# Proceedings of the 9th Real-Time Scheduling Open Problems Seminar (RTSOPS)

held in conjunction with the 30th Euromicro Conference on Real-Time Systems (ECRTS)

Edited by Tam Chantem[1] and Dorin Maxim[2]
[1] *Virginia Tech, USA*
[2] *University of Lorraine, France*

Barcelona, Spain
July 3, 2018

# Message from the Chairs

It is our great pleasure to introduce this volume that includes the Proceedings of the 9th Real-Time Scheduling Open Problems Seminar (RTSOPS 2018). This volume represents the continued openness in the Real-Time Systems research community to share and discuss unsolved problems concerning real-time scheduling theory and applications.

This 9th edition of the seminar series is co-located with the 30th Euromicro Conference on Real-Time Systems (ECRTS 2018) that is being held in Barcelona, Spain. The day long seminar features 12 paper presentations distributed over 4 sessions. Each session includes ample collaboration time, during which we encourage all participants to interact in groups and tackle the presented problems. The hope is that we make headway in solving the problems and that more complete problem definitions and solutions will emerge as a result of the discussions initiated during the workshop.

We would like to thank the generosity of the Program Committee for their time and attention to detail that helped us assemble the program for the day. We are also grateful to the RTSOPS Steering Committee for their feedback and advice. This program would not have been possible without the efforts and support of the ECRTS 2018 organizing committee.

We invite all of you to join us in taking advantage of this excellent opportunity to learn and interact with our fellow colleagues. We hope you enjoy RTSOPS 2018.

**Tam Chantem[1] and Dorin Maxim[2]**
[1]Virginia Tech, USA
[2]University of Lorraine, France
RTSOPS 2018 Program Chairs

## RTSOPS 2018 Technical Program Committee

Yasmina Abdeddaïm, ESIEE Paris, France

Kunal Agrawal, Washington University in St. Louis, USA

Mohammad Ashjaei, Mälardalen University, Sweden

Antoine Bertout, University of Poitier, France

Alessandro Biondi, Scuola Superiore Sant'Anna, Italy

Jalil Boudjadar, Aarhus University, Denmark

Silviu Craciunas, TTTech Computertechnik, Austria

Benjamin Lesage, University of York, UK

Mitra Nasri, Max Planck Institute for Software Systems, Germany

Florian Pölzlbauer, Virtual Vehicle, Austria

## RTSOPS Steering Committee

Liliana Cucu-Grosjean, INRIA Paris, France

Robert Davis, University of York, UK

Nathan Fisher, Wayne State University, USA

# Technical Program

# Use of probabilities and formal methods to control system criticality levels

Jasdeep Singh[1], Zhishan Guo[2], Luca Santinelli[1], Guillaume Infantes[1], David Doose[1] and Julien Brunel[1]
[1]ONERA -DTIS Toulouse, name.surname@onera.fr
[2] Missouri University of Science and Technology Rolla MO, guozh@mst.edu

**Introduction and motivations.** The gap between the actual execution and the worst-case bounds may be significantly large with current real-time systems. Instead of completely wasting the processor capacities within the gap, latest trends start to implement functionalities of different degrees of importance, or criticality, upon a common platform. This allows less important tasks to execute in these gaps under normal circumstances, and may be dropped in an occasional situation where jobs of higher importance execute beyond their estimated running time [1, 2].

A Mixed Criticality (MC) real-time system is one that has two or more distinct criticality levels e.g., safety critical jobs (HI-criticality), mission critical or low critical jobs (LO-criticality). Such systems are defined to execute in a number of criticality modes, each mode specifying execution conditions and system criticality. All the possible modes have to be characterized and analyzed in order to guarantee the predictability of the system. See [3] for an overview of the MC problems.

As drawn by [5], MC problems can be approached with probabilities to quantify and manage the unlikely events and reduce the pessimism. We believe that a tighter coupling between MC problems and probabilistic frameworks can end up into smarter scheduling decisions and more efficient utilization of the computational resource.

With this abstract, we propose to model and investigate criticality mode behaviors of MC systems by using probabilities and formal methods. The goals are: 1) quantifying the probability of jobs and system entering HI-criticality mode, and 2) controlling this probability by selectively removing LO-criticality jobs from the schedule. Formal methods and probabilities are used to model jobs and system HI-criticality modes. The probabilities are actively applied into scheduling decisions.

