Scheduling real-time mixed-criticality jobs

Sanjoy Baruah[1,⋆], Vincenzo Bonifaci[2], Gianlorenzo D'Angelo[3], Haohan Li[1], Alberto Marchetti-Spaccamela[4], Nicole Megow[2], and Leen Stougie[5]

[1] The University of North Carolina, Chapel Hill, NC, USA.
{baruah,lihaohan}@cs.unc.edu.
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany.
{bonifaci,nmegow}@mpi-inf.mpg.de
[3] Università degli Studi dell'Aquila, Monteluco di Roio, Italy.
gianlorenzo.dangelo@univaq.it
[4] Sapienza University of Rome, Rome, Italy. alberto@dis.uniroma1.it
[5] Vrije Universiteit, and CWI, Amsterdam, The Netherlands. lstougie@feweb.vu.nl

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 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 are tight for these two techniques.

1 Introduction

Due to cost and increased chip computational power, 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* 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 [3]. Recognizing these challenges,

⋆ 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.

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 will be 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 are 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.
- 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 by civilian Certification Authorities (CA's), which tend to be very conservative concerning the safety requirements. These CA's are not interested in the mission-critical functionalities, which must be validated by the clients and the vendor-manufacturer. The latter are also interested in the level 2 functionalities, but typically to standards that are less rigorous than the ones used by the civilian CA's. As such, we may consider the level 2 functionalities as a subset of the level 1 functionalities.

The difference in certification requirements is expressed by different *Worst-Case Execution Times* (WCET) for the execution of any real-time code depending on the considered critical level. In fact, each CA has its own rules, tools, etc., for determining the value of the WCET. With reference to the previous example, the WCET of the same piece of code of a flight critical functionality has two values: one lower value that express the WCET if we are considering all mission critical functions and one higher value if we restrict to all flight-critical functions. On the other side, a level 1 functionality has only one WCET.

We refer to [5] for further explications and motivations for modeling this certification requirement process. We restrict here to giving 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. J_1 is characterized by two WCETs: at level 1 its WCET is $P_1(1)$ and at level 2 its WCET is $P_1(2)$ (where $P_1(1) \leq P_1(2)$); J_2 is characterized by only one WCET $P_2(1)$.

Suppose that $P_1(1) = 3$, $P_1(2) = 5$ and $P_2(1) = 6$. 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 CA responsible for mission-critical certification determines that J_1 completes latest by time-instant 3, and J_2 by time-instant 9. Thus they both complete by their deadlines, and the system passes certification.

Note that by scheduling first J_2 and then J_1 we do not meet the requirements of the flight critical functionalities. In fact in this case the execution of J_1 could start at time 6 and therefore the job does not complete by its deadline if we assume its WCET of 5.

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

On the other side, suppose the deadline of J_2 would change to 8, then neither scheduling J_1 before J_2 nor scheduling J_2 before J_1 can be certified. In this case, scheduling J_1 before J_2 can result in a completion time of J_2 at time 9 greater than J_2's deadline. □

In Section 2, we present the model for representing mixed-criticality real-time systems, which has been proposed in [4, 5]. This mixed-criticality (MC) scheduling 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.

In previous papers [4, 5], the problem to decide schedulability of a given MC system was conjectured to be NP-hard, but a proof was never given. We do so here in Section 3. However, the exact complexity of the problem remains open, since it is not clear if 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 a special case.

In Section 4 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* (cf. resource augmentation in performance analysis of approximation algorithms, as initiated in [7]). The results here extend the results in [5], who considered the techniques for dual-criticality systems, i.e., in which there are only two different criticality levels. Our results improve the results in [4], where also the techniques for L criticality levels are studied. Moreover, we prove here that our results are tight.

2 Model and definitions

We consider a mixed-criticality (MC) system with L criticality levels, for some L. A *job* in an 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 usually represent it by $(P_j(1), \ldots, P_j(L))$.

It is natural to assume $P_j(\ell)$ to be monotonically nondecreasing for increasing ℓ. This we will do throughout, and mention if the assumption can be dropped where possible. 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 allow jobs to be *preempted* by the machine.

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 shortly 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 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 of a *dual-criticality* system: a system with $L = 2$. Consider an 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.

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

For this example instance, any scenario in which p_1, p_2, p_3, and p_4 are no larger than 1, 2, 1, and 1, respectively, has criticality 1; while any scenario not of criticality 1 in which p_1, p_2, p_3, and p_4 are no larger than 2, 2, 1, and 2, 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_1 over [0,1]. If J_1 has remaining execution (i.e., p_1 is revealed to be greater than 1), then continue with scheduling policy S_1 below; else, continue with executing scheduling policy S_2 below.
S_1: Execute J_1 over (1,2], J_3 over (2,3], and J_4 over (3,5].
S_2: Execute J_2 over (1,3], J_3 over (3,4], and J_4 over (4,5].

