Scheduling real-time mixed-criticality jobs

Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie

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 interesting scheduling problems, that cannot be satisfactorily addressed using techniques from conventional scheduling theory. In this paper, we study a formal model for representing such mixed-criticality workloads. We demonstrate first the intractability of determining whether a system specified in this model can be scheduled to meet all its certification requirements, even for systems subject to merely two sets of certification requirements. Then we quantify, via the metric of processor speedup factor, the effectiveness of two techniques, reservation-based scheduling and priority-based scheduling, that are widely used in scheduling such mixed-criticality systems, showing that the latter of the two is superior to the former. We also show that the speedup factors we obtain are tight for these two techniques.

I. Introduction

Due to considerations of cost, energy efficiency, thermal dissipation, etc., there is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. It is typically the case that not all these functionalities are equally critical for the overall successful performance of the system. The analysis of such *mixed criticality* (MC) systems has been identified as one of the core foundational focal areas in the emerging discipline of Cyber Physical Systems. Coming up with procedures that will allow for the cost-effective certification of such mixed-criticality systems has been recognized as a unique, particularly challenging, collection of problems [**?**]. Recognizing these challenges, several US government R&D organizations including the Air Force Research Laboratory, the National Science Foundation, the National Security Agency, the National Aeronautics and Space Administration, etc., have led initiatives such as the Mixed Criticality Architecture Requirements (MCAR) program aimed at streamlining the certification process for safety-critical embedded systems; these initiatives have brought together participants from industry, academia, and standards bodies to seek out more advanced, efficient, and cost-effective certification processes. Within this setting, new interesting scheduling problems arise that are the focus of this paper.

We illustrate this by an example from the domain of unmanned aerial vehicles (UAV's), used for defense reconnaissance and surveillance. The functionalities on board such UAV's may be classified into two levels of criticality:

- Level 1: the *mission-critical* functionalities, concerning reconnaissance and surveillance objectives, like capturing images from the ground, transmitting these images to the base station, etc.

Some preliminary results and the first parts of Theorems 5 and 6 in this paper have been published in conference proceedings [**?**], [**?**], [**?**]

Baruah and Li are with the Department of Computer Science, The University of North Carolina at Chapel Hill, USA ({baruah,lihaohan}@cs.unc.edu).

Bonifaci and Megow are with Max-Planck-Institut für Informatik, Saarbrücken, Germany ({bonifaci,nmegow}@mpi-inf.mpg.de).

D'Angelo is with Università degli Studi dell'Aquila, Monteluco di Roio, Italy (gianlorenzo.dangelo@univaq.it).

Marchetti-Spaccamela is with Sapienza University of Rome, Rome, Italy (alberto@dis.uniroma1.it).

Stougie is with Division of Operations Research, Dept. of Economics and Business Administration, Vrije Universiteit, and CWI, Amsterdam, The Netherlands (lstougie@feweb.vu.nl).

- Level 2: the *flight-critical* functionalities: to be performed by the aircraft to ensure its safe operation.

For permission to operate such UAV's over civilian airspace (e.g., for border surveillance), it is mandatory that its flight-critical functionalities be certified correct by civilian Certification Authorities (CA's), which tend to be very conservative concerning the safety requirements. However, these CA's are not concerned with the mission-critical functionalities: these must be validated separately by the system designers (and presumably the customers – those who will purchase the aircraft). The latter are also interested in ensuring the correctness of the level 2 (i.e., flight-critical) functionalities, but the notion of correctness adopted in validating these functionalities is typically less rigorous than the one used by the civilian CA's.

This difference in correctness criteria may be expressed by different *Worst-Case Execution Times* (WCET) estimates for the execution of a piece of real-time code. In fact, the CA and the system designers (and other parties responsible for validating the mission-critical functionalities) will each have their own tools, rules, etc., for estimating WCET; the value so obtained by the CA is likely to be larger (more pessimistic) than the one obtained by the system designer. We illustrate via a (contrived) example.

Example 1: Consider a system comprised of two jobs: J_1 is flight-critical while J_2 has lower mission-critical criticality. Both jobs arrive at time-instant 0, and have their deadlines at time-instant 10. For $i \in \{1, 2\}$, let $P_i(1)$ denote the WCET estimate of job J_i as made by the system designer, and $P_i(2)$ the WCET estimate of job J_i as made by the CA.

Suppose that $P_1(1) = 3$, $P_1(2) = 5$ and $P_2(1) = P_2(2) = 6$. Consider the schedule that first executes J_1 and then J_2.

- The CA responsible for safety-critical certification would determine that J_1 completes latest by time-instant 5 and meets its deadline. (Note that if the execution time of J_1 is 5 then in the worst case it is not possible to complete J_2 by its deadline; however, this CA is not interested in J_2; hence the system passes certification.)
- The system designers (and other parties responsible for validating the correctness of the mission-critical functionalities) determine that J_1 completes latest by time-instant 3, and J_2 by time-instant 9. Since both jobs complete by their deadlines, the system is determined to be correct by its designers.

We thus see that the system is deemed as being correct by both the CA and the designers, despite the fact that the sum of the WCET's of the jobs at their own criticality levels (6 and 5) exceeds the length of the scheduling window over which they are to execute.

■

Current practice in safety-critical embedded systems design for certifiability is centered around the technique of "space-time partitioning," as codified in, e.g., the ARINC-653 standard [**?**], [**?**]. Loosely speaking, *space partitioning* means that each application is granted exclusive access to some of the physical resources on board the platform, and *time partitioning* means that the time-line is divided into slots with each slot being granted exclusively to some (pre-specified) application. Interactions among the partitioned applications may only occur through a severely limited collection of carefully-designed library routines. This is one of several *reservation-based* approaches, in which a certain amount of the capacity of the shared platform is reserved for each application, that have been considered for designing certifiable mixed-criticality systems. It is known that reservation-based approaches tend to be pessimistic (in the sense of under-utilizing platform resource); for instance, a reservation-based approach to the example above would require that 5 units of execution be reserved for job J_1, and 6 units for job J_2, over the interval $[0, 10)$.

The central thesis explored in our research, as illustrated in Example 1, is that *efficient resource utilization in systems that are subject to multiple different correctness criteria requires the development of new approaches for resource-allocation and scheduling*. This paper describes our efforts to date towards developing such approaches.

1) We have proposed [**?**], [**?**] a formal model for representing mixed-criticality real-time systems – this mixed-criticality (MC) 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. This model is described in Section II.
2) We have studied the computational complexity of mixed-criticality scheduling problems. In previous papers [**?**], [**?**], the problem of deciding schedulability of a given MC system was conjectured to be NP-hard. We provide a proof of this result here, in Section III. However, the exact complexity of the problem remains open, since it is not clear whether the problem is actually in NP. We prove that it is, if the number of criticality levels is a constant. Otherwise, we can only show that it is in PSPACE. In the same section we present an algorithm that decides MC-schedulability efficiently for the special case in which all jobs have the same deadline.
3) In Section IV we quantitatively evaluate, via the metric of *processor speedup factor* (cf. resource augmentation in performance analysis of approximation algorithms, as initiated in [**?**]), two techniques that are currently widely used for resource-allocation and scheduling in mixed criticality systems subject to certification. Our results here extend the results in [**?**], which considered the techniques for *dual-criticality systems*, in which there are only two different criticality levels (as in Example 1 above). Our results improve the results in [**?**], where also the techniques for L criticality levels are studied. Moreover, we prove here that our results are tight.
4) In Section V we show that the bounds of Section IV do not hold for restricted classes of mixed-criticality systems. We focus on systems in which the range of WCET parameter estimate values is restricted. More specifically, we consider mixed-criticality systems in which there are at most two distinct criticality levels (as in Example 1 above), and we assume that the sum of the level 2 WCET estimates is bounded by a constant β times the sum of the level 1 WCET estimates of these jobs. We show that, depending on β, performance guarantees can be made that are superior to the tight bounds of the general case.

