Motivating Agent-Based Learning for Bounding Time in Mixed-Criticality Systems

Behnaz Ranjbar, Ali Hosseinghorban, and Akash Kumar, *Senior Member, IEEE*
Chair of Processor Design, CFAED, Technische Universität Dresden, Dresden, Germany
{behnaz.ranjbar, akash.kumar}@tu-dresden.de, ali.hosseinghorban1394@sharif.edu

***Abstract*—In Mixed-Criticality (MC) systems, the high Worst-Case Execution Time (WCET) of a task is a pessimistic bound, the maximum execution time of the task under all circumstances, while the low WCET should be close to the actual execution time of most instances of the task to improve utilization and Quality-of-Service (QoS). Most MC systems consider a static low WCET for each task which cannot adapt to dynamism at run-time. In this regard, we consider the run-time behavior of tasks and motivate to propose a learning-based approach that dynamically monitors the tasks' execution times and adapts the low WCETs to determine the ideal trade-off between mode-switches, utilization, and QoS. Based on our observations on running embedded real-time benchmarks on a real platform, the proposed scheme reduces the utilization waste by 47.2%, on average, compared to state-of-the-art works.**

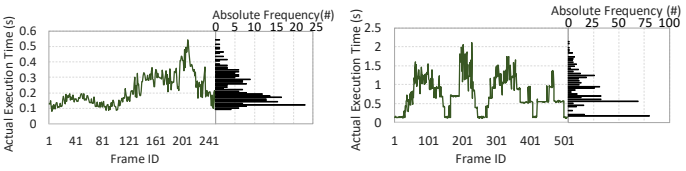
***Index Terms*—Mixed-Criticality, Mode Switching Probability, Machine Learning, Service Adaptation, WCET Analysis.**

I. Introduction

MIXED-CRITICALITY (MC) systems integrate a large number of real-time tasks with different criticality levels onto a common hardware platform to meet stringent requirements such as cost, space, and timing [1]–[4]. Medical devices, automotive, and avionics are the most common safety-critical applications, evolving into MC systems [2], where the successful execution of tasks with Higher-Criticality levels (HC tasks) must be guaranteed in all circumstances to prevent catastrophic damages, while a higher number of Low-Criticality (LC) tasks should be executed to improve service requirements (i.e., Quality-of-Service (QoS)) and consequently, the processor utilization [3], [5].

From the MC tasks' execution times perspective, multiple WCETs are determined corresponding to the multiple criticality levels. A well-known type of MC system is a dual-criticality system (consisting of LC and HC tasks) in which two WCETs (low (C^{LO}) and high (C^{HI})) are determined [1]–[7]. The C^{HI} of a task is a pessimistic bound, the maximum execution time of the task under all circumstances. However, this bound is high, and considering it to schedule the tasks leads to poor processor utilization and QoS (i.e., fewer LC tasks can be scheduled) [3]. To this end, MC systems consider a C^{LO} for HC tasks that should be close to the actual execution time of most task instances to improve utilization and QoS. At run-time, the system starts its operation in low-criticality mode (LO mode), and if the execution time of at least one HC task exceeds its C^{LO}, the system switches to the high-criticality mode (HI mode). To guarantee the correct execution of HC tasks in HI mode, C^{HI} are considered to schedule HC task. Since HC tasks may execute longer in HI mode compared to LO mode, the LC tasks are dropped/degraded to their minimum service requirements to guarantee the correct execution of HC tasks before their deadlines [2], [6]–[8].

Supported in part by the German Research Foundation within the Cluster of Excellence Center for Advancing Electronics Dresden at the TU Dresden.



(a) Input video with few objects to detect

(b) Input video with few and many objects to detect

Fig. 1: Execution time values for two different time recording videos as input for Object Detection function during run-time and their time distribution. This figure shows that both aspects of run-time and design-time behavior should be considered in MC system design and task properties determination.

