Information Processing Letters 116 (2016) 508–512

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Generalizing fixed-priority scheduling for better schedulability in mixed-criticality systems

Yao Chen [a,*], Kang G. Shin [b], Huagang Xiong [a]

[a] School of Electronic and Information Engineering, Beihang University, Beijing, 100191, China
[b] Real-Time Computing Laboratory, EECS, The University of Michigan, Ann Arbor, MI 48109, USA

ARTICLE INFO

Article history:
Received 2 August 2015
Received in revised form 15 February 2016
Accepted 15 February 2016
Available online 27 February 2016
Communicated by Nathan Fisher

Keywords:
Real-time systems
Mixed-criticality
Fixed-priority scheduling
Schedulability analysis
Priority assignment algorithm

ABSTRACT

The design of mixed-criticality systems is often subject to mandatory certification and has been drawing considerable attention over the past few years. This letter studies fixed-priority scheduling of mixed-criticality systems on a uniprocessor platform but in a more general way, using different priority orderings in different execution phases and considering them collectively. Then a sufficient response-time analysis is developed and a new priority assignment scheme is proposed. This generalized approach has potential in better schedulability performance for mixed-criticality systems.

1. Introduction

There has been an increasing trend of integrating multiple functionalities of different criticality levels upon a shared hardware platform to address the growing demand for computing power cost-efficiently in safety-critical real-time systems. When certifying such mixed-criticality (MC) systems, the certification authorities and manufacturers mandate different assumptions about the worst-case system behavior, depending on the criticality of concerned functionalities. To simultaneously guarantee temporal correctness at all different levels of assurance, scheduling issues arising from these multiple certification requirements have been studied extensively [1–4].

As a preferred approach in industry due to its flexibility and ease of predictability, fixed-priority (FP) preemptive scheduling was firstly introduced into the MC scenario in Vestal's seminal work [1]. Schedulability analysis based on response-time is presented there and then improved by Baruah [2]. The Adaptive Mixed Criticality (AMC) scheme in [2] has been shown to be one of the most effective MC scheduling approaches and forms the basis of further related studies [5–7]. Note the above studies share a common assumption that enforces the same task priority ordering throughout the system's life.

As in general multi-mode systems [8], enabling change of priorities in the event of a mode-change has been studied in MC systems [3,9]. However, one characteristic of these schemes is that they do not distinguish the phase of mode transition from the steady new mode. Besides, the Priority May Change (PMC) approach [9] deals with behaviors of different criticality levels individually, ignoring the dependency. In this letter, the execution model is further relaxed that priorities can be re-assigned not only in the event of mode change but also when the mode transition ends. Based on this, we investigate FP scheduling of MC systems by considering different execution phases collectively.

* Corresponding author.
E-mail addresses: chenyao.kevin@gmail.com (Y. Chen), kgshin@umich.edu (K.G. Shin), hgxiong@buaa.edu.cn (H. Xiong).

http://dx.doi.org/10.1016/j.ipl.2016.02.009
0020-0190/

2. System model

Task model: An MC sporadic task τ_i is characterized by a four-tuple $\tau_i = (T_i, D_i, \xi_i, \vec{C}_i)$, where T_i denotes its period, D_i the relative deadline, ξ_i the criticality level, and $\vec{C}_i$ a vector of worst case execution time (WCET) estimations. In this letter constrained deadline is assumed and our attention will be restricted to dual-criticality systems, but the main principle can be scaled to an increased number of criticality levels with further efforts. Formally, for task τ_i, we assume $\xi_i \in \{LC, HC\}$ and $\vec{C}_i = \{C_i^L, C_i^H\}$ with $C_i^L \leq C_i^H$, where C_i^L (C_i^H) denotes low-criticality (*LC*) (high-criticality (*HC*)) WCET.

Certification requirements: Consider a system $\Gamma = \{\tau_i | 1 \leq i \leq n\}$ consisting of a set of independent MC tasks. During different runs the system could show different behaviors generally and its mandated temporal correctness differs depending on the concerned criticality level. For such MC system to be certified correct, the following timing requirements should be guaranteed:

- No job of any task τ_i can execute for more than $C_i^{\xi_i}$, otherwise the system is exhibiting *erroneous behavior*.
- As long as no job executes for more than C_i^L, the system is regarded as exhibiting *LC behavior*, and all jobs should meet their deadlines.
- If some *HC* job executes for C_i^L without completion, the system begins to exhibit *HC behavior* and from this instant of criticality change only *HC* jobs are required to meet their deadlines.