II. Preliminaries

Although the examples that we considered in Section I were characterized by just two criticality levels, systems may in general have more criticality levels defined. (For instance, the RTCA DO178-B standard, widely used in the aviation industry, specifies 5 different criticality levels, with the system designer expected to assign one of these criticality levels to each job. The ISO 26262 standard, used in the automotive domain, specifies 4 criticality levels, known in the standard as "safety integrity levels" or SIL's.)

Accordingly, the formal model that we propose allows for the specification of arbitrarily many criticality levels. Let $L \in \mathbb{N}^+$ denote the number of distinct criticality levels in the MC system being modeled. A *job* in this MC system is characterized by a 4-tuple of parameters: $J_j = (r_j, d_j, \chi_j, P_j)$, where

- $r_j \in \mathbb{Q}_+$ is the release time;
- $d_j \in \mathbb{Q}_+$ is the deadline, $d_j \geq r_j$;
- $\chi_j \in \mathbb{N}_+$ is the criticality of the job;
- $P_j \in \mathbb{Q}_+^L$ is a vector, the ℓ-th coordinate of which specifies the worst-case execution time (WCET) estimate of job J_j at criticality level ℓ. In a job-specification we will usually represent it by $(P_j(1), \ldots, P_j(L))$.

We will, for the most part, assume that $P_j(\ell)$ is monotonically non-decreasing with increasing ℓ. This is a reasonable assumption: these $P_j(\ell)$ values represent *upper bounds*, at different degrees of confidence, on the WCET of the job. Larger values of ℓ correspond to greater degrees of confidence, and are therefore likely to be larger. At any moment, we call a job *available* if its release time has passed and the job has not yet completed execution.

An instance I of the MC-schedulability problem consists of a set of n jobs. In this paper we assume that there is only one machine (processor) to execute the jobs. We assume that this processor is *preemptive*: executing jobs may have their execution interrupted at any instant in time and resumed later, with no additional cost or penalty.

To define MC-schedulability we define the notion of a *scenario*. Each job J_j requires an amount of execution time p_j within its *time window* $[r_j, d_j]$. The value of p_j is not known from the specification of J_j, but is only discovered by actually executing the job until it *signals* that it has completed execution. This characterizes the uncertainty of the problem. We call a collection of realized values $(p_1, p_2, \ldots, p_n)$ a *scenario* of instance I.

We define the *criticality level*, or simply criticality, of a scenario $(p_1, p_2, \ldots, p_n)$ of I as the smallest integer ℓ such that $p_j \leq P_j(\ell)$ for all $j = 1, \ldots, n$. (If there is no such ℓ, we define that scenario to be *erroneous*.)

Definition 1: A schedule for a scenario $(p_1, \ldots, p_n)$ of criticality ℓ is *feasible* if every job J_j with $\chi_j \geq \ell$ receives execution time p_j during its time window $[r_j, d_j]$.

A *clairvoyant* scheduling policy knows the scenario of I, i.e., $(p_1, \ldots, p_n)$, prior to determining a schedule for I.

Definition 2: An instance I is *clairvoyantly-schedulable* if for each non-erroneous scenario of I there exists a feasible schedule.

By contrast, an *on-line* scheduling policy discovers the value of p_j only by executing J_j until it signals completion. In particular, the criticality level of the scenario becomes known only by executing jobs. At each time instant, scheduling decisions can be based only on the partial information revealed thus far.

Definition 3: An on-line scheduling policy is *correct* for instance I if for any non-erroneous scenario of instance I the policy generates a feasible schedule.

Definition 4: An instance I is *MC-schedulable* if it admits a correct on-line scheduling policy.
The MC-SCHEDULABILITY problem is to determine whether a given instance I is MC-schedulable or not. A little thought should make it clear that for deciding MC-schedulability one only needs to consider scenarios in which for each i, $p_i = P_i(\ell)$ for some ℓ. The following is obvious.

Proposition 1: If an instance I is MC-schedulable on a given processor, then I is clairvoyantly-schedulable on the same processor.

Example 2: Consider an instance I of a *dual-criticality* system: a system with $L = 2$. I is comprised of 2 jobs: job J_1 has criticality level 1 (which is the lower criticality level), and the other job has the higher criticality level 2.

$J_1 = (0, 2, 1, (1, 1))$
$J_2 = (0, 3, 2, (1, 3))$

For this example instance, any scenario in which p_1 and p_2, are no larger than 1, has criticality 1; while any scenario not of criticality 1 in which p_1 and p_2 are no larger than 1, and 3, respectively, has criticality 2. All remaining scenarios are, by definition, erroneous. It is easy to verify that this instance is clairvoyantly-schedulable.

Policy S_0, described below, is an example of an on-line scheduling policy for instance I:

S_0: Execute J_2 over [0,1]. If J_2 has no remaining execution (i.e., p_2 is revealed to be no greater than 1), then continue with scheduling J_1 over $(1, 2]$; else continue by completing scheduling J_2.

It is easy to see that policy S_0 is correct for instance I. However, S_0 is not correct if we modify the deadline of J_1 obtaining the following instance I':

$J_1 = (0, 1, 1, (1, 1))$
$J_2 = (0, 3, 2, (1, 3))$

It is easy to see that I' is clairvoyantly schedulable but not MC-schedulable.

The above example shows that there are instances that are clairvoyantly schedulable but not MC-schedulable. Indeed, this is true even if the machine upon which the on-line algorithm executes is faster than the one upon which the clairvoyant scheduler executes[1]. This is shown in the following proposition.

Proposition 2: There are dual-criticality instances that are clairvoyantly schedulable on a given processor, but that are not MC-schedulable on a processor that is less than $(1+\sqrt{5})/2$ times as fast.

Proof: Let $\sigma = (1+\sqrt{5})/2$ and consider the following instance

- $J_1 = (0, 1, 1, (1, 1))$;
- $J_2 = (0, \sigma, 2, (\sigma - 1, \sigma))$.

This system is clairvoyantly schedulable. To analyze its MC-schedulability, consider the possible policies on a higher speed-s processor. The first one starts with J_2 and runs it till $P_2(1) = (\sigma-1)/s$, and if it signals completion, schedule J_1 which then finishes latest by $(\sigma-1)/s + 1/s = \sigma/s$. This is feasible only if $\sigma/s \leq 1$, that is, $s \geq \sigma$. The other policy is simply to first schedule J_1 and then J_2, which may require a total execution time $1/s + \sigma/s$, which is feasible only if $(1+\sigma)/s \leq \sigma$, that is, $s \geq (\sigma+1)/\sigma$. Hence, if the processor has speed $s < \min\{\sigma, (\sigma+1)/\sigma\}$, neither of the possible scheduling policies is correct. Taking $\sigma = (\sigma+1)/\sigma$, that is, $\sigma = (1+\sqrt{5})/2$, implies $s \geq \sigma$. ■

III. Complexity of MC-Schedulability

In this section we investigate the complexity of MC-SCHEDULABILITY. We show that it is NP-hard in the strong sense. However, a little thought should make it clear that it is not trivial to decide if the problem belongs to NP or not. We prove that it actually belongs to NP if the number of criticality levels is bounded by a fixed constant. For the general case, in which the number of criticality levels is part of the input, we show that it belongs to the class PSPACE, leaving membership to NP as an open question. We complete the section by presenting a well-solved special case of MC-SCHEDULABILITY, in which all jobs have equal deadline.