**Background - probabilistic computational models:** A real-time application is a set $\Gamma$ of jobs, $\Gamma = \{J_{ij}\}$ , $J_{ij}$ is the $j$-th job of $i$-th task. A probabilistic real-time application is called so whenever at least one of its elements is described with a probabilistic parameter; in this case it is the probabilistic Worst-Case Execution Time (pWCET) for task execution, but it can be extended to any model parameter. The pWCET is the worst case distribution which is able to upper bound any job execution behavior [4]. Hereby, we assume the pWCETs to be continuous distributions, but the same conclusions can be drawn with discrete distributions.

The pWCET as a continuous random variable $\mathcal{X}$ is represented with the Probability Density Function (PDF) $f_{\mathcal{X}}(x)$, the Cumulative Distribution Function (CDF) $F_{\mathcal{X}}(x) = \int_0^x f_{\mathcal{X}}(y)dy$, and the Inverse Cumulative Distribution Function (ICDF) $\overline{F}_{\mathcal{X}}(x) = 1 - \int_0^x f_{\mathcal{X}}(y)dy$ which gives the exceeding threshold probability.

The job model $J_{ij} = \{\mathcal{C}_{ij}, a_{ij}, d_{ij}, p_{ij}\}$ is such that $J_{ij}$ arrives at time $a_{ij}$, $d_{ij}$ is the job absolute deadline, $p_{ij}$ is the job priority, and $\mathcal{C}_{ij}$ is the job pWCET. Jobs can be seen as task instances in a periodic or sporadic task application. A task $\tau_i$ is the tuple $\tau_i = \{\mathcal{C}_i, T_i, D_i\}$ where $\mathcal{C}_i$ is the pWCET ($\mathcal{C}_{ij} = \mathcal{C}_i \forall j$), $T_i$ is the period, and $D_i$ is the deadline of the task ($D_i \leq T_i$).

The scheduling policy defines the job ordering by imposing job-wise priorities $p_{ij}$. Both job static priority or dynamic priority schemes can be used here. The hyperperiod is defined as the least common factor $lcm()$ of all the task periods, $lcm(T_i), i = 1, 2, \ldots, m$. It is the scope of the schedulability analysis and the criticality analysis we propose, since we assume the job execution suspended at the job deadline. In the hyperperiod there are $n$ jobs from $m$ tasks.

In a probabilistic framework, for each job there exists a probabilistic Worst-Case Response Time which is a distribution to represent the worst-case response time of the job. pWCRT is the result of probabilistic schedulability analysis. Same as for the pWCET, the pWCRT is assumed to be a continuous random variable and can be represented with PDF, CDF and ICDF. Discrete pWCRTs are applicable without modifying the proposed reasoning. To compute pWCRT, it is possible to apply any of the existing probabilistic schedulability analysis approaches e.g., [8, 7] for discrete distributions, and [9] for continuous distributions.

**Background - Markov decision process:** Markov decision processes (MDPs) are mathematical frameworks for modeling decision making in situations where the outcomes are partly random and partly under the control of a decision maker [6]. More precisely, a MDP decision process is a discrete time stochastic control process where at each time step, the process is in some state $s$, and the decision maker may choose any action $a$ that is available in $s$. The probability that the process moves into its new state $s'$ is influenced by the chosen action. Specifically, it is given by the probability state transition function $Pr_a(s, s')$. A Markov decision process $M$ is defined as a set of states $S_{MDP}$ and state transitions given by a Q-matrix $Q_{MDP}$, $M = \{S_{MDP}, Q_{MDP}\}$. Formal model checking can be done on MDP in order to know the probability of reaching certain states (property to formally verify) by taking certain paths. Figure 1 shows an example of MDP with states and state transitions weighted over the probability $p$ for the transition of happening. We propose to use MDP to model jobs and system criticality and make use of its mathematical foundations.

**Mixed criticality.** At first instance, we consider the two-criticality level case for mixed criticality probabilistic real-time applications. In it, each job is designated as being of either higher criticality HI-criticality or lower criticality LO-criticality. Two different behaviors (modes) are specified for each HI-criticality job: a high mode, where the job executes in highly critical and more demanding conditions; a low mode which is the nominal working condition for the job where it executes in normal conditions. A LO-criticality job has only low mode.

