Mixed-criticality scheduling: improved resource-augmentation results

Sanjoy Baruah[*] Haohan Li Leen Stougie[†]
The University of North Carolina Vrije Universiteit and CWI

Abstract

Many safety-critical embedded systems are subject to certification requirements; some systems may be required to meet multiple sets of certification requirements, from different certification authorities. Certification requirements in such "mixed-criticality" systems give rise to some interesting scheduling problems, that cannot be satisfactorily addressed using techniques from conventional scheduling theory. It had previously been shown that determining whether a system specified in this model can be scheduled to meet all its certification requirements is highly intractable. Prior work [4] had also introduced a simple, priority-based scheduling algorithm called OCBP for mixed criticality systems, and had quantified, via the metric of processor speedup factor, the effectiveness of OCBP in scheduling dual-criticality systems – systems subject to two sets of certification requirements.

In this paper, we extend this result to systems with arbitrarily many distinct criticality levels, by deriving a quantitative processor speedup factor (that depends on n) for OCBP when scheduling systems with n criticality levels for arbitrary n.

1 Introduction

There is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. The concept of *mixed criticalities* is fast coming to be regarded as an important concept in such systems, and has in fact been identified as one of the core foundational concepts in the emerging discipline of *Cyber Physical Systems*. In such systems, mixed criticalities can mean two different things. The first meaning is the obvious one: upon platforms that offer support for multiple functionalities, it is some of these functionalities are probably more important (more "critical") to the overall welfare of the platform than others. However, there is another aspect [2] to mixed criticalities that arises in application domains (such as civilian and defense avionics) that are subject to mandatory certification requirements by statutory organizations. It is *the aspect of mixed criticalities arising as a consequence of such certification requirements, that is the focus of this paper*.

Mixed-criticality and certification. We illustrate the certification aspect of mixed criticality via an example taken from the domain of unmanned aerial vehicles (UAV's) that are used for reconnaissance and surveillance. The functionalities on board such UAV's can be classified into two categories:

- The *flight-critical* functionalities, that must be performed by the aircraft in order to ensure its safe operation.
- The *mission-critical* functionalities, which are concerned with the reconnaissance and surveillance objectives.

In order that such UAV's be permitted to operate over civilian airspace (e.g., for border surveillance), it is mandatory that its flight-critical functionalities be certified by civilian Certification Authorities (CA's), such as the Federal Aviation Authority in the US. Such CA's tend to be very conservative: they require that the correctness of the UAV be demonstrated under extremely rigorous and pessimistic assumptions, which are very unlikely to occur in reality. However, the CA's are not concerned with the correctness (or otherwise) of the mission-critical functionalities — their sole concern is with the safety of the aircraft. The mission-critical functionalities must instead be validated by the clients and the vendor-manufacturer, typically to standards that are less rigorous than the ones used by the CA's.

We illustrate this difference in rigor by considering a particular characterizing parameter of real-time code: the *Worst-Case Execution Time* (WCET). The WCET of a piece of code represents an upper bound on the amount of time required to execute this code. Each CA specifies its own rules, tools, etc., for determining the value of the WCET:

- For flight-critical certification purposes, the CA's require that we have a great deal of confidence that the

[*]Supported in part by AT&T, IBM, and Sun Corps.; NSF grants CNS 0834270 and CNS 0834132; ARO grant W911NF-09-1-0535; and AFOSR grant FA9550-09-1-0549.
[†]Supported by the Dutch BSIK-BRICKS project.

value we assign to this parameter be an actual upper bound on the execution time of the code. Such confidence could be obtained by, e.g., severely restricting the kinds of programming constructs that may be used, analyzing the programs very carefully to identify the worst-case execution path, and subjecting this path to careful cycle-counting analysis under extreme pessimistic assumptions regarding cache state etc.

- For mission-critical validation, it may suffice to estimate the WCET by performing simulation experiments on the code, covering what we believe are all extremal behaviors of the system, measuring the runtimes to determine the largest value, and perhaps inflating the largest observed value by an additional "fudge" factor to give us greater confidence. The resulting estimate will still be very conservative, but presumably not as large as the value determined above for use in the flight-critical certification process.

