A Note on Some Open Problems in Mixed-Criticality Scheduling

Pontus Ekberg and Wang Yi
Uppsala University, Sweden
Email: {pontus.ekberg | yi}@it.uu.se

In this note we list a few open problems that, despite being foundational for mixed-criticality (MC) scheduling theory, have received little or no attention in the existing literature. We also present a couple of (unpublished) claims related to the listed problems. The latter should, of course, be viewed with a healthy dose of skepticism.

We mainly consider the MC sporadic task model that is commonly used in the literature (sometimes called the "Vestal model"), see, e.g., [3] for a concise description of its semantics. To the best of our knowledge, all the listed problems remain open for all common variations of the system model. For example, they are open regardless of whether we assume (i) implicit, constrained or arbitrary deadlines, (ii) K criticality levels for any constant or non-constant $K \geqslant 2$, (iii) a uniprocessor or multiprocessor platform, (iv) preemptive or non-preemptive scheduling, or (v) any common subclass such as task sets with harmonic periods.

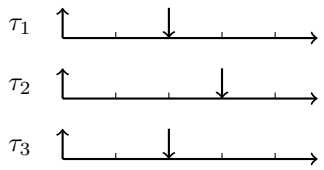
In the remainder of this note we assume a system with constrained deadlines, two criticality levels (called LO and HI) and a preemptive uniprocessor. It is also assumed that all schedulers are online (i.e., non-clairvoyant), like any real scheduler must be. In particular, a scheduler cannot know the actual execution time of a job except by executing it until completion, and it cannot know the exact release patterns of the sporadic tasks beforehand (it only knows about the *minimum* release separations).

Let a *job sequence* be a static sequence of unbounded length that specifies all the release times of jobs in one execution of the system. While the release times are fixed in a given job sequence, execution times are still unknown. For non-MC sporadic task systems on preemptive uniprocessors, it is known [1] that a particular job sequence generated by that task system—the synchronous arrival sequence (SAS)—is a guaranteed worst case, in the sense that if the SAS is schedulable, then so is every other sequence of jobs that the system can generate. This fact greatly simplifies feasibility analysis of such systems. Unfortunately, this property does not hold for MC sporadic tasks, as is demonstrated by the task set in Figure 1. It is easily verified that this task system has a correct scheduling policy for the SAS (execute τ_1 first, then τ_2 or τ_3 depending on τ_1's finishing time). However, it can also be verified that the job sequence where a job of τ_1 is released 1 time unit after the jobs from the other tasks does not have a correct online scheduling policy.

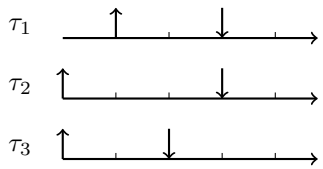
Claim 1: *The synchronous arrival sequence is not guaranteed to be the worst case for MC sporadic tasks.*

	C(LO)	C(HI)	D	P	Level
τ_1	1	2	2	10	HI
τ_2	1	1	3	10	HI
τ_3	1	–	2	10	LO

(a) The task system



(b) The SAS is schedulable



(c) An unschedulable sequence

Figure 1: A task system demonstrating that the SAS is not always a worst case.

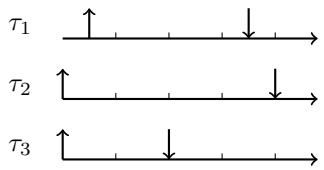
The situation is even worse if we allow non-integer release times of jobs. Figure 2 shows a task set for which all job sequences with integer release times are schedulable, but for which a job sequence with rational release times is unschedulable. This is in contrast to non-MC tasks, for which the SAS remains the worst-case even if non-integer release times are allowed.

Claim 2: *It is not enough to consider integer release times when analyzing sporadic MC task systems.*

The lack of a concrete job sequence to focus on severely hampers any effort to perform schedulability analysis of MC sporadic task systems. Can we identify a suitable replacement for the SAS?

	$C(\text{LO})$	$C(\text{HI})$	D	P	Level
τ_1	1	2	3	10	HI
τ_2	1	2	4	10	HI
τ_3	1	–	2	10	LO

(a) The task system



(b) An unschedulable sequence

Figure 2: A task set demonstrating that the worst case may have have non-integer release times.

Question 1: *For a given MC sporadic task system, can we efficiently identify some smallish set of job sequences, such that the task system is online schedulable if and only if all job sequences in that set are online schedulable?*

It is possible that the answer to Question 1 is "no", not only because it could be hard to identify such as set, but because such a set might not even exist. With current knowledge it is conceivable that an MC sporadic task system can generate two job sequences that share a common prefix, such that both job sequences have correct online schedules, but require different scheduling decisions during that prefix. Since an online scheduler for an MC sporadic task system presented with that prefix can not know which of the job sequences that eventually will be generated, it must make a scheduling decision during the prefix that is incorrect for at least one potential future. This is the motivation for the next question. If the answer to Question 2 is "no", then job sequences do not seem to be a good abstraction for analyzing MC sporadic task systems.

Question 2: *If all job sequences that can be generated by an MC sporadic task system have correct online schedules, does it follow that the task system has a correct online scheduler?*

Another task model to consider is the (strictly) periodic task model, in which each task system can generate only a single job sequence. If we are unable to find satisfactory answers to Questions 1 and 2, it seems as if the added flexibility that the sporadic task model offers over the periodic one comes at a high price in the MC setting. This is in contrast to the non-MC case where sporadic tasks are as easy to analyze as (synchronous) periodic ones on uniprocessors. Perhaps, then, research into the scheduling of MC periodic task systems is also a worthwhile effort? We are not aware of any results targeted at MC periodic tasks specifically.

Question 3: *Is scheduling or analysis for MC periodic tasks significantly easier than for MC sporadic tasks?*

Not only do we have a limited understanding of the difficult cases that may arise, but also poor understanding of how to best schedule MC systems. No optimal scheduling policy, of any complexity, is yet known for MC sporadic or periodic task systems.

Question 4: *What online scheduling policies are optimal for MC sporadic or periodic task systems?*

A remaining question concerns the computational complexity of the feasibility decision problem: does a given MC task system have a correct scheduling policy? Baruah et al. [2] have showed that the corresponding decision problem for (synchronous) fixed MC job sequences is strongly NP-complete for any constant K number of criticality levels ($K \geqslant 2$). It is also known that the feasibility problem for non-MC sporadic or periodic task systems is strongly coNP-complete. It is trivial to reduce the non-MC case to the MC case, and it is also trivial to reduce the feasibility problem for synchronous MC job sequences to the feasibility problem for MC periodic task systems (either synchronous or asynchronous). If follows that the feasibility problem for MC periodic task systems is both NP-hard and coNP-hard, and is therefore unlikely to be in either of NP or coNP. It is not clear that feasibility for MC sporadic task systems is also NP-hard, but it seems reasonable to suspect that it is at least as hard as feasibility for MC periodic tasks.

Question 5: *What is the complexity of the (online) feasibility problem for MC sporadic or periodic tasks?*

References

[1] S. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. *Handbook of scheduling: Algorithms, models, and performance analysis*, 3, 2004.

[2] S. Baruah, K., V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, Aug. 2012.

[3] H. Li and S. Baruah. An algorithm for scheduling certifiable mixed-criticality sporadic task systems. In *RTSS*, pages 183 –192, 2010.