A HI-criticality job $J_{ij}^{\text{HI}}$ is $J_{ij}^{\text{HI}} = \{\mathcal{C}_i, a_{ij}, d_{ij}, l_{ij}, \chi_{ij}\}$, where $\mathcal{C}_i$ is the job pWCET, $a_{ij}$ is the arrival instant, $d_{ij}$ is the deadline of the job, and $\chi_{ij}$ is the job criticality level [1]. $\chi_{ij}$ can take two values at runtime: HI and LO; $\chi_{ij} \in \{\text{HI}, \text{LO}\}$. $l_{ij}$ describes the threshold with which we define the job criticality mode. A LO-criticality job $J_{kr}^{\text{LO}}$ is $J_{kr}^{\text{LO}} = \{\mathcal{C}_k, a_{kr}, d_{kr}, \chi_{kr}\}$. $\chi_{kr}$ for a LO-criticality job can take only one value, LO, $\chi_{kr} \in \{\text{LO}\}$.

The criticality mode of the jobs can change at runtime depending on their scheduling. The threshold $l_{ij}$ applies to the job pWCRT and defines the HI-criticality job criticality mode – response time threshold. It is such that if the job finishing time is beyond the threshold, the job is considered to execute in HI-criticality mode. Otherwise, the job executes in LO-criticality mode. Figure 2 illustrates that with HI and LO criticality regions which are defined such that job $J_{ij}$ is in high criticality mode if it finishes executing in the interval $[l_{ij}, \infty)$. It is in LO-criticality mode if it finishes executing in the interval $[0, l_{ij})$. To the HI-criticality mode there is an associated probability $P^{\text{HI}}$ that is the probability that the execution of a job $J_{ij}$ exceeds $l_{ij}$ (the exceeding probability for $l_{ij}$ as the probability for the job of entering

HI-criticality mode); $P^{\text{LO}} = 1 - P^{\text{HI}}$ is the probability that $J_{ij}$ remains in LO-criticality mode. It is also possible to have the threshold $l_{ij}$ applying to the job pWCET – execution time threshold, for a more classical MC modeling of the jobs criticality levels. We choose to apply pWCRT to characterize the job criticality level because this way all the interference effects are included and accounted for when deciding which conditions trigger a HI-criticality mode for the task.

For MDP modeling the MC behavior of the system, the HI-criticality jobs are represented with two states: HI-criticality and LO-criticality, and the state transitions is the probability of mode transitions $P^{\text{HI}}$ and $P^{\text{LO}}$; Figure 1 as an example of MDP for two-critical probabilistic real-time application. In there, HI-criticality states (HC) and LO-criticality states (LC) are represented for each job together with the probability for $J_{ij}$ of being in one state or the other $P_{ij}$ and $1 - P_{ij}$. From the job criticality level, the system criticality level can be defined as: *the system criticality level $\chi$ is equal to* HI *if at least $k$ out of $n$* HI*-criticality jobs enter high criticality mode*. This is a generic definition of system criticality level for $1 \leq k \leq n$, which extend to more conservative definitions where as soon as one HI-criticality job moves to HI-criticality mode, the system moves to HI-criticality mode.

**Open problem.** The problem we intend to tackle concerns to the investigation of the system criticality at runtime with the help of probabilities and formal methods. We can state the problem as: the probability that a mixed criticality real-time application enters high criticality mode should be less than or equal to certain probability $P_{sys}^{max}$. $P_{sys}^{max}$ is a requirement given and the analysis we propose is such that the resulting MC scheduling will allow to meet the requirement.

This is an open problem in the sense that there are no solutions to that. We also believe it is an interesting one because it represents a way to apply probabilities for characterizing the system criticality behavior as well as use them into scheduling decisions. To approach it, we would like to define what we call *criticality analysis*.

From $\Gamma$ and the pWCRTs computed with probabilistic timing analysis, we identify the criticality modes of each jobs (with the threshold $l$ applied to pWCRTs) and the probability $P^{\text{HI}}$ associated. $P^{\text{HI}}$ is the probability used to label the MDP in order to model criticality mode transitions. We can assume that the MDP is made from the HI-criticality jobs only, since the LO-criticality jobs would have only one state associated and will not contribute to the analysis.

