Mixed-criticality scheduling

S. K. Baruah * V. Bonifaci † G. D'Angelo ‡ H. Li *

A. Marchetti-Spaccamela § N. Megow † L. Stougie (Speaker) ¶

There is an increasing trend in embedded systems towards implementing multiple functionalities upon a single shared computing platform. This can force tasks of different criticality to share a processor and interfere with each other. These *mixed-criticality* (MC) systems are the focus of our research. We consider the scheduling of finite collections of jobs to be executed on a single machine (processor), allowing preemption.

A job in an MC system with L criticality levels is characterized by a 4-tuple of parameters: $J_j = (r_j, d_j, \chi_j, c_j)$, where r_j is the release time, d_j is the deadline ($d_j \geq r_j$), $\chi_j \in \{1, \ldots, L\}$ is the *criticality level* of the job and c_j is an L-tuple $(c_j(1), \ldots, c_j(L))$ representing the *worst-case execution times* (WCET) of job J_j at level $1, \ldots, L$, respectively. Each job J_j in a collection $J_1, \ldots, J_n$ should receive execution time C_j within time window $[r_j, d_j]$. The value of C_j is not known but is discovered by executing job J_j until it signals completion. A collection of realized values $(C_1, C_2, \ldots, C_n)$ is called a scenario. The criticality level of a scenario $(C_1, \ldots, C_n)$ is defined as the smallest integer ℓ such that $C_j \leq c_j(\ell)$, $\ell = 1, \ldots, L$. (We only consider scenarios where such an ℓ exists.) A schedule for a scenario $(C_1, \ldots, C_n)$ of criticality ℓ is *feasible* if every job J_j with $\chi_j \geq \ell$ receives execution time C_j during its time window $[r_j, d_j]$. Notice the crucial aspect of this model that, in a scenario of level ℓ, it is necessary to guarantee only that jobs of criticality at least ℓ are completed before their deadlines. In other words, once a scenario is known to be of level ℓ, the jobs of criticality $1, \ldots, \ell - 1$ can safely be dropped. Throughout we will assume that $c_j(\ell) \geq c_j(k)$ if $\ell > k$ and that for all j, $c_j(\ell) = c_j(\chi_j)$ for all $\ell > \chi_j$.

A *clairvoyant* scheduling policy knows the scenario of I, i.e., $(C_1, \ldots, C_n)$, prior to determining a schedule for I. We call an instance I *clairvoyantly-schedulable* if for each scenario of I there exists a feasible schedule.

By contrast, an *on-line* scheduling policy discovers the value of C_j only by executing J_j until it signals completion. In particular, the criticality level of the scenario becomes known only by executing jobs. An on-line scheduling policy is *correct* for instance I if for any scenario of instance I the policy generates a feasible schedule.

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. It is easy to see that for deciding MC-schedulability one only

* baruah@cs.unc.edu, lihaohan@cs.unc.edu. University of North Carolina, USA.

† bonifaci@mpi-inf.mpg.de, nmegow@mpi-inf.mpg.de. Max-Planck-Institut für Informatik, Saarbrücken, Germany.

‡ gianlorenzo.dangelo@univaq.it. University of L'Aquila, Italy.

§ alberto@dis.uniroma1.it. Sapienza University of Rome, Italy.

¶ lstougie@feweb.vu.nl. Vrije Universiteit Amsterdam & CWI, the Netherlands.

needs to consider scenarios in which for each i, $C_i = c_i(\ell)$ for some ℓ.

Example. Consider an instance I of a *dual-criticality* system: $L = 2$. I has 2 jobs:

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

Here, any scenario in which C_1 and C_2 are no larger than 1, has criticality 1; all other scenarios we consider have criticality 2. It is easy to verify that I is clairvoyantly-schedulable. The following describes 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., C_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.

With respect to complexity we prove that MC-SCHEDULABILITY is strongly NP-hard even if $L = 2$. We do not know if the problem belongs to NP. It does belong to PSPACE. For L constant the problem is in NP, hence NP-complete. Certain subcases are polynomial time solvable, for instance the case that all jobs have equal deadlines.

Since MC-SCHEDULABILITY is intractable 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.

The first, straightforward, approach is to map each MC job J_j into a "traditional" job with the same arrival time r_j and deadline d_j and processing time $c_j = c_j(\chi_j) = \max_\ell c_j(\ell)$ (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.

Theorem 1. *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.*

The second approach is a *fixed priority policy*: Off-line, before the actual execution times are known, a priority list of the jobs is determined and at each moment in time the available job with the highest priority is scheduled. 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 [6]. First determine the lowest priority job: Job J_i has lowest priority if there is at least $c_i(\chi_i)$ time between r_j and d_j its release time and its deadline available when every other job J_j is executed before J_i for $c_j(\chi_i)$ 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.

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. Clearly, if a priority list exists, it can be determined in polynomial time.

Theorem 2. *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 $s_L = \Theta(L/\ln L)$.*

We note that for $L = 2$ in the above theorem, $s_2 = (1+\sqrt{5})/2$ is equal to the golden ratio ϕ. We show that under fixed priority policies OCBP is in a sense best possible, by proving that instances with L criticality levels exist, that are clairvoyantly schedulable, but not Π-schedulable for any fixed priority policy Π on a machine that is less that s_L times as fast, with s_L being the root of the equation $x^L = (1+x)^{L-1}$.

Related work. The mixed-criticality model presented here has first been proposed and analyzed by Baruah, Li and Stougie [4]. Most of the results presented appear in Baruah et al. [3]. The mixed-criticality model has been extended to task systems by Li and Baruah [7] and by Bonifaci et al. [5].

References

[1] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The 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] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. In P. Hliněný and A. Kučera, editors, *Proc. 35th Symp. on Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 90–101. Springer, 2010.

[4] S. K. Baruah, H. Li, and L. Stougie. Towards the design of certifiable mixed-criticality systems. In *Proc. 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22. IEEE, 2010.

[5] V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Mixed-criticality scheduling of sporadic task systems. Submitted to *10th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2011.

[6] E. L. Lawler. Optimal sequencing of a single machine subject to precedence constraints. *Management Science*, 19(5):544–546, 1973.

[7] H. Li and S. K. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *Proc. 16th IEEE Real-Time Systems Symposium*, pages 183–192. IEEE, 2010.