Scheduling policy S_0 is however not correct for I, as can be seen by considering the schedule that is generated on the scenario $(1, 2, 1, 2)$. This particular scenario has criticality 2, since $p_4 = 2 > P_4(1) = 1$. Hence, a correct schedule would need to complete jobs J_1, J_3 and J_4 by their deadlines. However, the schedule generated by S_0 has executed J_4 for only one unit before its deadline. In fact, it turns out that instance I is not MC-schedulable.

3 Complexity of MC-Schedulability

In this section we investigate the complexity of the MC-SCHEDULABILITY problem. 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.

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 [6]. 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 $0 \leq k < m$, $\sum_{s_i \in S_k} s_i = B$.

We give here just the polynomial transformation and defer the rest of the proof to a full version of the paper. From a given instance I_{3P} we construct an MC-SCHEDU-LABILITY 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 $J_{3m+k} = (0, 2(k+1)B, 1, (B, B))$.

□

The question remains if MC-SCHEDULABILITY actually belongs to the complexity class NP. In case the number of criticality levels L is a constant, we answer this question affirmatively. The proof is based on a polynomial-time checkable characterization of an online scheduling policy.

Theorem 2. MC-SCHEDULABILITY *for L criticality levels is in* NP *for any fixed L, and in* PSPACE *when L is part of the input.*

Equal deadlines. Theorem 1 above shows that the problem is in general NP-hard even if release times are identical. On the other hand, we show here 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 3. *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. □

Notice that this theorem also holds when $P_j(\ell)$ is not monotonic in ℓ. Some other well-solved sub-problems will be presented in the full version of the paper.

4 MC-schedulability testing using resource augmentation

Since MC-SCHEDULABILITY is intractable even for dual-criticality instances, we concentrate here on sufficient MC-schedulability conditions that can be verified in polynomial time. We study two such scheduling policies that yield such sufficient conditions and compare their strength under augmenting the speed of the machine or server. Taking the required speed to give a necessary condition for MC-schedulability as a measure of performance quality, the second policy we present outperforms the former one.

We make here 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.

As stated in Section 1, 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. 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. The speed-up factor in the following theorem has been proved in [4]. We complement it by proving tightness.

Theorem 4. *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.*

We now present another schedulability condition that can also be tested in polynomial time, but offers a performance guarantee (as measured by the processor speedup factor) that is superior to the performance guarantee offered by the WCR-approach.

In this algorithm 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" [1,2]; it is also related to a technique introduced by Lawler [8]. 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 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.

It turns out that the OCBP-test is more powerful than the WCR-test according to the speedup criterion.

Theorem 5. *If an instance is OCBP-schedulable on a given 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. It holds $s_L = \Theta(L/\ln L)$.*

Proof. We prove here the critical part of the claim: that a speedup of s_L is sufficient. The proof that OCBP-schedulability implies MC-schedulability has been given in [4].

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 $t \notin \cup_{j=1}^{n}[r_j, d_j]$, 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. 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:

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

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. □

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. 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. □

As a corollary, $\Lambda_L = \emptyset$ 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

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

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

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

Hence,

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

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

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

which concludes the proof. □

We note that for $L = 2$ in the above theorem, $s_2 = (1 + \sqrt{5})/2$, the golden ratio; thus the result is a true generalization of earlier results in [5]. In general, $s_L = \Theta(L/\ln L)$; hence, this priority-based scheduling approach asymptotically improves on the 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 5 only uses the clairvoyant-schedulability of the instance, which is a weaker condition than MC-schedulability (recall Proposition 1). The following claim shows that it is not possible to get an improved test if the proof of its speedup bound is based on clairvoyant-schedulability alone.

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. 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 $s \geq \sigma$. □

Nevertheless, it remains the question 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 time. 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. The following result shows that OCBP is best possible among fixed-priority policies.

Theorem 6. *There exist MC-instances with L criticality levels that are MC-schedulable, but that are not Π-schedulable for any fixed priority policy Π on a processor that is s_L times as fast, with s_L being the root of the equation $x^L = (1 + x)^{L-1}$.*

References

1. N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical Report YCS 164, Department of Computer Science, University of York, England, 1991.
2. N. C. Audsley. *Flexible Scheduling in Hard-Real-Time Systems.* PhD thesis, Department of Computer Science, University of York, 1993.
3. J. Barhorst, T. Belote, P. Binns, J. Hoffman, J. Paunicka, P. Sarathy, J. S. P. Stanfill, D. Stuart, and R. Urzi. White paper: A research agenda for mixed-criticality systems, April 2009. Available at `http://www.cse.wustl.edu/~cdgill/CPSWEEK09_MCAR/`.
4. S. Baruah, H. Li, and L. Stougie. Mixed-criticality scheduling: improved resource-augmentation results. In *Proc. of the 25th ISCA International Conference on Computers and their Applications.* To appear, 2010.
5. S. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. of the 16th IEEE Real-Time Technology and Applications Symposium*, Stockholm, Sweden, 2010.
6. M. Garey and D. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W. H. Freeman and company, NY, 1979.
7. B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.
8. E. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.