Thus, the same piece of code will be characterized by *different* WCET's in safety-critical certification and for mission-critical certification, and it is incumbent that the platform pass both certification processes. This would not be an issue if *all* the functionality on board the platform needed to be certified by both CA's: in that case, we would simply take the more conservative bound (the larger WCET estimate) and use this in both certification processes. However (as stated above), it is typically the case that only some of the functionality must be certified according to the more rigorous flight-critical certification process, while the entire system (comprising the flight-critical plus the mission-critical functionalities) must pass the less rigorous mission-critical certification. We illustrate by an example.

Example 1 Consider a system comprised of two jobs: J_1 is flight-critical while J_2 is only mission-critical. Both jobs arrive at time-instant 0, and have their deadlines at time-instant 10. The WCET of J_1, estimated according to the techniques associated with flight-critical certification, is determined to equal 6, while the WCET of J_2, estimated using the techniques associated with mission-critical certification is 5. Using the WCET estimates of 6 and 5 respectively, there is no way that both jobs can be scheduled to guarantee completion by their deadlines. Recall, however, that

- For the purposes of flight-critical certification, it is irrelevant whether J_2 completes on time or not; and
- the value of 6 that is assigned to J_1's WCET parameter may be deemed too pessimistic for the purposes of mission-critical certification.

Let us suppose that the WCET of J_1, estimated using mission-critical certification techniques, is determined to be equal to 3 (rather than 6), and step through the certification processes if we were to schedule the jobs by assigning J_1 greater priority than J_2.

- The CA responsible for safety-critical certification would determine that J_1 completes by time-instant 6 and meets its deadline; hence the system passes certification.
- The CA responsible for mission-critical certification determines that J_1 completes by time-instant 3, and J_2 by time-instant 8. Thus they both complete by their deadlines, and the system passes certification.

We thus see that the system is certified as being correct by both the flight-critical and the mission-critical CA's, despite our initial observation that the sum of the relevant WCET's (6 and 5) exceeds the length of the scheduling window over which they are to execute. ■

This research. Example 1 above illustrates the central thesis of our ongoing research: *the efficient utilization of computing resources in mixed-criticality systems that are subject to multiple certification requirements requires the development of fundamental new scheduling theory.* Some progress has recently been made [2, 6, 7, 3, 4] towards such a theory, by (1) constructing formal models for accurately representing mixed-criticality systems; (ii) identifying the computational complexity of some basic and fundamental problems in mixed-criticality scheduling; (iii) proposing efficient (polynomial-time) approximation algorithms for solving some of the problems that are shown to be intractable; and (iv) making some initial steps towards quantifying the effectiveness of these polynomial-time approximation algorithms. In this paper, we continue to work towards this objective of obtaining approximation algorithms with quantifiable deviation from optimality for some of the scheduling problems that were shown to be intractable. We quantify the effectiveness of an inexact scheduling algorithm by its *processor speedup metric*: the minimum multiplicative factor by which processors must be made faster in order to compensate for the inexactness of the test. We extend prior results that quantified the performance of a particular polynomial-time mixed-criticality scheduling algorithm – *Own Criticality-Based Priority*, or OCBP – when it is charged with scheduling dual-criticality systems, by computing OCBP's processor speedup factor when it is scheduling systems with arbitrarily many criticality levels.

2 Model and definitions

In this section we formally define the mixed-criticality job model. These definitions are illustrated by means of examples in Section 2.1; while reading the following definitions, it may occasionally be useful to refer forward to Section 2.1.

Although our eventual interest is in the scheduling of collections of recurrent (periodic or sporadic) mixed-criticality tasks each of which can generate an infinite number of jobs[1], many fundamental questions remain unanswered regarding even the simpler case of finite collections of jobs. Hence we focus in this paper on the simpler case where a system is comprised of a finite number of (non-recurring) jobs. Our results may be considered as a first step towards a more comprehensive analysis of systems of mixed-criticality recurrent tasks. In addition, these results have immediate applicability for the scheduling of *frame-based* recurrent real-time systems, in which the recurrent nature of the behavior is expressed as the infinite repetition of a finite collection of jobs of the kind considered here.

