1 Introduction

In mixed-criticality systems, functionalities of different degrees of importance (or criticalities) are implemented upon a common platform. Such mixed-criticality implementations are becoming increasingly common in embedded systems – consider, for example, the Integrated Modular Avionics (IMA) software architecture used in aviation [5] and the AUTOSAR initiative (AUTomotive Open System ARchitecture – www.autosar.org) for automotive systems. As a consequence the real-time systems research community has recently been devoting much attention to better understanding the challenges that arise in implementing such mixed-criticality systems; this includes research on various aspects of mixed-criticality scheduling. Most of this prior work draws inspiration from the seminal work of Vestal [6], and has taken the approach of validating the correctness of highly critical functionalities under more pessimistic assumptions than the assumptions used in validating the correctness of less critical functionalities. (For example, a piece of code may be characterized by a larger worst-case execution time (WCET) [6] in the more pessimistic analysis, or recurrent code that is triggered by some external recurrent event may be characterized by a higher frequency [1].) All functionalities are expected to be demonstrated correct under the less pessimistic analysis, whereas the analysis under the more pessimistic assumptions need only demonstrate the correctness of the more critical functionalities.

In this paper we take a somewhat different perspective on mixed-criticality scheduling: the system is analyzed only once, under a single set of assumptions. The mixed-criticality nature of the system is expressed in the requirement that while we would like all functionalities to execute correctly under normal circumstances, it is essential that the more critical functionalities execute correctly even when circumstances are not normal. To express this formally, we model the workload of a MC system as being comprised of a collection of real-time jobs — these jobs may be independent, or they may be generated by recurrent tasks. Each job is characterized by a release date, a worst-case execution time (WCET), and a deadline; each job is further designated as being hi-criticality (more important) or lo-criticality (less important). We desire to schedule the system upon a single processor. This processor is unreliable in the following sense:
while under normal circumstances it completes one unit of execution during each time
unit (equivalently, it executes as a speed-1 processor), it may at any instant lapse into a
degraded mode during which it may only complete as few as \( s \) units of execution during
each time unit, for some (known) constant \( s < 1 \). It is not \textit{a priori} known when, or
whether, such degradation will occur. We seek efficient scheduling strategies that \textit{guar-
antee to complete all jobs by their deadlines if the performance of the processor does not
degrade during run-time, while simultaneously guaranteeing to complete all HI-criticality
jobs if the processor does suffer a degradation in performance}.

\section{Results obtained}

We first considered the problem of scheduling finite collections of mixed-criticality jobs
upon processors that are able to detect if and when they transit from normal to faulty
mode. For this problem, our scheduling strategy is as follows. Given a finite instance \( I \)
of mixed-criticality jobs, prior to run-time we construct a scheduling table \( S(I) \), for use
while the processor is in normal (i.e., not faulty) mode. This scheduling table should
possess the property that each job completes by its deadline if the schedule is executed
upon a unit-speed processor. During run-time scheduling decisions are initially made
according to scheduling table \( S(I) \). If at any instant it is detected that the processor
has transited to faulty mode, scheduling table \( S(I) \) is no longer used; instead, all LO-
criticality jobs are immediately discard and henceforth the (remaining) HI-criticality ones
are executed according to EDF [3, 2].

We have obtained algorithms based on linear programming, for constructing these
scheduling tables \( S(I) \) from instance \( I \) in an optimal manner, if the implementation
platform is a preemptive uniprocessor.

We have also obtained optimal algorithms for solving the following related optimiza-
tion version of our problem (again, on preemptive uniprocessors): given a collection \( I \)
of mixed-criticality jobs, what is the smallest speed \( s \) such that we are able to devise a
correct scheduling strategy for \( I \) upon a unit-speed processor that has degraded speed
\( s \)?

We have considered recurrent (periodic [3] and sporadic [4]) mixed-criticality tasks as
well, and have obtained algorithms for devising correct scheduling strategies for systems
of such tasks upon preemptive uniprocessors.

\section{Future work}

A large number of interesting and important problems on scheduling upon unreliable
processors remains open. We briefly list a couple that we are currently working on.

- Although the LP-based algorithms mentioned in Section 2 above establish that
the problems can be solved in polynomial time, we continue to seek more efficient
algorithms; ideally, we would like algorithms that have run-time comparable to
EDF: \( O(n \log n) \) where \( n \) denotes the number of jobs (in addition to possessing
other “EDF-like” properties such as relatively few preemptions, etc.)

- The results described in Section 2 were obtained for a model that assumes that
the implementation platform is capable of self-monitoring, in the sense that the
processor immediately knows if and when it degrades from normal to degraded mode. Not all processors are capable of such self-monitoring; we are studying mixed-criticality scheduling upon processors that lack this capability.

- If preemptions are not allowed, all these problems become NP-hard. We are seeking efficient algorithms for obtaining approximate solutions.

References