The *HC* jobs that are active (released but not yet completed) upon occurrence of the criticality change are referred as *carry-over jobs* [4]. Recall that *LC* jobs are not required to complete by their deadlines for *HC* system behaviors, which is an implication of the certification requirements. This strictness was then relaxed by Santy [10] which allowed *LC* tasks to execute after the criticality change and the system to change back to *LC* mode. And more related schemes can be found in a recent survey [11].

Scheduling strategy: As a special case of multi-mode systems [8], dual-criticality systems could go through three distinct phases: steady *LC* mode, mode transition period and steady *HC* mode. The mode transition period represents the time interval between the criticality mode change and the instant when all carry-over jobs have completed their execution. To favor accommodating the change of system load upon occurrence of the criticality change, in this letter we prefer to use different priority orderings in different phases and introduce a new strategy for task dispatching, called Generalized Fixed-Priority (GFP). Specially, each task τ_i has three unique-priority parameters:

- P_i^L: priority for jobs executed in the steady *LC* mode;
- P_i^T: priority for the carry-over job when the system exhibits *HC* behavior;
- P_i^H: priority for jobs released after the criticality change.

Starting from 1, assume the larger value represents the higher priority. Initially, the system starts in the steady *LC* mode and the scheduler selects the highest priority job for execution according to the ordering of P_i^L. When the criticality change occurs, the system switches to the mode transition period, *LC* jobs are discarded and *HC* jobs are scheduled according to the ordering of P_i^T and P_i^H, which may interleave with each other for different tasks, for example $P_i^T > P_j^T > P_j^H > P_i^H$. Finally in the steady *HC* mode, the scheduler selects the highest priority job according to the ordering of P_i^H.

Note that P_i^T is exclusive for the carry-over job, straddling the criticality change. The advantage is to mitigate the problem that some carry-over job may execute late in the steady *LC* mode and thus has to complete its remaining *HC* execution in a very short scheduling window after the criticality change. Since each job must have completed execution before the next release under the assumption of constrained deadline, we have $P_i^T > P_i^H$ for *HC* tasks. As for *LC* tasks, since they are prevented from executing after the criticality change, their priorities P_i^T and P_i^H are useless and ignored here.

3. Response time analysis

In this section a sufficient schedulability test is derived based on the same analysis framework as in [2,6], where response times in three distinct scenarios are studied.

3.1. Jobs finished in the steady LC mode

MC tasks behave exactly the same as traditional (non-MC) ones in the steady *LC* mode. Thus, the standard RTA method can be applied to derive τ_i's response time R_i^L:

$$R_i^L \leftarrow C_i^L + \sum\nolimits_{\forall \tau_k \in hp_L(i)} \lceil R_i^L / T_k \rceil C_k^L \tag{1}$$

where $hp_L(i) = \{\tau_k \in \Gamma | P_k^L > P_i^L\}$ denotes the set of tasks with higher priority than that of τ_i in the steady *LC* mode.

3.2. Carry-over job

Consider the carry-over job J_i^p in Fig. 1, released before the criticality change with span $s \in [0, R_i^L]$. For convenience of presentation, define task subset $hp_T^{xyz}(i)$ as

$$\{\tau_k \in \Gamma \backslash \tau_i | f(P_k^L, P_i^L) = x, f(P_k^T, P_i^T) = y, \\ f(P_k^H, P_i^T) = z\}$$

where x, y, z are binary variables, $f(u, v)$ is a binary function returning 1 if $u \geq v$, otherwise 0. Depending on the priority orderings, there are five possible situations for τ_k to interferer with J_i^p within the busy period $[t_0, f_i^p]$: $\tau_k \in hp_T^{xyz}(i)$ with $xyz \in \Omega = \{111, 110, 100, 011, 010\}$. Specially, $x = 1$ means τ_k has higher priority than τ_i in the steady *LC* mode and similar interpretation applies to y and z. Note that $hp_T^{000}(i)$ cannot interfere with τ_i and combinations $\{001, 101\}$ are ruled out due to constraint $P_k^T > P_k^H$.

3.2.1. Interference calculation

For each situation $xyz \in \Omega$, two kinds of interference of the higher priority task $\tau_k \in hp_T^{xyz}(i)$ are calculated: