Towards the design of certifiable mixed-criticality systems

Sanjoy Baruah[*] Haohan Li
Department of Computer Science
The University of North Carolina
Chapel Hill, NC. USA

Leen Stougie[†]
Division of Operations Research
Dept. of Economics and Business Administration
Vrije Universiteit and CWI
Amsterdam, The Netherlands

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. In this paper, we propose a formal model for representing such mixed-criticality workloads. We demonstrate the intractability of determining whether a system specified in this model can be scheduled to meet all its certification requirements. For dual-criticality systems – systems subject to two sets of certification requirements – we quantify, via the metric of processor speedup factor, the effectiveness of 2 techniques (reservation-based scheduling and priority-based scheduling) that are widely used in scheduling such mixed-criticality systems.

1 Introduction

Due to cost and related considerations, 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. (Indeed, mixed criticalities has 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 highly likely that some of these functionalities will be more important (more "critical") to the overall welfare of the platform than others. For instance, it is more important to the correct behavior of an automotive control system that the anti-lock brake system (ABS) works correctly than that the on-board radio does so. This aspect of mixed criticalities is widely studied by embedded systems designers, who typically address such differences in criticalities by means of priority-based scheduling approaches: more critical functionalities are accorded greater priority so that they will be less likely to suffer performance degradation in the event of system overload . (Lately, much more sophisticated approaches have been proposed for dealing with such systems, based on "servers" that isolate functionalities of different criticalities from each other via the technique of virtualization.)

However, there is another aspect [3] 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 defense reconnaissance and surveillance. The functionalities on board such UAV's can be classified into two categories:

- The *flight-critical* functionalities: these are the functions that must be performed by the aircraft in order to ensure its safe operation.

- The *mission-critical* functionalities, which are concerned with the UAV's reconnaissance and surveillance objectives. These could include capturing images from the ground, transmitting these images to the base station, perhaps (depending on how sophisticated the UAV's capabilities are) doing some image processing in order to track potential targets and

[*] 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.

more generally to determine where next to direct its surveillance activities, etc.

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, and similar organizations in other parts of the world. 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. These 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 Certification Authority is likely to specify 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 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 are permitted to be used, analyzing the resulting programs very carefully to identify the worst-case execution path through it, and subjecting this worst-case path to careful analysis using rigorous cycle-counting under extreme pessimistic assumptions regarding cache state etc.

- For mission-critical validation, it may suffice to estimate the WCET of the same piece of code by performing a large number of simulation experiments on the code, covering what we believe are all extremal behaviors of the system, measuring the run-times to determine the largest value, and perhaps inflating the largest observed value by an additional "fudge" factor to give us greater confidence in the result. 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 in 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 the techniques associated with mission-critical certification, 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 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.* This paper reports on some of our initial attempts towards such a theory.

Alternative approaches. In order to justify the need for a new theory, we need to show that current scheduling theory is inadequate for addressing the needs of mixed-criticality certification. Example 1 above illustrates that a *reservations-based approach*, whereby we a priori reserve adequate computing capacity to guarantee each job enough execution to meet its WCET requirement at its criticality level, is pessimistic. (Indeed, the example is easily modified to make the pessimism arbitrarily large: in the worst case, each criticality level could end up needing to be assigned the equivalent of its own dedicated processor.)

But what about *priority-based scheduling*, the other technique commonly used by systems engineers in dealing with mixed criticalities? In the specific case of Example 1, observe that assigning greater priority to the higher-criticality job – a "criticality-monotonic" scheduling policy – worked. However, it is in fact not too difficult to construct examples (similar to the one above), in which such a criticality-monotonic scheme will also fail. For instance, consider a 2-job system in which the lower-criticality job has a much earlier deadline, and a far smaller WCET, than the higher-criticality job. A criticality-monotonic schedule may fail to pass certification at the lower criticality level even though there may be adequate computing capacity to meet both jobs' WCET's at their specified criticality levels. These drawbacks of such a criticality-monotonic approach are also highlighted in [7].