A mixed-criticality (MC) *job* is characterized by a 4-tuple of parameters: $J_i = (A_i, D_i, \chi_i, C_i)$, where

- $A_i \in R^+$ is the release time.
- $D_i \in R^+$ is the deadline. We assume that $D_i \geq A_i$.
- $\chi_i \in N^+$ denotes the criticality of the job, with a larger value denoting higher criticality.
- $C_i : N^+ \rightarrow R^+$ specifies the worst case execution time (WCET) estimate of J_i for each criticality level. (It is reasonable to assume that $C_i(\ell)$ is monotonically non-decreasing with increasing ℓ.)

We will sometimes refer to the time interval $[A_i, D_i)$ as the *scheduling window* of job J_i.

MC instance. An MC instance is specified as a finite collection of such MC jobs: $I = \{J_1, J_2, \ldots, J_n\}$. Given such an instance, we are concerned here with determining how to schedule it to obtain correct behavior; in this document, we restrict our attention to scheduling on *preemptive uniprocessor* platforms.

Behaviors. The MC job model has the following semantics. Each job J_i is released at time-instant A_i, needs to execute for some amount of time γ_i, and has a deadline at time-instant D_i. The value of γ_i is not known from the specifications of J_i, but only becomes revealed by actually executing the job until it *signals* that it has completed execution. γ_i may take on very different values during different execution runs: we will refer to each collection of values $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ as a possible *behavior* of instance I.

The *criticality level* of the behavior $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ of I is the smallest integer ℓ such that $\gamma_i \leq C_i(\ell)$ for all $i, 1 \leq i \leq n$. (If there is no such ℓ, then we define that behavior to be *erroneous*.)

[1]Some partial results concerning the scheduling of such task systems may be found in [8, 5].

Scheduling strategies. A *scheduling strategy* for an instance I specifies, in a completely deterministic manner for all possible behaviors of I, which job (if any) to execute at each instant in time. An *on line* scheduling strategy does not have a priori knowledge of the behavior of I: for each $J_i \in I$, the value of γ_i only becomes known by executing J_i until it signals that it has completed execution. Since these actual execution times – the γ_i's – only become revealed during run-time, an on-line scheduling strategy does not a priori know what the criticality level of any particular behavior is going to be; at each instant, scheduling decisions are made based only on the partial information revealed thus far.

Correctness. A scheduling strategy is *correct* if it satisfies the following criterion for each $\ell \geq 1$: when scheduling any behavior of criticality level ℓ, it ensures that every job J_i with $\chi_i \geq \ell$ receives sufficient execution during the interval $[A_i, D_i)$ to signal that it has completed execution.

MC schedulability. Let us define an instance I to be MC schedulable if there exists a correct on-line scheduling strategy for it. The *MC schedulability problem* then is to determine whether a given MC instance is MC schedulable or not

2.1 An example

Consider an MC instance I comprised of 4 jobs at 2 criticality levels. We specify the WCET function of each task for the two criticality levels by explicit enumeration: $[C_i(1), C_i(2)]$.

- $J_1 = (0, 3, 2, [1, 2])$
- $J_2 = (0, 3, 1, [2, 2])$
- $J_3 = (0, 5, 2, [1, 1])$
- $J_4 = (3, 5, 2, [1, 2])$

For this instance, any behavior in which $\gamma_1, \gamma_2, \gamma_3$, and γ_4 are no larger than $1, 2, 1$, and 1 respectively has criticality 1; while any behavior not of criticality 1 in which $\gamma_1, \gamma_2, \gamma_3$, and γ_4 are no larger than $2, 2, 1$, and 2 respectively has criticality 2. All remaining behaviors are erroneous.

S0 below denotes a possible on-line scheduling strategy for this instance I:

S0: Execute J_1 over [0,1). If J_1 has remaining execution (i.e., γ_1 is greater than 1), then execute scheduling strategy S1 below; else,execute scheduling strategy S2 below.

S1: Execute J_1 over [1,2), J_3 over [2,3), and J_4 over [3,5).

S2: Execute J_2 over [1,3), J_3 over [3,4), and J_4 over [4,5).

