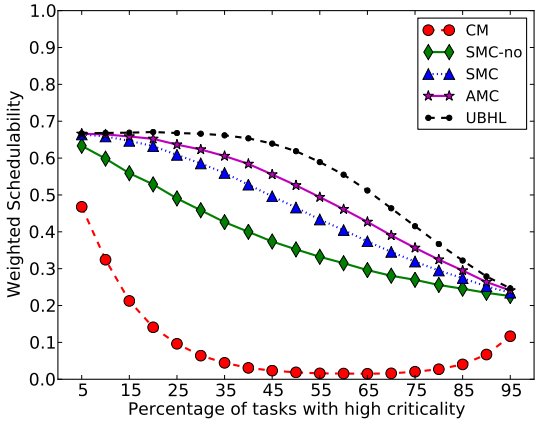
Fig. 2. Percentage of Schedulable Task sets

From the graphs in Figure 2 we see that the AMC priority assignment scheme outperforms CM, SMC-no, and SMC. Also, as expected, the curves that correspond to the different priority assignment schemes are below the curve that corresponds to the UBHL test.

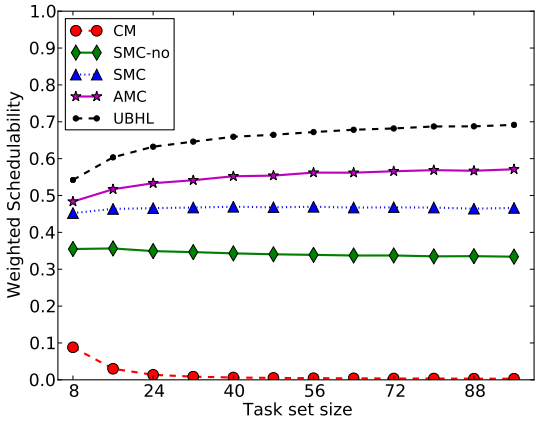
In Figure 3 we show our results in terms of weighted schedulability. Similar weighted schedulability tests are described in [6]. The weighted schedulability measure $W_y(p)$ [8]



(a) Varying Criticality Factor



(b) Varying Criticality Mix



(c) Varying Task set size

Fig. 3. Weighted Schedulability

for schedulability test y, as a function of parameter p, combines results of all task sets τ, generated for a set of equally spaced utilization values. Let $S_y(\tau, p)$ be the binary result (1 or 0) of schedulability test y for a task set τ with parameter value p:

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

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

In our experiments we computed the weighted schedulability for each of the priority assignment schemes, as a function of the key parameters: criticality factor, CF, percentage of tasks with high criticality, CP, and size of the task set, N. We varied one of the key parameters at a time. The other parameters were set to the values assigned to them in the experiments for Figure 2. For each value of the varying parameter, we computed the weighted schedulability by generating task sets with utilization in the range 0.025 to 0.975 in increments of 0.025. For each of the utilization values 1000 task sets were generated.

The weighted schedulability measure reduces what would otherwise be a 3-dimensional plot to 2 dimensions [8]. A higher weighted schedulability reflects that task sets with higher utilization are schedulable.

Figures 3(a)- 3(c) show the results of varying each of the key parameters. Figure 3(a) varies the criticality factor, Figure 3(b) varies the percentage of tasks with HI criticality, and Figure 3(c) varies the size of the task set. The following observations can be made:

- In all the graphs we see that the weighted schedulability of AMC is greater than or equal to the weighted schedulability of CM, SMC-no, and SMC.
- In Figure 3(a), as the criticality factor, CF, increases the value of the HI-criticality period, $T_i(\text{HI})$, for each task increases, which also implies that the task deadlines, D_i, are greater. Thus, as CF increases we see an increase in the weighted schedulability of all the priority assignment schemes. Also, when $CF = 1$, $T_i(\text{LO}) = T_i(\text{HI})$. In this case it can easily be shown that the schedulability under SMC-no, SMC, and AMC will be the same and this can also be observed in Figure 3(a).
- In Figure 3(b), the curve for CM is U-shaped because each end of the interval represents a task set with more tasks of the same criticality. When there are more tasks of the same criticality, the priorities assigned by CM are closer to optimal.
- In Figure 3(c), varying the task set size does not significantly vary the weighted schedulability of the different priority assignments schemes.

The overall observation that we make from our experimental results and theoretical analysis is that AMC is an effective priority assignment scheme for scheduling fixed-priority, mixed criticality systems.

XI. 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. Much of this research has focused on dealing with pessimism regarding WCET parameters. In this document, we have described some ongoing work on scheduling mixed-criticality systems when certification pessimism is expressed in terms of increased frequency of occurrence of events needing certification. The algorithms described above are easily generalized in several directions; we briefly list some of these extensions below.

More than two criticality levels. We have thus far 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.

Multiple specifications along both dimensions. To keep the discussion simple, we have chosen to suppose that while the period parameters are specified more pessimistically at higher criticality levels, the WCET parameters remain the same across all levels. Much previous work has discussed certification-cognizant scheduling of systems in which the period remains the same but the WCET parameters get more pessimistic with increasing criticality; to our knowledge, this work is the first to consider increased pessimism for periods. It is relatively straightforward to combine both forms of pessimism into a single framework, and come up with scheduling strategies that are able to deal with added pessimism in both WCET and period at higher criticality levels.

ACKNOWLEDGEMENTS

We are very grateful to Alan Burns and Rob Davis for help with the simulation experiments reported here, and for useful discussions. This research builds upon the work reported in a paper [6] co-authored by the first author with Alan and Rob.

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; and AFRL grant FA8750-11-1-0033.

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] S. Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *Proceedings of the IEEE Symposium on Industrial Embedded Systems (SIES)*. IEEE Press, 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 *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 555–566, Saarbrucken, Germany, September 2011. Springer-Verlag.

[5] S. Baruah and A. Burns. Implementing mixed criticality systems in Ada. In A. Romanovsky and T. Vardanega, editors, *Proceedings of Reliable Software Technology - Ada Europe 2011*, volume LNCS 6652, pages 174–188. Springer, 2011.

[6] 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.

[7] S. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC, 2003.

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

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

[10] A. Burns and S. Baruah. Timing faults and mixed criticality systems. In Jones and Lloyd, editors, *Dependable and Historic Computing*, volume LNCS 6875, pages 147–166. Springer, 2011.

[11] A. Burns and A. Wellings. *Real-Time Systems and Programming Languages*. Addison-Wesley, fourth edition edition, 2009.

[12] F. Dorin, P. Richard, M. Richard, and J. Goossens. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems*, 2010.

[13] 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.

[14] 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.

[15] H. Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proceedings of the Real-Time Systems Symposium*, pages 183–192, San Diego, CA, 2010. IEEE Computer Society Press.

[16] 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.

[17] 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.

[18] 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.

[19] 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.