Organization of this paper. Current real-time workload models are inadequate for representing MC systems. Hence in Section 2, we propose a formal model for representing mixed-criticality real-time systems. This formal model extends the conventional model of a real-time job by allowing for the specification of different WCET's for a job at different criticality levels. We also define and explain important concepts concerning MC scheduling, by, for instance, highlighting the inherent *on-line* nature of the MC scheduling problem, and by formally capturing the notion of what it actually means for a MC system to be schedulable. We cite results proving that schedulability analysis of MC systems is highly intractable. In Section 3, we consider a special case for which tractable analysis is possible, and present and prove the correctness of an algorithm for performing such analysis. In Sections 4 and 5, we study the two techniques that are most widely used in designing mixed-criticality systems for certifiability; we quantify the sub-optimality of both techniques via the metric of *processor speedup factor*. We briefly survey some other work on mixed-criticality real-time systems in Section 6.

2 Model and definitions; prior results

In this section we formally define the mixed-criticality job model, and explain terms and concepts used throughout the remainder of this document. 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], we will see in this document that 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.
- $C_i : N^+ \to R^+$ 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

[1] Some partial results concerning the scheduling of such task systems may be found in [10, 6].

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.

Dual criticality instances. A *dual criticality* instance I is an MC instance with the additional property that all jobs have criticality either one or two: $\chi_i \in \{1, 2\}$ for all $J_i \in I$. The example instance considered in Example 1 is clearly a dual criticality instance (this trivially follows since the instance is comprised of just two jobs); so is the example (with more than 2 jobs) that will be discussed below in Section 2.1.

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 values of A_i and D_i are known from the specification of the job. However, 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*.)

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. A *clairvoyant* scheduling strategy knows the behavior of I — i.e., the value of γ_i for each $J_i \in I$ — prior to generating a schedule for I. By contrast, 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].

2.1 An example

Consider an MC instance I comprised of 4 jobs. Job J_2 has criticality level 1 (which is the lower criticality level), and the other 3 jobs have the higher criticality level 2. 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 example 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, by definition, 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 revealed to be 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 particular behavior has criticality 2 (since γ_4, at 2, is greater than $C_4(1)$ which has value 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.

[2] Another problem — the *scheduling strategy verification problem* — verifies whether a given scheduling strategy is correct for a given problem instance. We will not discuss the scheduling strategy verification problem much in this document, since a thorough analysis requires us to first agree on what constitutes an acceptable representation of a scheduling strategy.

2.2 The complexity of MC schedulability testing

Unfortunately, determining whether a given MC instance is MC schedulable or not turns out to be higly intractable:

Theorem 1 (From [4]) *The MC schedulability problem — given an MC instance, determine whether it is MC-schedulable — is NP-hard in the strong sense. This hardness result holds even in the highly restricted case where all jobs in the MC instance 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).

Note. In fact, it is not at all clear that a polynomial time *scheduling strategy verification* algorithm exists. Thus, the MC Schedulability problem may very well not belong to NP, but to a higher complexity class in the polynomial-time hierarchy [9]. It remains open to settle the exact complexity of the problem.

3 Special case: all deadlines equal

The intractability result in Section 2.2 above tells us that the MC schedulability analysis problem is intractable even when all jobs have the same arrival time. What about the case when all jobs have the same deadline? In this case, it turns out that the problem is in fact tractable, even when arrival times may be different. In this section, we derive an efficient polynomial-time MC schedulability testing algorithm for such mixed criticality systems.

Consider an MC instance I satisfying the constraint that for all $J_i \in I$, the deadline parameters D_i are all equal — we will denote this common deadline by D.

We first derive a necessary condition for such an instance I to be MC schedulable. Consider the behavior of I in which each job J_i needs exactly $C_i(\ell)$ units of execution – by definition, such a behavior has criticality level ℓ. In order that I be deemed MC-schedulable, it is therefore necessary that this behavior be schedulable in such a manner that each job J_i with $\chi_i \geq \ell$ receives at least $C_i(\ell)$ units of execution between its release time A_i and the common deadline D. For each $\ell \geq 1$, let F_ℓ denote the completion time of the last such job in any preemptive work-conserving schedule, in which each J_i with $\chi_i \geq \ell$ receives exactly $C_i(\ell)$ units of execution. It is evident that F_ℓ is easily computed in polynomial time, as shown in Figure 1. A necessary MC-schedulability condition is hence as follows:

$$\forall \ell \; : \; \ell \geq 1 : F_\ell \leq D \, . \qquad (1)$$

We claim that this necessary condition is in fact also *sufficient*, and that the *criticality-monotonic* (CM) on-line scheduling algorithm — schedule at each instant an available job needing execution that is of greatest criticality — will successfully schedule any instance I that satisfies this necessary condition. To see why this must be true, consider any behavior of I that has some criticality level ℓ. By definition of criticality-monotonic scheduling, the scheduling of jobs of criticality ℓ or higher is not at all effected by the presence of lower-criticality jobs — such lower-criticality jobs get to execute only when there are no jobs of criticality ℓ or higher awaiting execution. Hence, CM can be thought of as a work-conserving scheduling algorithm on those jobs of I that are of criticality level ℓ or higher; by our MC-necessary schedulability condition (Condition 1 above), this does not exceed the common deadline D.

Theorem 2 *The Criticality-monotonic (CM) on-line scheduling algorithm is optimal for the scheduling of MC instances in which all jobs have the same deadline.*

Note. Notice that $\max_{\ell \geq 1} F_\ell$ is the ***makespan*** of I; hence, this technique can be used to compute the makespan of any MC instance in which jobs do not have deadlines specified.

4 Sufficient MC-schedulability testing: a reservations approach

Since determining MC schedulability is highly intractable even for dual-criticality instances, we now seek sufficient MC-schedulability conditions that can be implemented with polynomial time-complexity, but that can nevertheless make quantitative performance guarantees.

We make an assumption here 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 worst-case reservations approach. As stated in Section 1, one straightforward approach is to map each MC job J_i into a "traditional" (i.e., non MC) job with the same arrival time and deadline as J_i, and a WCET equal to $C_i(\chi_i)$, and determine whether the resulting collection of traditional jobs is schedulable using some preemptive uniprocessor scheduling algorithm such as EDF[3]. Since

[3] In fact, this approach forms the basis of current practice, as formulated in the ARINC-653 standard: each J_i is guaranteed $C_i(J_i)$ units

1. Let I_ℓ denote the jobs in I with criticality level ℓ or higher: $I_\ell = \{J_i \in I \mid \chi_i \geq \ell\}$.
2. Let $J_1, J_2, \ldots, J_{n_\ell}$ denote all the jobs in I_ℓ ordered by non-decreasing release times: $A_i \leq A_{i+1}$ for all $i, 1 \leq i < n_\ell$.
3. Consider the sequence $f_1, f_2, \ldots, f_i$ of numbers defined according to the following recurrence

$$\begin{aligned} f_1 &= A_1 + C_1(\ell) \\ f_i &= \max(f_{i-1}, A_i) + C_i(\ell), \text{ for } i > 1 \end{aligned}$$

4. $F_\ell \leftarrow f_{n_\ell}$.

Figure 1: Computing F_ℓ: the makespan of a preemptive work-conserving schedule for that behavior of I in which each job $J_i \in I$ of criticality $\geq \ell$ executes for exactly $C_i(\ell)$ time units.

the EDF schedule for n traditional jobs can be obtained in time polynomial in n, this test can clearly be done in polynomial time. We will refer to dual-criticality instances that are deemed schedulable by this test as *worst-case reservations schedulable* instances.

Lemma 1 *If dual-criticality instance I is worst-case reservations schedulable on a given processor, then I is MC schedulable on the same processor.*

Proof: Consider first any behavior of I of criticality level 1. Each job $J_i \in I$ needs at most $C_i(1)$ units of execution. Since $C_i(1) \leq C_i(2)$ for all jobs J_i, the worst-case reservations approach ensures that at least $C_i(1)$ units of execution can be accommodated for each job. Therefore, an on-line scheduling algorithm such as EDF would schedule I's criticality-level 1 behavior to meet all deadlines.

Consider now any criticality-level 2 behavior of I. Recall the assumption we have made, that no J_i with $\chi_i = 1$ executes for more than $C_i(1)$ units. Hence, in a behavior of I of criticality level 2, each J_i with $\chi_i = 1$ executes for at most $C_i(1)$ and each J_i with $\chi_i = 2$ executes for at most $C_i(2)$. But this is exactly how much execution has been reserved for each job in the worst-case reservations approach; hence once again an optimal on-line scheduling algorithm such as EDF would schedule I's criticality-level 2 behavior to meet all deadlines. ■

Lemma 1 asserts that worst-case reservations schedulability testing represents a sufficient MC-schedulability test. But how far from an exact test is this? The following two lemmas provide an answer, in terms of the *processor speedup metric*: the minimum multiplicative factor by which processors must be made faster in order to compensate for the inexactness of the test.

Lemma 2 *If dual-criticality instance I is MC schedulable on a given processor, then I is worst-case reservations schedulable on a processor that is twice as fast.*

Proof: If dual-criticality instance I is MC schedulable on a unit-speed processor, it follows that the behavior in which each criticality-one job J_i executes for $C_i(1)$ time units and all criticality-two jobs experience no execution is schedulable using EDF. Similarly, the behavior in which all criticality-one jobs do not execute at all while each criticality-two job J_i executes for $C_i(2)$ time units is also schedulable using EDF. Hence, a processor-sharing schedule[4] on a speed-two processor would ensure that each job J_i gets to execute for as much as $C_i(\chi_i)$ time units, regardless of the behavior of other jobs. ■

The drawback of the worst-case reservations approach is that while the semantics of the MC model do not require criticality-1 jobs to complete if any criticality-2 job executes for more than its criticality-1 WCET, such knowledge is not exploited by the worst-case reservations approach. This severely limits the effectiveness of this approach, as formalized by the following lemma.

Lemma 3 *There exist dual-criticality instances that are MC schedulable on a given processor, but not worst-case reservations schedulable on a processor that is less than twice as fast.*

Proof: Consider the instance I comprised of the following two jobs (as before, we specify the WCET function of each job for the two criticality levels by explicit enumeration: $[C_i(1), C_i(2)]$).

- $J_1 = (0, D, 1, [D, D])$
- $J_2 = (0, D, 2, [0, D])$

of execution in a *time partitioned* schedule, obtained by partitioning the time-line into distinct slots and only permitting particular jobs to execute in each such slot.

[4] For instance, the time-line could be partitioned into equi-sized slots of arbitrarily short duration, with alternate slots devoted to jobs of each criticality level.

This instance is MC-schedulable on a unit-speed processor by the scheduling strategy of assigning greater priority to J_2. Any behavior in which J_2 has an execution requirement > 0 is a behavior of criticality level 2, and hence J_1 is not obliged to meet its deadline in this behavior. In any behavior of criticality level 1, on the other hand, J_2 has an execution requirement of zero and J_1 gets to execute for the entire duration $[0, D)$ and thereby meets its deadline.

However, the worst-case reservations approach would require that $C_1(\chi_1) = C_1(1) = D$ units of execution be reserved for J_1, and $C_2(\chi_2) = C_2(2) = D$ units of execution be reserved for J_2 over this interval as well. Both these reservations can only be accommodated on a speed-≥ 2 processor. ■

Generalization to more than 2 **criticality levels.** Lemmas 2 and 3 show that the processor speedup factor of 2 is tight for dual criticality systems. Straightforward extensions of these lemmas can be used to show that the natural generalization of the worst-case reservations approach to systems with $k > 2$ criticality levels has a tight processor speedup factor of k.

5 A priority-based approach

In this section, we present another sufficient schedulability condition that can also be implemented in polynomial time, but offers a performance guarantee (as measured by the processor speedup factor) that is superior to the performance guarantee offered by time-partitioning. As in Section 5, 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 a dual-criticality 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 approach commonly referred to in the real-time scheduling literature as the "Audsley approach" [1, 2]. 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. This does not mean that the mixed-criticality instance is not MC Schedulable, as the following example illustrates.

Example 2 Let $J_i \rhd J_j$ denote that J_i has greater priority than J_j. Consider three jobs J_1, J_2 and J_3 with $\chi_2 = 1$ and $\chi_1 = \chi_3 = 2$, for which $C_1(2) > C_1(1)$. In considering the priority ordering $J_1 \rhd J_2 \rhd J_3$, we would require that J_3 receive $C_3(2)$ units of execution if J_1 receives $C_1(2)$ units of execution and J_2 receives $C_2(2)$ (which, by assumption, equals $C_2(1)$) units of execution. This is despite the fact that *if* J_1 *receives more than* $C_1(1)$ *units of execution, it is not necessary that* J_2 *receive any execution.* In other words, we do not attempt to take advantage of this knowledge, that the behavior is of criticality level 2 and hence criticality-one jobs need not execute to completion, to modify the scheduling strategy during run-time; instead, we stick with the priority ordering that was determined off-line and execute jobs according to this priority ordering. ■

We point out that this algorithm for assigning priorities runs in polynomial time and is guaranteed to find a total priority ordering of the jobs, if one exists, such that scheduling according to this priority ordering is a correct on-line scheduling strategy. Hence, the non-optimality of OCBP scheduling (as highlighted in Example 2), is a consequence of the fact that no such priority-ordering approach can be optimal rather than a failure to find an appropriate ordering.

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

Lemma 4 *If dual-criticality 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, $J_1, J_2, \ldots, J_n$ 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. ■

Lemma 5 *If dual-criticality instance I is MC schedulable on a given processor, then I is OCBP-schedulable on a processor that is $(1+\sqrt{5})/2$ times as fast.*

Proof: (We observe that $(1+\sqrt{5})/2$ is the famous constant typically denoted as Φ, and often known as the *Golden Ratio*.)

Let I denote a minimal instance I that is MC-schedulable on a speed-1 processor, but not OCBP-schedulable on a speed-s processor for some $s > 1$. We will show that $s < (1+\sqrt{5})/2$.

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[5] contains t. If there were such a t, it would follow that either the 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; this contradicts the assumed minimality of I.

Observation 5.1 *The job[s] in I with the latest deadline must [all] be of criticality* 2.

Proof: If a criticality-1 job has the latest deadline but nevertheless cannot therefore be assigned lowest priority, it follows from the optimality of EDF for scheduling "regular" (non-MC) real-time workloads that the criticality-level 1 behavior of I in which each job J_i executes for $C_i(1)$ time units is not schedulable. This contradicts the assumed MC schedulability of I. ■

Let j_2 denote such a latest-deadline job of criticality 2 with deadline d_2, and let j_1 denote the job of criticality 1 with the latest deadline, this deadline being at d_1.

Let c_1, $c_2(1)$, and $c_2(2)$ denote certain cumulative execution requirements, defined as follows:

$$\begin{aligned} c_1 &= \sum_{j \,|\chi_j = 1} C_j(1) \\ c_2(1) &= \sum_{j \,|\chi_j = 2} C_j(1) \\ c_2(2) &= \sum_{j \,|\chi_j = 2} C_j(2) \end{aligned}$$

In words, c_1 denotes the cumulative WCET of all criticality-one jobs, while $c_2(\ell)$ denotes the cumulative WCET at criticality level ℓ, of all criticality-two jobs.

Consider now any work-conserving schedule of I upon a speed-s processor, when each job J_i requests exactly $C_i(1)$ units of execution[6]. Let $\Lambda_1, \Lambda_2, \ldots$ denote the intervals, of cumulative length λ, during which the processor is idle in this schedule.

Observation 5.2 *No J_i with criticality $\chi_i = 1$ has a scheduling window that overlaps with Λ_ℓ.*

Proof: Suppose that some criticality-1 job J_i were to overlap with Λ_ℓ. This means that in a behavior of criticality level one, all the jobs which arrive prior to Λ_ℓ complete by the beginning of Λ_ℓ. Hence, J_i would complete by its deadline in any behavior of criticality level one, if it were assigned lowest priority. But this contradicts the assumed non-OCBP-schedulability of I on speed-s processors. ■

Since I is assumed to be MC schedulable on a speed-1 processor, the behavior in which each criticality-one job executes for its WCET is guaranteed to complete by d_1, the latest deadline of any criticality-one job. It therefore follows from Observation 5.2 that the cumulative WCET's of all criticality-one jobs cannot exceed $(d_1 - \lambda)$:

$$c_1 \quad \leq \quad d_1 - \lambda \tag{2}$$

Since we're assuming that I is not 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

$$c_1 + c_2(1) > (d_1 - \lambda)\, s \tag{3}$$

We now argue from the MC schedulability of I on a speed-1 processor that the behavior of criticality level one, in which all jobs execute for their WCET at criticality one, is guaranteed to complete by the latest deadline d_2 of any job. Inequality 4 below, immediately follows.

$$c_1 + c_2(1) \quad \leq \quad d_2 \tag{4}$$

It similarly follows from the MC schedulability of I on a speed-1 processor that the behavior of criticality level two, in which each criticality-two job executes for its WCET at criticality two, is guaranteed to complete by the latest deadline d_2 of any job. Inequality 5 follows:

$$c_2(2) \quad \leq \quad d_2 \tag{5}$$

Observation 5.3 *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*[7]. *There are no idle intervals in this schedule.*

[5] Recall that the *scheduling window* of a job J_i is the interval $[A_i, D_i))$.