A preliminary observation is that determining clairvoyant-schedulability has the same complexity as the ordinary scheduling problem with only 1 criticality level: verify for each criticality level $\ell = 1, \ldots, L$ if the jobs of that criticality level or higher can be scheduled to complete before their deadlines if each such job j has execution time $P_j(\ell)$. In particular this means that clairvoyant-schedulability of any instance on a fully preemptive processor platform can be verified in polynomial time. This also holds if $P_j(\ell)$ is not monotonic in ℓ.

We show that it is strongly NP-hard to determine whether a given clairvoyantly-schedulable system is also MC-schedulable upon a fully preemptive single-processor platform.

Theorem 1: MC-SCHEDULABILITY is NP-hard in the strong sense, even when all release times are identical and there are only two criticality levels.

Proof: The proof is by reduction from the strongly NP-complete problem 3-PARTITION [**?**]. In an instance I_{3P} of 3-PARTITION, we are given a set S of $3m$ positive integers $s_0, s_1, \ldots, s_{3m-1}$ and a positive integer B such that $B/4 < s_i < B/2$ for each i and $\sum_{i=0}^{3m-1} s_i = mB$. The problem is to decide whether S can be partitioned into m disjoint sets $S_0, S_1, \ldots, S_{m-1}$ such that, for $0 \leq k < m$, $\sum_{s_i \in S_k} s_i = B$.

From a given instance I_{3P} we construct an MC-SCHEDULABILITY instance I_{MC} consisting of $4m$ jobs with release time 0, which in the 4-tuple notation are:

- *3P-jobs:* For each i, $0 \leq i < 3m$, job $J_i = (0, 2mB, 2, (s_i, 2s_i))$;
- *Blocking jobs:* For each k, $0 \leq k < m$, job $J_{3m+k} = (0, 2(k+1)B, 1, (B, B))$.

Clairvoyant-schedulability can be verified easily. In each scenario of criticality level 2, the blocking jobs do not need to be scheduled, and the total execution time of the 3P-jobs is at most $\sum_{i=0}^{3m-1} P_i(2) = \sum_{i=0}^{3m-1} 2s_i = 2mB$. For each scenario of criticality level 1, each blocking job is scheduled for at most B time-units immediately preceding its deadline. The total available execution time before the common deadline $2mB$ of the 3P-jobs is at least $mB \geq \sum_{i=0}^{3m-1} P_i(1)$.

[1]This notion, of comparing the performance of an on-line algorithm executing upon a faster processor than the processor available to the clairvoyant algorithm, is formalized in the concept of *resource augmentation*; this concept is explored further in Section IV.

We show that instance I_{3P} is a YES-instance of 3-PARTITION if and only if the corresponding instance I_{MC} is MC-schedulable.

Suppose there is a feasible partition $S_0, S_1, \ldots, S_{m-1}$ for instance I_{3P}. Based on this partition, a feasible online scheduling policy for jobs in I_{MC} is as follows. For each $k = 0, 1, \ldots, m-1$, reserve, for the 3P-jobs J_i corresponding to each $s_i \in S_k$, $P_i(1) = s_i$ units in the time-interval $[2kB, (2k+1)B)$. If all jobs associated with a set S_k signal that they have completed execution in this time interval, then schedule the blocking job J_{3m+k} over the interval $[(2k+1)B, (2k+2)B)$. Else, discard job J_{3m+k} and complete the execution of the jobs that had not completed. Note that this is possible since $\sum_{i|s_i \in S_k} P_i(2) = 2B$ which is equal to the length of the interval $[2kB, (2k+2)B)$, which is now completely available for execution of 3P-jobs.

Suppose that there is no 3-partition for instance I_{3P}. Consider any online policy. Since preemption is allowed, we can assume w.l.o.g. that the policy schedules each blocking job J_{3m+k} as late possible, that is, in intervals $[(2k+1)B, 2(k+1)B)$, $k = 0, 1, \ldots, m-1$, as long as the system's scenario is in criticality level 1. Take a scenario in which each job needs either $P_i(1)$ or $P_i(2)$. Let k' be the smallest value of k for which there exists a 3P-job J_j which receives execution time p'_j, with $0 < p'_j < P_j(1)$, during the interval $[0, (2k'+1)B)$, and such that the scenario is still of criticality level 1 at time $(2k'+1)B$. Because there is no 3-partition, a scenario must exist such that k' and J_j exist. Since the system's scenario is still at criticality level 1, the online policy must schedule a blocking job at $[(2k'+1)B, 2(k'+1)B)$.

Now, let any 3P-job J_i be scheduled after time $2(k'+1)B$ report an execution time of $P_i(2)$. Clearly, any sensible online policy discards all blocking jobs J_{3m+k} with $k > k'$, and uses all the remaining available time up to the common deadline of the 3P-jobs, that is, $2mB - 2(k'+1)B$, for processing 3P-jobs. However, the total remaining processing requirement of these jobs, is $2(mB - (k'+1)B) + p'_j > 2mB - 2(k'+1)B$.

■

The question remains as to whether MC-SCHEDULABILITY belongs to the complexity class NP or not. In case the number of criticality levels L is a constant, we answer this question affirmatively; otherwise, the best we are able to show is that MC-SCHEDULABILITY $\in$ PSPACE. The proof is based on a polynomial-time checkable characterization of an online scheduling policy.

Call a scenario $(p_1, \ldots, p_n)$ *basic* if for each $j = 1, \ldots, n$ there exists $\ell_j \leq \chi_j$ such that $p_j = P_j(\ell_j)$. Call an on-line scheduling policy *basically correct* for instance I if for any basic scenario of I the policy generates a feasible schedule. We have the following.

Lemma 1: An instance is MC-schedulable if it admits a basically correct scheduling policy.

Proof: Let π be a basically correct policy for instance I. We modify π to obtain a new policy π'. Policy π' simulates π, except that when a job j does not follow a basic scenario (say, $P_j(k-1) < p_j < P_j(k)$), then π' runs π as if job j did not complete before executing $P_j(k)$ units. Whenever π prescribes the execution of a job that has already signaled completion, π' idles the processor. The resulting simulated scenario is basic, so π feasibly schedules it. Thus π' feasibly schedules the original scenario (which has the same criticality level). ■

In view of Lemma 1, any online policy can be represented as a finite-size decision tree in which each path from the root to a leaf defines the scheduling decisions for a particular scenario. More precisely, a scheduling decision encodes which job should be executed for which amount of time. The decisions are based on when the jobs complete their execution. We show the following lemma, which is crucial for significantly reducing the size of the decision tree for an optimal dynamic policy.

Lemma 2: If an instance is MC-schedulable, then there exists an optimal online scheduling policy that preempts each job j only at time points t such that at time t either some other job is released, or j has executed for exactly $P_j(i)$ units of time for some $1 \leq i \leq L$.