As can be realized, the low WCETs (C^{LO}) play an important role in improving the MC system's QoS. Determining the high values for C^{LO}s can minimize the mode switches but reduce the processor utilization due to scheduling fewer tasks. On the other hand, the utilization can be maximized by determining the low values for C^{LO}s, but with a high number of mode switches, which is not desirable. Although there are many approaches like what is presented in [9] and tools like OTAWA [10] to determine the C^{HI}, there are few approaches for determining the C^{LO}s in MC systems. These few approaches [1], [3], [6], [8] analyze the tasks at design-time, and set the constant WCETs for tasks in LO mode, which remain unchanged during run-time. Such static techniques can cause significant performance loss for LC tasks or processor under-utilization if the C^{LO}s are not close to actual execution times. In general, the actual execution time of tasks depends on their input values. Due to the spatial or temporal correlation in the input data stream like video, the execution times of the tasks are often temporally correlated.

A. Motivational Example

Fig. 1 shows the computational times of the object detection function running on the ODROID XU4 board powered by ARM Cortex A7. Note that the object detection function is one of the main functions in an autonomous driving application – an MC system. For input, videos from a road camera in the two different time slots, converted to motion jpegs, are given to

the function of detecting cars on the road. The videos were recorded for a period of time when it experienced both light and heavy traffic. Fig. 1 shows how the computation times of detecting objects vary during run-time. The computation time values in this function depend on the number of objects to be detected. As we can see, the times of multiple jpeg images are clustered due to the temporal correlation between the subsequent inputs presented to the application.

For this example, static approaches such as the one presented in [1], [3], [6] set the static C^{LO}, considering the execution time of the majority of instances. This static WCET works fine for some time, but it may lead to frequent mode switches when there are many objects to detect (e.g., heavy traffic) or poor utilization when there are few objects to detect in this function (e.g., light traffic). As a result, proposing an adaptive scheme to determine C^{LO} dynamically may significantly improve the mode switches, QoS, and utilization. Therefore, the system's run-time behavior can be investigated by monitoring the execution times and adjusting C^{LO}.

B. Proposed Approach

We propose a novel learning-based run-time scheme for determining C^{LO} to:

- Effectively reduce the nummber of system mode switches
- Have high processor utilization and, consequently, a high value of QoS. Note that, the QoS is defined as $(n_{LC}^{schd})/(n_{LC}^{max})$ [11]–[13], where n_{LC}^{schd} and n_{LC}^{max} are the number of LC tasks, that can be scheduled, and the number of all LC tasks in the system, respectively
- Guarantee the system schedulability in each criticality mode
- Not be affected by sudden changes in execution times

To the best of our knowledge, there is no method yet to determine C^{LO} of MC tasks at run-time based on the behavioral system changes while improving the QoS.

It is a challenge to set C^{LO} for each HC task to draw a trade-off between the objectives: system utilization and the number of mode switches. To address the challenge, we monitor the run-time execution times of HC tasks and adapt their C^{LO} at run-time to achieve a higher QoS while having fewer mode switches based on the variation in execution times due to the input and environmental changes. The proposed approach consists of design- and run-time phases. Here, the task schedulability must be guaranteed at both phases, and the C^{LO} adaptation is done at run-time. The existing MC scheduling technique, EDF-VD [1], [3], [6] (which has been used in many studies in the last decade), is applied. Hence, the learning process is separate from the task scheduler, and we do not use learning techniques for task scheduling. At run-time, reinforcement learning is used to perform run-time management and update the C^{LO} values.

C. Evaluation

The proposed approach's efficacy is evaluated in terms of mode switches, and utilization waste. We conducted experiments by the real benchmark, object detection function, explained in the motivational example of Section I-A. To obtain their execution times, we run the benchmark with various inputs of light and heavy traffic on Cortex A7 of the ODROID XU4 board (equipped with Ubuntu 18.04 as OS) with maximum frequency ($1.4GHz$). We compare the results with the results of [3], [6]. As mentioned in [3], since most papers like [1], [6], [8], consider the same policy to determine the C^{LO} (i.e., defining a fraction of C^{HI} as C^{LO}), we select [6] as a representative of these schemes and do the experiments with three fractions of C^{HI} as C^{LO} ($\lambda = \frac{C^{LO}}{C^{HI}} = \{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}\}$). Table I presents the percentage of task overruns (which leads to mode switches) and the average percentage of wasted utilization for the proposed approach and [3], [6]. As can be seen, although the number of mode switches in the proposed approach is higher than the results of some scenarios, the wasted utilization is lower compared to all other approaches (47.2% on average), which leads to higher QoS.

TABLE I: Number of deadline misses and gained utilization of different methods for Object Detection function in Fig. 1, where there are many objects

Metrics	Proposed Approach	[3]	[6] $\lambda = \frac{1}{2}$	[6] $\lambda = \frac{1}{4}$	[6] $\lambda = \frac{1}{8}$
$\#MS^{Sys}$	17%	11%	0	5%	45%
$Util^{Wst}$	28%	46%	76%	52%	47%

II. Conclusion and Future Work

This article motivated the agent-based learning to analyze the MC tasks at run-time and determine their WCET based on the task behavioral changes. The proposed adaptive scheme employs the ML techniques to improve the QoS, while guaranteeing the task schedulability and timeliness.

In future research, we would extend our scheme by reducing the complexity of the ML technique to reduce its timing overhead. Besides, a design-time data set training would be investigated to speed up the run-time learning process.

References

[1] S. Baruah *et al.*, "The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems," in *ECRTS*, 2012.

[2] A. Burns and R. I. Davis, "A Survey of Research into Mixed Criticality Systems," *ACM CSUR*, vol. 50, no. 6, 2017.

[3] B. Ranjbar *et al.*, "Improving the Timing Behaviour of Mixed-Criticality Systems Using Chebyshev's Theorem," in *DATE*, 2021.

[4] X. Gu and A. Easwaran, "Dynamic Budget Management with Service Guarantees for Mixed-Criticality Systems," in *IEEE RTSS*, 2016.

[5] B. Ranjbar *et al.*, "BOT-MICS: Bounding Time Using Analytics in Mixed-Criticality Systems," *IEEE TCAD*, vol. 41, no. 10, 2022.

[6] D. Liu *et al.*, "Scheduling Analysis of Imprecise Mixed-Criticality Real-Time Tasks," *IEEE TC*, vol. 67, no. 7, 2018.

[7] B. Ranjbar *et al.*, "FANTOM: Fault Tolerant Task-Drop Aware Scheduling for Mixed-Criticality Systems," *IEEE Access*, vol. 8, 2020.

[8] H. Su *et al.*, "An Elastic Mixed-Criticality Task Model and Early-Release EDF Scheduling Algorithms," *ACM TODAES*, vol. 22, no. 2, 2016.

[9] R. Wilhelm *et al.*, "The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools," *ACM TECS*, vol. 7, no. 3, 2008.

[10] C. Ballabriga *et al.*, "OTAWA: An open toolbox for adaptive WCET analysis," in *SEUS*. Springer, 2010.

[11] B. Ranjbar *et al.*, "Learning-Oriented QoS-and Drop-Aware Task Scheduling for Mixed-Criticality Systems," *Computers*, vol. 11, no. 7, 2022.

[12] Z. Li and S. He, "Fixed-Priority Scheduling for Two-Phase Mixed-Criticality Systems," *ACM TECS*, vol. 17, no. 2, 2017.

[13] B. Ranjbar *et al.*, "Toward the Design of Fault-Tolerance-and Peak-Power-Aware Multi-Core Mixed-Criticality Systems," *IEEE TCAD*, vol. 41, no. 5, 2022.