[6] We are not attempting to meet deadlines in this schedule, simply keeping the processor active whenever there are jobs remaining that have arrived but not completed execution, regardless of whether their deadlines are met or not.

[7] As in Observation 5.2, we are not attempting to meet deadlines 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 5.3 above, it must then be the case that

$$c_1 + c_2(2) > d_2\ s \tag{6}$$

Let x be defined as follows:

$$x = \frac{d_1 - \lambda}{d_2}$$

From Inequalities 3 and 4, we have

$$\begin{aligned} & d_2 > (d_1 - \lambda)s \\ \equiv \quad & d_2 > x\ d_2\ s \\ \equiv \quad & \frac{1}{x} > s \end{aligned}$$

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

$$\begin{aligned} & c_1 + c_2(2) > s\ d_2 \\ \Rightarrow \quad & (d_1 - \lambda) + d_2 > s\ d_2 \\ \equiv \quad & x\ d_2 + d_2 > s\ d_2 \\ \equiv \quad & x + 1 > s \end{aligned}$$

We thus conclude, it must be the case that

$$s < \min\Big\{\frac{1}{x}, (1+x)\Big\}$$

It is evident that $\frac{1}{x}$ decreases, and $(1 + x)$ increases, with increasing x. Therefore the minimum value of s occurs at the value of x that solves the equation $\frac{1}{x} = (1 + x)$. Solving for x, we get $x = (\sqrt{5} - 1)/2$, and $s \leftarrow (1 + x)$ = $(1 + \sqrt{5})/2$. Lemma 5 follows. ■

Generalization to more than 2 criticality levels. Although the techniques used in the proof of Lemma 5 do indeed generalize to MC systems with $k > 2$ criticality levels, such a generalization is not as straightforward as it was in the reservations approach. We have recently proved in [5] that for MC systems with k criticality levels, an upper bound on the speedup factor s_k can be expressed as a recurrence:

$$\begin{aligned} s_1 &= 1 \\ s_k &= \frac{1 + \sqrt{4\ s_{k-1}^2 + 1}}{2}, \quad k > 1 \end{aligned} \tag{7}$$

(Note that for $k = 2$, we get the same result as we had obtained above: an upper bound on the speedup factor of $(1 + \sqrt{5})/2$.)

It can be shown that the RHS of Equation 7 approaches $(k/2)$ as $k \to \infty$; hence, this priority-based scheduling approach asymptotically becomes at least twice as good as the reservations-based approach from the perspective of processor speedup factors, with increasing numbers of criticality levels.

Discussion. We thus see that of the two polynomial-time sufficient MC-schedulability conditions for dual-criticality systems — reservations based and priority based — the priority-based approach requires a processor speedup of no more than the Golden Ratio (Φ; approx 1.618) whereas the reservations-based approach requires a processor speedup of 2.

6 Related work

Most prior work on mixed-criticality scheduling (such as the work described in [8, 7, 6, 10]) 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. [8], use a reservations-based approach to ensure strong isolation among sub-systems of different criticalities; this paper proposes innovative design and architectural techniques for preserving such isolation despite some necessary interaction (e.g., in the sharing of additional non-preemptable resources) between jobs of different criticalities. The focus here is not on maximizing resource utilization, but on ensuring isolation; hence, this research does not attempt to avoid the disadvantage of the reservations-based approach that we have highlighted here, of potentially making poor use of system resources.

De Niz et al. [7] 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. Nevertheless, [7] contains very interesting and novel ideas that merit mention. This work observes that the complete inter-criticality isolation offered by the reservations approach 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 [7]). 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 [10], in the context of the fixed-priority preemptive uniprocessor scheduling of recurrent task systems. Some results concerning EDF scheduling of such systems appear in [6].

7 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 [3]. 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. 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 shown how this intractability can be dealt with by providing faster processors.

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

References

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

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

[3] 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.

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

[5] Baruah, S., Li, H., and Stougie, L. Mixed-criticality scheduling: improved resource-augmentation results. In *Proceedings of the ICSA International Conference on Computers and their Applications (CATA)* (April 2010), IEEE.

[6] 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.

[7] 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.

[8] 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.

[9] Stockmeyer, L. The polynomial-time hierarchy. *Theoretical Computer Science 3* (1977), 1–22.

[10] 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.