Scheduling strategy S0 is not correct for I, as can be seen by considering the schedule that is generated on the behavior $(1, 2, 1, 2)$. This behavior has criticality 2 (since γ_4, at 2, is greater than $C_4(1) = 1$, it is not criticality 1); hence, a correct schedule would need to complete jobs J_1, J_3 and J_4 by their deadlines. However, the schedule generated by this scheduling strategy would have executed J_4 for only one unit by its deadline. In fact, it turns out that instance I is not MC schedulable.

3 OCBP scheduling

It has been shown that determining whether a given MC instance is MC schedulable or not is higly intractable:

Theorem 1 (From [3])
The MC schedulability problem — given an MC instance, determine whether it is MC-schedulable — is NP-hard in the strong sense. This is true even in the highly restricted case where all jobs have the same arrival times, and each job's criticality level is either 1 or 2.

This intractability implies that under the assumption that P $\neq$ NP, there can be no polynomial or pseudo-polynomial time algorithm for solving the MC schedulability problem (even in the restricted case of equal arrival times and only two criticality levels). Therefore, a sufficient schedulability test that can be implemented in polynomial time were proposed in [4]; we review this test here.

We assume that for each job J_i, $C_i(\ell) = C_i(\chi_i)$ for all $\ell \geq \chi_i$. That is, no job is allowed to execute for more than its WCET at its own specified criticality. The high-level description of our algorithm is as follows. Given an MC instance I, we aim to derive offline (i.e., prior to run-time) a total priority ordering of the jobs of I such that scheduling the jobs according to this priority ordering guarantees a correct schedule, where *scheduling according to priority* means that at each moment in time the highest-priority available job is executed.

The priority list is constructed recursively using the so-called "Audsley approach" [1]. We first determine the lowest priority job: Job J_i has lowest priority if there is at least $C_i(\chi_i)$ time between its release time and its deadline available if every other job J_j has higher priority and is executed for $C_j(\chi_i)$ time units (the WCET of job J_j according to the criticality level of job i). Then the procedure is repeated to the set of jobs excluding the lowest priority job, until all jobs are ordered, or at some iteration a lowest priority job does not exist.

Because of the priority of a job being based only on its own criticality level, we say the instance I is **Own Criticality Based Priority** (or *OCBP*)**-schedulable** if we find a complete ordering of the jobs. If at some recursion in the algorithm no lowest priority job exists, we say the instance is not OCBP-schedulable.

We now show that testing for OCBP-schedulability comprises a sufficient MC-schedulability test.

Lemma 1 *If MC instance I is OCBP-schedulable on a given processor, then I is MC schedulable on the same processor.*

Proof: Suppose that I is OCBP-schedulable, and let, after renaming of the jobs, J_1, J_2, …, J_n denote a priority ordering that bears witness to this.

Let J_k denote any job in this priority ordering. In order to show MC schedulability, it is incumbent to demonstrate that J_k can receive $C_k(\chi_k)$ units of execution in any behavior of I of criticality-level χ_k or lower. But in any behavior of criticality level χ_k or lower, each job J_i executes for no more than $C_i(\chi_k)$ units. And the OCBP-schedulability of I with priority-ordering $J_1 \rhd J_2 \rhd \cdots \rhd J_n$ implies that J_k will receive $C_k(\chi_k)$ units of execution if each $J_i \in \{J_1, \ldots, J_{k-1}\}$ executes for no more than $C_i(\chi_k)$ units; hence, J_k will indeed meet its deadline in all behaviors of criticality-level χ_k or lower. ■

We will now determine a processor speedup factor for OCBP when it is charged with scheduling systems with k-distinct criticality levels, for arbitrary $k \geq 1$.

First, a definition. Let us define the *maximum criticality* of MC instance I as the largest criticality of any job in I: maximum criticality of $I = \max_{J_i \in I}\{\chi_i\}$.

Let k denote any positive integer. Below, we will prove that *any MC instance I that is MC schedulable on a unit-speed processor and has maximum criticality $\leq k$ is OCBP-schedulable on a speed-s_k processor*, where s_k is defined according to the following recurrence:

$$\begin{aligned} s_1 &= 1 \\ s_k &= (1 + \sqrt{4\, s_{k-1}^2 + 1})/2, \quad k > 1 \end{aligned} \qquad (1)$$

Note that as per this definition, s_k is monotonically increasing with increasing k: $s_{k+1} > s_k$ for all $k \geq 1$. It can be shown that as $k \to \infty$, s_k approaches $k/2$.

We also note that for $k = 2$, s_k equals the golden ratio Φ; hence, the result derived here is indeed a true generalization of the previous result from [4].

Lemma 2 *Any MC-schedulable instance I of maximum criticality k is OCBP-schedulable on a speed-s_k*

processor, where s_k is as defined by the recurrence (1) above.

Proof: The proof is by induction on k. The **base case**, $k = 1$, is easily seen to hold: an MC instance with maximum criticality 1 is the same as a "regular" (non-MC) job instance, and the OCBP-schedulability test is merely Audsley's test [1].

As an **inductive hypothesis**, assume that all MC instances with maximum criticality $\leq (k-1)$ that are MC-schedulable on speed-one processors are OCBP-schedulable on speed-s_{k-1} processors. We will now demonstrate that any MC-instance with maximum criticality k that is MC-schedulable on a speed-one processor is OCBP-schedulable on a speed-s_k processors.

Let I denote a minimal instance of maximum criticality k that is MC-schedulable on a speed-1 processor, but not OCBP-schedulable on a speed-s processor for some $s \geq s_{k-1}$. Without loss of generality, let us assume that $\min_{J_i \in I} A_i = 0$ (i.e., the earliest release time is zero). Observe that it must be the case that there is no time-instant $t \in [0, \max_{J_i \in I} D_i)$ such that no job's scheduling window contains t. If there were such a t, it would follow that either the smaller instance comprised of only those jobs with scheduling windows before t, or the instance comprised of only those jobs with scheduling windows after t, is not OCBP-schedulable on a speed-s processor; this contradicts the assumed minimality of I.

From instance I, let us define another mixed-criticality instance I' in the following manner:

$$I' = \bigcup_{J_i=(A_i,D_i,\chi_i,C_i)\in I} \left\{J'_i = (A_i, D_i, \max(\chi_i, k-1), C'_i)\right\}$$

where the WCET function C'_i is defined as follows:

$$C'_i(\ell) = \begin{cases} C_i(\ell), & \text{if } \ell \leq k-1 \\ C_i(k-1), & \text{otherwise} \end{cases}$$

That is, I' is obtained from I by capping the criticality of each job in I at the smaller of its original criticality, and $k-1$. Observe that I' has maximum criticality $\leq (k-1)$, and hence, according to the inductive hypothesis, *I' is guaranteed to be OCBP-schedulable on a speed-s_{k-1} processor.*

Observation 2.1 *All jobs in I with the latest deadline must be of criticality k.*

Proof: If a job of criticality $(k-1)$ or lower has the latest deadline but nevertheless cannot be assigned lowest priority, it follows from the optimality of EDF for scheduling "regular" (non-MC) real-time workloads that the criticality-level $(k-1)$ behavior of I in which each job J_i executes for $C_i(k-1)$ time units is not OCBP-schedulable on a speed-s processor. Since we're assuming that $s \geq s_{k-1}$, this contradicts the inductive hypothesis, which mandates the OCBP-schedulability of I' on a speed-s_{k-1} processor. ■

Let j_k denote such a latest-deadline job of criticality k with deadline d_k, and let j_{k-1} denote the job of criticality $< k$ with the latest deadline, this deadline being at d_{k-1}.

Let c_{k-1}, $c_k(k-1)$, and $c_k(k)$ denote certain cumulative execution requirements, defined as follows:

$$\begin{aligned} c_{k-1} &= \sum_{j \;|\chi_j<k} C_j(k-1) \\ c_k(k-1) &= \sum_{j \;|\chi_j=k} C_j(k-1) \\ c_k(k) &= \sum_{j \;|\chi_j=k} C_j(k) \end{aligned}$$

In words,

- c_{k-1} denotes the cumulative WCET at criticality level $(k-1)$ of all jobs of criticality $< k$;
- $c_k(k-1)$ denotes the cumulative WCET at criticality level $(k-1)$ of all jobs of criticality k; and
- $c_k(k)$ denotes the cumulative WCET at criticality level k of all jobs of criticality k.

Consider now any work-conserving schedule of I upon a speed-s processor, when each job J_i requests exactly $C_i(k-1)$ units of execution Let $\Lambda_1, \Lambda_2, \ldots$ denote the intervals, of cumulative length λ, during which the processor is idle in this schedule.

Observation 2.2 *No J_i with criticality $\chi_i < k$ has a scheduling window that overlaps with Λ_ℓ.*

Proof: Suppose that some job J_i with $\chi_i < k$ were to overlap with Λ_ℓ. This means that in a behavior of criticality level $k-1$, all the jobs which arrive prior to Λ_ℓ complete by the beginning of Λ_ℓ. Hence, such a J_i completes by its deadline in any behavior at its criticality level (which is $\leq k-1$), if it were assigned lowest priority. But this contradicts the assumed non-OCBP-schedulability of I on speed-s processors. ■

Since I' is OCBP-schedulable on a speed-s_{k-1} processor, the behavior in which each job of criticality $< k$ executes for its WCET is guaranteed to complete by d_{k-1}, the latest deadline of any such job, when executing on a speed-s_{k-1} processor. In conjunction with Observation 2.2, it therefore follows that the cumulative WCET's of all these jobs cannot exceed $(d_{k-1} - \lambda) \times s_{k-1}$:

$$c_{k-1} \quad \leq \quad (d_{k-1} - \lambda)\, s_{k-1} \qquad (2)$$

Since I is assumed to not be OCBP-schedulable on a speed-s processor, it must be the case that j_1 cannot be the lowest-priority job on such a processor. Hence, it is necessary that

$$c_{k-1} + c_k(k-1) > (d_{k-1} - \lambda)\ s \tag{3}$$

We now argue from the OCBP-schedulability of I' on a speed-s_{k-1} processor that the behavior of criticality level $(k-1)$, in which all jobs execute for their WCET at criticality $(k-1)$, is guaranteed to complete by the latest deadline d_k of any job. Inequality 4 below, immediately follows.

$$c_{k-1} + c_k(k-1) \quad \leq \quad d_k\ s_{k-1} \tag{4}$$

From the MC schedulability of I on a speed-1 processor, it follows that the behavior of criticality level k, in which each criticality-k job executes for its WCET at criticality k, is guaranteed to complete by the latest deadline d_k of any job. Inequality 5 follows:

$$c_k(k) \quad \leq \quad d_k \tag{5}$$

Observation 2.3 *Consider any work-conserving schedule of I upon a speed-s processor, when each job J_i requests exactly $C_i(\chi_i)$ units of execution. There are no idle intervals in this schedule.*

Proof: If there were an idle interval, any job whose scheduling window spans the idle interval would meet its deadline upon the speed-s processor if it were assigned lowest priority. But this contradicts the assumed non-OCBP-schedulability of I on speed-s processors. ■

Since we are assuming that I is not OCBP-schedulable on a speed-s processor, it must be the case that j_2 cannot be the lowest-priority job on such a processor. Given Observation 2.3 above, it must then be the case that

$$c_{k-1} + c_k(k) > d_k\ s \tag{6}$$

Let x be defined as follows: $x = \frac{d_{k-1}-\lambda}{d_k}$. From Inequalities 3 and 4, we have

$$\begin{aligned} & d_k\ s_{k-1} > (d_{k-1} - \lambda)s \\ \equiv\quad & d_k\ s_{k-1} > x\ d_k\ s \\ \equiv\quad & \frac{s_{k-1}}{x} > s \end{aligned}$$

From Inequality 6 and the Inequalities 2 and 5, we have

$$\begin{aligned} & c_{k-1} + c_k(k) > d_k\, s \\ \Rightarrow\quad & (d_1 - \lambda)\, s_{k-1} + d_k > s\ d_k \\ \equiv\quad & x\ d_k\, s_{k-1} + d_k > s\ d_k \\ \equiv\quad & x\, s_{k-1} + 1 > s \end{aligned}$$