Proof: Consider an optimal online scheduling policy that preempts some job j after it has executed for $P_j(i) < p < P_j(i+1)$ units of time, for some $0 \leq i < L$ (with the convention that $P_j(0) = 0$). Let t be the first decision time when such a situation occurs, that is, in the optimal decision tree representation this decision is closest to the root for all paths to leaves, and assume t is not the release time of any job. Now, we modify the policy in the following way: we change the decision to preempt j after p units of time into preempting j already after $P_j(i) < p$ units of time. Furthermore, at the next decision point (or next release time) t', we simply add the remaining amount of processing $p - P_j(i)$ to the scheduled amount of processing. Clearly, this modification has to be done for each scenario affected by that change, that is, in the full subtree below the first modified decision. Notice that by preempting j earlier, we do not lose any information on job completions because by Lemma 1 we can assume the scenario is basic. If the original policy completed j no later than its deadline for all scenarios that require this, then this is still true for the modified policy. Moreover, we do not change the amount of execution for jobs that are scheduled between t and t'. Thus, if the original policy was feasible, then the modified one is feasible as well. Repeated applications of this argument completes the proof. ■

Theorem 2: The problem of deciding MC-schedulability for L criticality levels is in NP when L is a constant.

Proof: We show that an online policy can be represented and verified in polynomial time and space, if L is constant. Consider an optimal online policy and the corresponding decision tree in which each path from the root to a leaf defines the scheduling decisions for a particular scenario. By Lemma 2, we can assume that the decision times of the policy are those points in time when some job j has completed $P_j(i)$ units of time for some $1 \le i \le L$.

Let $N(n, L)$ denote the number of nodes of an optimal decision tree for an instance with n jobs and L criticality levels. We show by induction that $N(n, L) = \mathcal{O}(n^L)$. For $L = 1$, there is no decision to be made since the earliest deadline first (EDF) policy is optimal. For larger L, any optimal policy starts by executing some job j for some time $P_j(i)$. If j completes, we are left with an instance with one less job. If not, the criticality level of the scenario is increased by at least one and thus (by updating the input parameters appropriately) we are left with an instance with n jobs and (at most) $L - 1$ levels. Thus, we have the recurrence $N(n, L) \le N(n - 1, L) + N(n, L - 1) + 1$. Since by induction $N(n, L - 1) = \mathcal{O}(n^{L-1})$, we have $N(n, L) \le N(n - 1, L) + \mathcal{O}(n^{L-1})$, implying $N(n, L) = \mathcal{O}(n^L)$.

Thus, the tree has polynomial size. To verify that the policy is feasible for any possible scenario, we check for each individual path from the root to a leaf if the decisions of the policy lead to a feasible schedule for any scenario compatible with the information extracted by the algorithm. This takes polynomial time. ■

Theorem 3: The problem of deciding MC-schedulability is in PSPACE.

Proof: Consider the tree representation of an optimal online policy as in the proof of Theorem 2. Notice that we cannot store the whole tree in space that is polynomial in n when L is large. However, we can still check that such a tree exists by generating in depth-first order all paths from the root to a leaf, while making sure that the common portion of consecutive paths is consistent. It is enough to store two paths at a time. Each path requires space proportional to its depth, which is $\mathcal{O}(nL)$, and to keep track of the depth-first search a counter of size $\mathcal{O}(nL)$ is enough, because there are $\mathcal{O}(2^{nL})$ potential paths, the tree being binary. Finally, as in the proof of Theorem 2 we verify for each path that the decisions of the policy generate a valid schedule. This yields a nondeterministic algorithm for deciding MC-schedulability that uses polynomial space. The claim follows by the well-known fact that nondeterminism can be removed from the algorithm, at the cost of squaring the required space. ■

a) **Equal deadlines.:** Theorem 1 above shows that MC-SCHEDULABILITY is in general NP-hard even if release times are identical. We will now show that the special case in which all jobs have equal deadlines ($d_j = D$, $j = 1, \ldots, n$) can be solved in polynomial time. We first derive a necessary condition for such an instance I to be MC-schedulable. Consider the criticality level ℓ scenario of I in which each job J_j needs exactly $p_j = P_j(\ell)$ execution time.

Necessary condition: If I is MC-schedulable then for each ℓ, a scheduling policy exists that allocates to each job J_j with $\chi_j \geq \ell$ at least $P_j(\ell)$ execution time within time window $[r_j, D]$, i.e., the *makespan* of the scenario is at most D.

This condition is easily checked: Let $I_\ell = \{J_j \in I \mid \chi_j \geq \ell\}$ and $|I_\ell| = n_\ell$. Let (after renumbering) $J_1, J_2, \ldots, J_{n_\ell}$ denote the jobs in I_ℓ in order of non-decreasing release times: $r_1 \leq r_2 \leq \ldots \leq r_{n_\ell}$. Clearly, the makespan of I_ℓ is given by

$$C^\ell_{\max} := \max_{j=1,\ldots,n_\ell} r_j + \sum_{i=j}^{n_\ell} P_j(\ell). \tag{1}$$

The necessary condition is then verified by checking if

$$\max_{\ell=1,\ldots,L} C^\ell_{\max} \leq D. \tag{2}$$

Consider the *criticality-monotonic* (CM) on-line scheduling policy, which schedules at each time instant an available job of highest criticality.

Theorem 4: CM is correct for all for MC-schedulable instances in which all jobs have the same deadline.

Proof: We prove this by showing that the necessary condition is also sufficient. Consider any scenario of I that has criticality level ℓ. In a CM-schedule, the scheduling of jobs of criticality ℓ or higher is not effected by the presence of lower-criticality jobs, since their execution is postponed as soon as jobs in I_ℓ become available. Hence, a CM-schedule can be thought of as a schedule that minimizes the makespan of the jobs in I_ℓ. By the necessary condition, this does not exceed the common deadline D. ■

We observe that this theorem also holds when $P_j(\ell)$ is not monotonic in ℓ.

IV. Algorithms for MC scheduling

Since MC-SCHEDULABILITY is intractable even for dual-criticality instances, we concentrate here on sufficient (rather than exact) MC-schedulability conditions that can be verified in polynomial time. We study two widely-used scheduling policies that yield such sufficient conditions and compare their capabilities under the *resource augmentation metric*: the minimum speed of the processor needed for the algorithm to schedule all instances that are MC-schedulable on a unit-speed processor. We show that the second policy we present outperforms the first one in terms of the resource augmentation metric, in the sense that it needs lower-speed processors to ensure such schedulability.

Run-time support for mixed criticality. In scheduling mixed-criticality systems, the kinds of performance guarantees that can be made depend upon the forms of support that are provided by the run-time environment upon which the system is being implemented. A particularly important form of platform support is the ability to *monitor* the execution of individual jobs, i.e., being able to determine how long a particular job has been executing.

Why is such a facility useful? In essence, knowledge regarding how long individual jobs have been executing allows the system to become aware, during run-time, when the criticality level of the behavior changes from a value k to the next-higher value $k+1$, due to some job executing beyond its level-k WCET without signalling completion; this information can then be used by the run-time scheduling and dispatching algorithm to no longer execute criticality-k jobs once the transition has occurred.

In the remainder of this section, we assume that this facility to monitor the execution of individual jobs is provided by the run-time environment. We may therefore make the assumption that for each job J_j, $P_j(\ell) = P_j(\chi_j)$ for all $\ell \geq \chi_j$. That is, no job executes longer than the WCET at its own specified criticality. This is without loss of generality for any correct scheduling policy: any such policy will immediately interrupt (and no longer schedule) a job J_j if its execution time p_j exceeds $P_j(\chi_j)$, since this makes the scenario of higher criticality level than χ_j, and therefore the completion of J_j becomes irrelevant for the scenario.

A. *Reservations-based scheduling*

As stated in Section I, one straightforward approach is to map each MC job J_j into a "traditional" (i.e., non-MC) job with the same arrival time r_j and deadline d_j and processing time $p_j = P_j(\chi_j) = \max_\ell P_j(\ell)$ (by monotonicity), and determine whether the resulting collection of traditional jobs is schedulable using some preemptive single machine scheduling algorithm such as the *Earliest Deadline First* (EDF) rule[2]. This test can clearly be done in polynomial time. We will refer to mixed-criticality instances that are MC-schedulable by this test as *worst-case reservations schedulable* (WCR-schedulable) instances.

Theorem 5: If an instance is WCR-schedulable on a processor, then it is MC-schedulable on the same processor. Conversely, if an instance I with L criticality levels is MC-schedulable on a given processor, then I is WCR-schedulable on a processor that is L times as fast, and this factor is tight.

Proof: If instance I is WCR-schedulable then for each job the maximum amount of time the job may execute is reserved between its arrival time and its deadline. Hence it is MC-schedulable.

Suppose now that instance I is MC-schedulable. If we were to use a separate processor for each of the L criticality levels, then each job will receive its maximum processing time between arrival time and deadline e.g. by using EDF on the machine corresponding to its criticality level. Hence, by processer sharing, WCR-schedulability on one processor of speed L times faster follows immediately.

Finally, we show that there exist instances with L criticality levels that are MC-schedulable on a given processor, but not WCR-schedulable on a processor that is less than L times as fast.

Consider the instance I comprised of the following L jobs:

$$\begin{array}{rcl} J_1 & = & (0,1,1,(1,1,\ldots,1,1)) \\ J_2 & = & (0,1,2,(0,1,\ldots,1,1)) \\ \vdots & & \vdots \\ J_L & = & (0,1,L,(0,0,\ldots,0,1)) \end{array}$$

This instance is MC-schedulable on a unit-speed processor by the scheduling policy of assigning priority in criticality-monotonic (CM) order: $J_L, J_{L-1}, \ldots . J_2, J_1$. Any scenario $(p_1, p_2, \ldots, p_L)$ with $p_h > 0$, $h \geq 2$, and $p_j = 0$ for all $j > h$, has criticality level h, hence all jobs of lower criticality level, in particular J_1, are not obliged to meet their deadline, and job h will meet its deadline. On the other hand, in any scenario of criticality level 1, $p_2 = p_3 = \ldots = p_L = 0$ and $p_1 \in [0,1]$, hence all jobs meet their deadline.

However, WCR-schedulability requires that each job J_j is executed for $P_j(\chi_j) = 1$, $j = 1, \ldots, L$ before common deadline 1, which clearly can only be achieved on a processor with speed at least L. ■

B. *Priority-based scheduling*

We now consider another schedulability condition, OCBP-schedulability, that offers a performance guarantee (as measured by the processor speedup factor) that is superior to the performance guarantee offered by the WCR-approach. OCBP-schedulability is a constructive test: we determine off-line, before knowing the actual execution times, a total ordering of the jobs in a priority list and for each scenario execute at each moment in time the available job with the highest priority.

The priority list is constructed recursively using the approach commonly referred to in the real-time scheduling literature as the "Audsley approach" [**?**], [**?**]; it is also related to a technique introduced by Lawler [**?**]. First determine the lowest priority job: Job J_i has lowest priority if there is at least $P_i(\chi_i)$ time between its release time and its deadline available when every other job J_j is executed before J_i

[2]In fact, this approach forms the basis of current practice, as formulated in the ARINC-653 standard: each J_j is guaranteed $P_j(\chi_j)$ units 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.

for $P_j(\chi_i)$ time units (the WCET of job J_j according to the criticality level of job i). The procedure is repeatedly applied 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. If job J_i has higher priority than job J_j we write $J_i \rhd J_j$.

Because the priority of a job is based only on its own criticality level, the instance I is called *Own Criticality Based Priority 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 can simply argue that this does not mean that the instance is not MC-schedulable: Suppose that scheduling according to the fixed priority list J_1, J_2, J_3 with $\chi_2 = 1$ and $\chi_1 = \chi_3 = 2$, proves the instance to be schedulable. It may not be OCBP-schedulable since this does not take into account that J_2 does not need to be executed at all if J_1 receives execution time $p_1 > P_1(1)$.

Clearly, if a priority list exists, it can be determined in polynomial time. The following theorem shows that the OCBP-test is more powerful than the WCR-test according to the speedup criterion.

Theorem 6: If an instance is OCBP-schedulable on a processor, then it is MC-schedulable on the same processor. Conversely, if instance I with L criticality levels is MC-schedulable on a given processor, then I is OCBP-schedulable on a processor that is s_L times as fast, with s_L equal to the root of the equation $x^L = (1+x)^{L-1}$, and this factor is tight. Furthermore, it holds that $s_L = \Theta(L/\ln L)$.

Proof: We present this proof in several parts:

(i) OCBP-schedulability implies MC-schedulability.
(ii) A speedup of s_L is sufficient.
(iii) The factor of s_L is tight.
(iv) $s_L = \Theta(L/\ln L)$.

i): OCBP-schedulability implies MC-schedulability. Suppose that I is OCBP-schedulable and suppose, after renumbering jobs, that $J_1 \rhd J_2 \rhd \cdots \rhd J_n$. Notice that in every behaviour of criticality level χ_k, the criticality level of job J_k, each job J_j has $p_j \leq P_j(\chi_k)$. OCBP-schedulability of I implies that J_k can receive $P_k(\chi_k)$ units of execution before its deadline if each $J_i \in \{J_1, \ldots, J_{k-1}\}$ executes for no more than $P_i(\chi_k)$ units.

ii): A speedup of s_L is sufficient. Notice that $s_1 = 1$, and that (as one can verify using elementary calculus) $s_{L'} \geq s_L$ if $L' > L$. Let I be an instance with at most L criticality levels that is MC-schedulable on a speed-1 processor, but not OCBP-schedulable on a speed-s processor for some $s \geq s_L$, and amongst such instances let it be minimal with respect to L and the number of jobs. Suppose I has n jobs. Minimality of I implies that there is no time-instant t such that $t \notin \cup_{j=1}^{n} [r_j, d_j]$, otherwise either the jobs with deadline before t or the jobs with release time after t would comprise a smaller instance with the same property.

Claim 1: Any job in I with the latest deadline must be of criticality L.

Proof: Suppose that a job J_i with $\chi_i = h < L$ has latest deadline. Create from I an instance I_h with level h by "truncating" all jobs with criticality level greater than h to their worst-case level-h scenarios:

$$\begin{aligned} J_j = (r_j, d_j, \chi_j, (P_j(1), \ldots, P_j(L))) &\in I \rightarrow \\ J'_j &= (r_j, d_j, \min(\chi_j, h), (P_j(1), \ldots, P_j(h))) \in I_h. \end{aligned}$$

Clearly, I_h being a restricted instance of I, is MC-schedulable as well, and, by minimality of I, I_h is OCBP-schedulable on a speed-s_h processor.