We thus conclude, it must be the case that

$$s < \min\left\{\frac{s_{k-1}}{x}, (x\, s_{k-1} + 1)\right\}$$

It is evident that (s_{k-1}/x) decreases, and $(x\, s_{k-1}+1)$ increases, with increasing x. Therefore the minimum value of s occurs at the value of x that solves the equation $(s_{k-1}/x) = (x\, s_{k-1}+1)$. Solving for x, we get $x = (\sqrt{4s_{k-1}^2+1} - 1)/(2\, s_{k-1})$, and $s \leftarrow (s_{k-1}/x) = (1 + \sqrt{4\ s_{k-1}^2 + 1})/2$. Since this is the definition of s_k in the recurrence (1), Lemma 2 follows. ■

4 Related work

Other than the work in [3, 4] (which are extensively discussed in this document), most prior work on mixed-criticality scheduling has considered different workload models than the one studied in this paper. As can be seen from the discussion below about each of these pieces of work, their goals are also very different from our objective of discovering and quantifying the fundamental limitations (such as intractability and impossibility results, and speedup bounds) of MC scheduling for certification considerations.

Pellizzoni et al. [7], use a reservations-based approach to ensure strong isolation among sub-systems of different criticalities. The focus here is not on maximizing resource utilization, but on ensuring isolation.

De Niz et al. [6] deal with a different problem from the one we here focus on here, in the sense that they do not directly address the certification issue in MC systems. This work observes that the complete inter-criticality isolation offered by reservations may have the undesirable effect of denying a higher-criticality job from meeting its deadline while allowing lower-criticality jobs to complete (this is called *criticality inversion* in [6]). On the other hand, assigning priorities according to criticality may result in very poor processor utilization. An innovative *slack-aware* approach is proposed that builds atop priority-based scheduling (with priorities not necessarily assigned according to criticality), to allow for asymmetric protection of reservations thereby helping to lessen criticality inversion while retaining reasonable resource utilization.

To our knowledge, the scheduling problem that arises from multiple certification requirements, at different criticality levels, was first identified and formalized by Vestal in [8], in the context of the fixed-priority preemptive uniprocessor scheduling of recurrent task systems. Some results concerning EDF scheduling of such systems appear in [5].

5 Context and Conclusions

Thanks to the rapid increase in the complexity and diversity of functionalities performed by safety-critical embedded systems, the cost and complexity of obtaining certification for such systems is fast becoming a serious concern [2]. We have argued in this document that in mixed-criticality systems, these certification considerations give rise to fundamental new resource allocation and scheduling challenges, and that these challenges are not adequately addressed by conventional real-time scheduling theory. In prior work, we have therefore proposed a novel job model that is particularly appropriate for representing mixed-criticality workloads, and have studied basic properties of this model. We have demonstrated that schedulability analysis for mixed-criticality systems is highly intractable, even for very simple workloads comprised of independent jobs, all of which have one of only two possible criticality levels, and which is being scheduled on a fully preemptive uniprocessor platform. We have previously shown how this intractability can be dealt with by providing faster processors for dual-criticality systems; in this paper, we have generalized these results to systems with more than two criticalities.

In addition to the specific results presented here, we consider the new techniques that we have introduced to be significant contributions of this paper. We are optimistic that the new techniques will facilitate further research into the very important problem of mixed-criticality scheduling and resource allocation.

References

[1] Audsley, N. C. *Flexible Scheduling in Hard-Real-Time Systems.* PhD thesis, Department of Computer Science, University of York, 1993.

[2] Barhorst, J., Belote, T., Binns, P., Hoffman, J., Paunicka, J., Sarathy, P., Stanfill, J. S. P., Stuart, D., and Urzi, R. White paper: A research agenda for mixed-criticality systems, April 2009. Available at http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR.

[3] Baruah, S. Mixed criticality schedulability analysis is highly intractable. Available at http://www.cs.unc.edu/~baruah/Pubs.shtml, 2009.

[4] Baruah, S., Li, H., and Stougie, L. Towards the design of certifiable mixed-criticality systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS)* (April 2010), IEEE.

[5] Baruah, S., and Vestal, S. 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.

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

[7] Pellizzoni, R., Meredith, P., Nam, M. Y., Sun, M., Caccamo, M., and Sha, L. Handling mixed criticality in SoC-based real-time embedded systems. In *Proceedings of the International Conference on Embedded Software (EMSOFT)* (Grenoble, France, 2009), IEEE Computer Society Press.

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