That J_i has latest deadline in I but cannot be assigned lowest priority on a speed-s processor implies that the scenario with $p_j = P_j(h)$ cannot be feasibly scheduled on a speed-s processor; thus I_h is not clairvoyantly schedulable on a speed-s processor. But I_h not being clairvoyantly schedulable implies I_h not being OCBP-schedulable, and because $s \geq s_L \geq s_h$, this contradicts the OCBP-schedulability of I_h on a speed-s_h processor and completes the proof of the claim. ■

For each $\ell \in \{1, \ldots, L\}$, let $d(\ell)$ denote the latest deadline of any criticality-ℓ job in I: $d(\ell) = \max_{J_j|\chi_j=\ell} d_j$. A *work-conserving* schedule on a processor is a schedule that never leaves the processor idle if there is a job available. Consider any such a work-conserving schedule on a unit-speed processor of all jobs in I of the scenario in which $p_j = P_j(\ell)$ for all j. We define Λ_ℓ as the set of time intervals on which the processor is idle before $d(\ell)$, and λ_ℓ as the total length of this set of intervals.

Claim 2: For each ℓ and each $J_j \in I$ with $\chi_j \leq \ell$ we have $[r_j, d_j] \cap \Lambda_\ell = \emptyset$.

Proof: Observe that since $s \geq s_L \geq 1$, all idle intervals of Λ_ℓ are also idle intervals in any work-conserving schedule of I on a speed-s processor. Hence, any job J_j with $\chi_j \leq \ell$ with $[r_j, d_j] \cap \Lambda_\ell \neq \emptyset$ would meet its deadline in such a schedule if it were assigned lowest priority. Since I is assumed to be non-OCBP schedulable on a speed-s processor, this implies that $(I \setminus \{J_i\})$ is non-OCBP schedulable on a speed-s processor, contradicting the minimality of I. This completes the proof of the claim. ■

It follows that $\Lambda_L = \emptyset$ and $\lambda_L = 0$.

For each $h = 1, \ldots, L$ and $\ell = 1, \ldots, L$, let

$$c_h(\ell) = \sum_{J_j|\chi_j=h} P_j(\ell)$$

Notice that by assumption

$$\forall\ \ell\ \forall h \leq \ell : c_h(\ell) = c_h(h). \tag{3}$$

Since instance I is clairvoyantly schedulable on a unit-speed processor, clearly we must have

$$\forall\ \ell : c_\ell(\ell) \leq d(\ell) - \lambda_\ell. \tag{4}$$

But also, due to clairvoyant schedulability, the criticality-ℓ scenario, in which each job J_j with criticality $\geq \ell$ receives exactly $P_j(\ell)$ units of execution, completes by the latest deadline $d(L)$:

$$\forall \ell : \sum_{i=\ell}^{L} c_i(\ell) \leq d(L) - \lambda_\ell. \tag{5}$$

Instance I is not OCBP-schedulable on a speed-s processor, which translated in terms of the introduced notation is:

$$\forall \ell : \sum_{i=1}^{L} c_i(\ell) > s(d(\ell) - \lambda_\ell). \tag{6}$$

Hence, for each ℓ,

$$\begin{aligned}
s(d(\ell) - \lambda_\ell) &< \sum_{i=1}^{\ell-1} c_i(\ell) + \sum_{i=\ell}^{L} c_i(\ell) \\
&= \sum_{i=1}^{\ell-1} c_i(i) + \sum_{i=\ell}^{L} c_i(\ell) \quad \text{(by (3))} \\
&\leq \sum_{i=1}^{\ell-1} (d(i) - \lambda_i) + (d(L) - \lambda_\ell) \quad \text{(by (4) and (5))} \\
&\leq \sum_{i=1}^{\ell-1} (d(i) - \lambda_i) + d(L).
\end{aligned}$$

Therefore, for all $\ell = 1, \ldots, L$,

$$s < \frac{d(L) + \sum_{i=1}^{\ell-1}(d(i) - \lambda_i)}{d(\ell) - \lambda_\ell}$$

Using notation $\delta_\ell = d(\ell) - \lambda_\ell$ (hence $\delta_L = d(L)$ since $\lambda_L = 0$) this yields

$$s < \min_{\ell=1,\dots,L} \frac{\delta_L + \sum_{i=1}^{\ell-1} \delta_i}{\delta_\ell} \tag{7}$$

The minimum is maximized if all L terms are equal. Let x be this maximum value. Then for all $\ell = 1, \dots, L$,

$$x = \frac{\delta_L + \delta_1 + \delta_2 + \cdots + \delta_{\ell-1}}{\delta_\ell} = \frac{x\delta_{\ell-1} + \delta_{\ell-1}}{\delta_\ell} = \left(\frac{1+x}{\delta_\ell}\right)\delta_{\ell-1}.$$

Hence,

$$\delta_\ell = \left(\frac{1+x}{x}\right)\delta_{\ell-1} \ \forall \ell = 1, \dots, L \quad \text{which implies} \quad \delta_L = \left(\frac{1+x}{x}\right)^{L-1}\delta_1 .$$

Since, in particular, $x = \frac{\delta_L}{\delta_1}$, we have

$$x = \left(\frac{1+x}{x}\right)^{L-1},$$

which concludes the proof that a speedup of s_L is sufficient.

iii): The factor of s_L is tight. We now show that the factor s_L is tight by giving instances with L criticality levels that are MC-schedulable on a unit-speed processor, but not OCBP-schedulable on a processor that is less than s_L times as fast.

Consider the following instance consisting of $2L - 1$ jobs:

- $J_1 = (0, d_1 = \sigma_1 = 1, 1, (\overbrace{1, 1, \dots, 1}^{L \text{ times}}))$.
- For each i, $2 \le i \le L$, there are two jobs:
  - $J_{2(i-1)} = (0, \sigma_{i-1}, i, (\overbrace{0, 0, \dots, 0}^{(i-1) \text{ times}}, \underbrace{\sigma_{i-1}, \dots, \sigma_{i-1}}_{L-(i-1) \text{ times}}))$
  - $J_{2i-1} = (0, \sigma_i, i, (\underbrace{\sigma_i - \sigma_{i-1}, \dots, \sigma_i - \sigma_{i-1}}_{L \text{ times}}))$, where $\sigma_i > \sigma_{i-1}$.

This instance is MC-schedulable by the following policy. Assign greatest priority to the jobs J_{2i} in reverse order of their indices: $J_{2L}, J_{2(L-1)}, \dots, J_2$. Consider the scenario in which $p_{2h} > 0$, $h \ge 1$, and $p_{2j} = 0$, $j > h$. Then we execute $J_{2h}, J_{2h+1}, J_{2h+3}, \dots, J_{2L+1}$ in this order; it is evident that each of them completes by its deadline.

For job J_{2h-1}, $h = 1, \dots, L$ to be assigned lowest priority in an OCBP-schedule, we would need a speedup factor s of the processor such that

$$\frac{(\sigma_L - \sigma_{L-1}) + (\sigma_{L-1} - \sigma_{L-2}) + \cdots + (\sigma_2 - \sigma_1) + \sigma_1 + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{s} = \frac{\sigma_L + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{s} \le \sigma_h.$$

Hence, for all $h = 1, \dots, L$, it requires

$$s \ge \frac{\sigma_L + (1 + \sigma_2 + \cdots + \sigma_{j-1})}{\sigma_h}.$$

We refer to the end of part (ii) of the proof to show that the right hand side is maximized for the root of the equation $x^L = (1+x)^{L-1}$.

iv): $\mathbf{s_L = \Theta(L/\ln L)}$. Rewrite the equation as $x = (1+1/x)^{L-1}$ and let x^* be its largest real root. The left hand side (resp., r.h.s.) is increasing (resp., decreasing) in x. The l.h.s. is larger (resp., smaller) than the r.h.s. precisely when $x > x^*$ (resp., $x < x^*$). So if substituting (say) $f(L)$ in place of x gives a l.h.s. larger (resp., smaller) than the r.h.s., it means that $f(L)$ is an upper (resp., lower) bound on x^*.

Substituting $2(L-1)/\ln L$ in place of x, we get for the r.h.s.:

$$(1+1/x)^{L-1} \leq e^{(L-1)/x} = e^{(L-1)(\ln L)/2(L-1)} = L^{1/2}$$

(where we have used $1 + y \leq e^y$). The l.h.s. becomes instead $2(L-1)/\ln L$, which is larger than the r.h.s. for all $L \geq 2$. So $x^* \leq 2(L-1)/\ln L$ for all $L \geq 2$.

Substituting $(L-1)/(2\ln L)$ in place of x, we get for the r.h.s.:

$$(1+1/x)^{L-1} \geq e^{(L-1)\frac{1}{2x}} = L$$

(where we have used $1+2y \geq e^y$ for all $y \in [0, 1.2]$, and assumed $L \geq 3$). The l.h.s. becomes instead $(L-1)/(2\ln L)$, which is smaller than the r.h.s. for all $L \geq 2$. So $x^* \geq (L-1)/(2\ln L)$ for all $L \geq 3$.

■

We note that for $L = 2$ in the above theorem, $s_2 = (1+\sqrt{5})/2$ is equal to the golden ratio ϕ; thus the result is a true generalization of an earlier result in [?]. In general, $s_L = \Theta(L/\ln L)$; hence, this priority-based scheduling approach asymptotically improves on the worst-case reservations-based approach by a factor of $\Theta(\ln L)$ from the perspective of processor speedup factors.

Notice that the proof of the speedup bound for OCBP-schedulability in Theorem 6 only uses the clairvoyant-schedulability of the instance, which is a weaker condition than MC-schedulability (recall Proposition 1). Moreover, Proposition 1 shows that it is not possible to get an improved test if the proof of its speedup bound is based on clairvoyant-schedulability alone.

Nevertheless, the question remains if a test other than OCBP can test MC-schedulability within a smaller speedup bound. We do not give a full answer to this question. However, we can rule out *fixed-priority policies*, that is, policies which execute the jobs in some ordering fixed before execution. This ordering is not adapted during execution, except that we do not execute jobs of criticality level $i < h$ after a scenario was revealed to be a level-h scenario. Such a policy admits a simple representation as a sequence of jobs and we say that an instance I is Π-schedulable if there exists an ordering of jobs Π that is feasible for for any non-erroneous scenario.

The following result shows that OCBP is best possible among fixed-priority policies.

Theorem 7: There exist instances with L criticality levels that are clairvoyantly-schedulable, but that are not Π-schedulable for any fixed priority policy Π on a processor that is less that s_L times as fast, with s_L being the root of the equation $x^L = (1+x)^{L-1}$.

Proof: Consider an instance with L criticality levels and L jobs:

$$J_1 : (0, 1, 1, (\overbrace{1, 1, \ldots, 1}^{L \text{ times}})),$$

and, for each $i = 2, \ldots, L$,

$$J_i : (0, \sigma_i, i, (\overbrace{\sigma_i - \sigma_{i-1}, \ldots, \sigma_i - \sigma_{i-1}}^{i-1 \text{ times}}, \overbrace{\sigma_i, \ldots, \sigma_i}^{L-i+1 \text{ times}})),$$

where σ_i will be specified later and satisfies $\sigma_{i-1} < \sigma_i$.

For $L = 3$ we have the following example:

$$\begin{array}{lllllll} J_1 : & (0, & 1, & 1, & (1, & 1, & 1)) \\ J_2 : & (0, & \sigma_2, & 2, & (\sigma_2 - 1, & \sigma_2, & \sigma_2)) \\ J_3 : & (0, & \sigma_3, & 3, & (\sigma_3 - \sigma_2, & \sigma_3 - \sigma_2, & \sigma_3)). \end{array}$$

The system is clairvoyantly schedulable as, for each scenario of level i and for each job J_j, $j \geq i$, $\sum_{\ell=i}^{j} P_\ell(i) = \sigma_j$. It follows that a schedule that executes job J_i in the interval $[0, \sigma_i]$ and each job J_j, $j > i$, in the interval $[\sigma_{j-1}, \sigma_j]$ is feasible.

We now show that the system is Π-schedulable for a s-speed machine only if $s \geq s_L$ where s_L is the positive real-valued solution of the equation

$$x^L = (x+1)^{L-1}.$$

Each fixed priority work-conserving policy is a sequence of jobs. Let us consider a sequence where the last scheduled job is J_i and a level i scenario. In this case the overall execution time is $\sum_{j=1}^{L} P_j(i) = \sigma_L + \sum_{j=1}^{i-1} \sigma_j$. Hence the schedule is feasible for a s-speed machine if and only if:

$$s\sigma_i \geq \sigma_L + \sum_{j=1}^{i-1} \sigma_j.$$

By using the same arguments for each possible schedule, it follows that a fixed priority policy Π system is correct for a s-speed machine if and only if

$$s \geq \min_{1 \leq i \leq L} \left\{ \frac{\sigma_L + \sum_{j=1}^{i-1} \sigma_j}{\sigma_i} \right\}.$$

As we showed in the proof of Theorem 6, s_L is the maximum value of s' satisfying the inequality:

$$\min_{1 \leq i \leq L} \left\{ \frac{\sigma_L + \sum_{j=1}^{i-1} \sigma_j}{\sigma_i} \right\} \geq s',$$

hence the system is Π-schedulable for a s-speed machine if and only if $s \geq s_L$. ■

V. MC-SCHEDULABILITY WITH BOUNDED WCET

Although Theorems 6 and 7 above provide tight bounds on the performance of OCBP and similar algorithms, better performance than is implied by these bounds may be possible when further restrictions are placed on the kinds of instances that could need to be scheduled. We illustrate this phenomenon in this section, by analyzing the OCBP scheduling policy upon dual-criticality instances when the overall WCET at level two of criticality-two jobs is a priori known to be bounded by a constant times the level-one WCET's of these jobs. That is, let $c_1 = \sum_{j|\chi_j=1} P_j(1)$ denote the cumulative WCET for jobs with criticality level 1, and let $c_2(1) = \sum_{j|\chi_j=2} P_j(1)$ and $c_2(2) = \sum_{j|\chi_j=2} P_j(2)$ denote the cumulative WCETs for jobs with criticality level 2 at levels 1 and 2, respectively. We consider the situation that $c_2(2) \leq \beta c_2(1)$ for a certain constant β, whose value is known.

The following theorem shows that if $\beta < 1 + \phi$, $\phi = (1+\sqrt{5})/2$, the speed required by OCBP to give a necessary condition for MC-schedulability is smaller than the bound of Theorem 6.

Theorem 8: If an instance I with 2 criticality levels is MC-schedulable on a given processor and $c_2(2) \leq \beta c_2(1)$ for a certain constant β, then I is OCBP-schedulable on a processor that is s times as fast, where s is given by

$$s = \begin{cases} \frac{\sqrt{\beta^2+\beta+1}+\beta}{\beta+1} & \text{if } \ \beta \leq 1+\phi \\ \phi & \text{if } \ \beta > 1+\phi. \end{cases}$$

Proof: Let I be a minimal instance that is MC-schedulable on a given processor and not OCBP-schedulable on a processor that is s times as fast for some $s > 1$. We show that

$$s < \begin{cases} \frac{\sqrt{\beta^2+\beta+1}+\beta}{\beta+1} & \text{if } \ \beta \leq 1+\phi \\ \phi & \text{if } \ \beta > 1+\phi. \end{cases}$$

Let d_1 and d_2 denote the latest deadlines at level 1 and 2, respectively, and let j_1 and j_2 be the jobs with deadlines d_1 and d_2. Recall from the first claim in the proof of Theorem 6 that the jobs in I with latest deadline must be of criticality 2. Define λ_1 as in the proof of Theorem 6. That is, consider any work-conserving schedule on a unit-speed processor of all jobs in I of the scenario in which $p_j = P_j(1)$ for all j: λ_1 is defined to be the total amount of time during which the processor is idle before d_1.

Since I is MC-schedulable then c_1 cannot exceed d_1 in any criticality 1 scenario. Moreover, in scenarios where all jobs execute for their WCET at criticality 1, $c_1 + c_2(1)$ cannot exceed d_2 and in scenarios where all jobs execute for their WCET at criticality 2, $c_2(2)$ cannot exceed d_2. Hence, by instantiating Inequality 4 for $\ell = 1$ we have

$$c_1 \leq (d_1 - \lambda_1) \tag{8}$$

By instantiating Inequality 5 for $\ell = 1$ and noting that $(d_2 - \lambda_1) \leq d_2$ we have

$$c_1 + c_2(1) \leq d_2 \tag{9}$$

By instantiating Inequality 4 for $\ell = 2$ and noting that $\lambda_2 = 0$ for dual-criticality systems ($L = 2$) we have

$$c_2(2) \leq d_2. \tag{10}$$

Since I is not OCBP-schedulable on a speed-s processor, j_1 and j_2 cannot be the lowest priority jobs on such a processor. Hence by instantiating Inequality 6 for $\ell = 1$ we have

$$c_1 + c_2(1) > s(d_1 - \lambda_1) \tag{11}$$

Similarly, by instantiating inequality (6) for $\ell = 2$ and noting that $\lambda_2 = 0$ for dual-criticality systems ($L = 2$) we have

$$c_1 + c_2(2) > sd_2. \tag{12}$$

As $c_2(2) \leq \beta c_2(1)$, inequality (12) implies

$$c_1 + \beta c_2(1) > sd_2. \tag{13}$$

We analyze the following two cases.

Case i): $\beta c_2(1) \leq d_2$.

In this case we have

$$\frac{\beta^2 c_2(1)}{\beta+1} \leq \frac{\beta d_2}{\beta+1}. \tag{14}$$

Multiplying (8) by $\frac{1}{\beta+1}$, (9) by $\frac{\beta}{\beta+1}$ and adding them to (14) yields:

$$c_1 + \beta c_2(1) \leq \frac{1}{\beta+1}(d_1 - \lambda_1) + \frac{2\beta}{\beta+1} d_2.$$

This together with (13) implies

$$sd_2 < \frac{1}{\beta+1}(d_1 - \lambda_1) + \frac{2\beta}{\beta+1} d_2.$$

which rewritten using $x = \frac{(d_1 - \lambda_1)}{d_2}$ is

$$s < \frac{1}{\beta+1} x + \frac{2\beta}{\beta+1}.$$

Fig. 1. Upper bounds for cases *(i)* and *(ii)*.

By (9) and (11) we have $s(d_1 - \lambda_1) < d_2$ and hence $xsd_2 < d_2$, implying that $s < \frac{1}{x}$. Hence

$$s < \min\left\{\frac{1}{\beta+1}x + \frac{2\beta}{\beta+1}, \frac{1}{x}\right\}.$$

The largest value of s satisfying the above inequality is

$$s = \frac{\sqrt{\beta^2+\beta+1}+\beta}{\beta+1}.$$

Case *ii)*: $\beta c_2(1) > d_2$.

Inequality (9) and $c_2(1) > \frac{d_2}{\beta}$ imply $c_1 < d_2 - \frac{d_2}{\beta}$ which together with inequality (12) implies

$$d_2 - \frac{d_2}{\beta} + c_2(2) > sd_2$$

As $c_2(2) \leq d_2$, we have $d_2 - \frac{d_2}{\beta} + d_2 > sd_2$ and hence

$$s < 2 - \frac{1}{\beta}.$$

See Figure 1 for a plot of the upper bounds given by cases *i)* and *ii)* as a function of β. The upper bounds are monotonically non-increasing with β and both of them are equal to ϕ for $\beta = 1 + \phi$. Moreover, the upper bound given by case *i)* is greater that the upper bound given by case (ii) for each $\beta < 1 + \phi$. Hence, by taking the worst of the above cases, for $\beta < 1 + \phi$ we obtain a better bound for OCBP. In detail:

$$s < \begin{cases} \frac{\sqrt{\beta^2+\beta+1}+\beta}{\beta+1} & \text{if} \quad \beta \leq 1+\phi \\ \phi & \text{if} \quad \beta > 1+\phi. \end{cases}$$

■

VI. Conclusions

As safety-critical real-time embedded systems become larger and more complex, the cost and complexity of obtaining certification for such systems is becoming a serious concern [**?**]. In mixed-criticality systems, these certification considerations give rise to fundamental new resource allocation and scheduling challenges; results from conventional real-time scheduling theory do not appear to be adequate for dealing with these challenges.

The research described in this document represents our efforts at devising new formal models, analysis, and algorithms for mixed-criticality scheduling. We have described a model for representing a simple kind of mixed-criticality system – those that can be represented as collections of independent jobs executing upon a single preemptable processor. We have studied the complexity of MC-SCHEDULABILITY: determining whether a given mixed-criticality instance, specified according to our model, is schedulable or not. We show that this is an NP-hard problem. For instances with a fixed (constant) number of criticality levels, it is NP-complete. We have also shown that the problem is in PSPACE for instances with arbitrarily many distinct criticality levels, but leave establishing the precise complexity as an open problem.

We gave an example of a special case of MC-SCHEDULABILITY that is solvable in polynomial time: when all the jobs have the same deadline. It is left for future research to investigate the boundaries between hardness and well-solvability further.

We have also provided a framework for comparing the powerfulness of MC-schedulability analysis algorithms via the metric of the speed-up factor. We have proposed a mixed-criticality scheduling algorithm (OCBP), and have quantified its performance guarantees according to the speedup-factor. We conjecture that the OCBP-schedulability test is the best possible in terms of the speedup factor metric, but we have only been able to prove this within a restricted class of scheduling policies.