Every combination of job modes are accounted for and encoded into the MDP as a path between states. Path analysis and model checking can be carried out for the MDP in order to compute the probability of the system criticality mode from the generic definition $(k, n)$, and any interpretation e.g., $k = 1$, $k = n$.

1) At first, we propose to *quantify* the probability of HI-criticality mode $P$ for the system. This is done from the MDP by considering the probability of each path corresponding to the $(k, n)$ system criticality definition: an exact probabilistic representation of job modes and system mode which could happen at runtime (probability of occurrence) is possible. 2) Then, with $P \leq P_{sys}^{max}$ we *act* on the system scheduling in order to meet such constraint and reduce the system HI-criticality. This control can apply to system criticality or to specific set of jobs. Figure 1 illustrates an example of MDP for the MC problem with paths between states and the state transition probabilities to be composed into path probabilities.

The scheduling strategies we intend to develop consist of finding the LO-criticality jobs to drop from schedule in order to reduce $P$. They are meant to select LO-criticality jobs with the most impact on HI-criticality jobs. Those strategies will be conceived as an off-line analysis that minimizes the number of LO-criticality jobs to be removed in order to guarantee the occurrence probability of HI-criticality modes.

As today, we are beginning to formalize the *criticality analysis* defining how to model criticality with MDP and how to formally explore MDPs. There is a MDP implementation already available, that we apply to study the impact that LO-criticality jobs have on the probability HI-criticality mode $P$. The implementation uses a Continuous Time Markov Chain based schedulability analysis for pWCRT computation. This is a follow up for the RTSOPS 2017 abstract 'Markov Chain Modeling of Probabilistic Real-Time Systems '. [9] and https://forge.onera.fr/projects/probscheduling are schedulability analysis implementations and theoretical proofs for that. We note that the MDP modeling can interface with any probabilistic schedulability analysis, with or without formal methods, and for discrete or continuous distributions. The use of MDP can benefit form more-than-two criticality levels. Furthermore, by utilizing the notion of rewards in the MDP, it is possible to quantify the cost of a job entering HI-criticality mode. In future work, we want proposing to combine MDP with the satisfiability modulo theories for developing optimal or pseudo-optimal scheduling strategies to control system criticality probability.

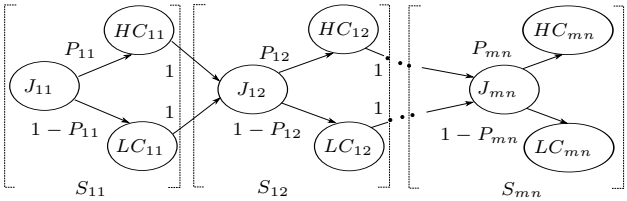



Fig. 1. System MDP model assuming $J_{11}, J_{12}, \ldots, J_{m^{\text{HI}}n^{\text{HI}}}$ the HI-criticality jobs in the hyperperiod for representation.

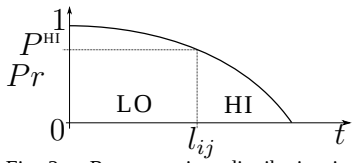



Fig. 2. Response time distribution in its ICDF form of a job $J_{ij}$. High and low criticality regions are separated by $l_{ij}$.

## REFERENCES


[1] S. K. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie. Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems. *J. ACM*, 62(2):14:1–14:33, 2015.

[2] S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS*, pages 34–43, 2011.

[3] A. Burns and R. Davis. Mixed-criticality systems: A review. http://www-users.cs.york.ac.uk/ burns/review.pdf+, 2017.

[4] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. *Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[5] Z. Guo, L. Santinelli, and K. Yang. Edf schedulability analysis on mixed-criticality systems with permitted failure probability. In *21th IEEE International Conference on Embedded and Real-Time Computing System and Applications*, 2015.

[6] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[7] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *IEEE Real-Time Systems Symposium*, 2013.

[8] L. Santinelli, P. Meumeu, D. Maxim, and L. Cucu-Grosjean. A component-based framework for modeling and analyzing probabilistic real-time systems. In *Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2011.

[9] J. Singh, G. Infantes, L. Santinelli, D. Doose, and J. Brunel. Continuous time markov chain modeling for probabilistic schedulability analysis. In *CRTS Workshop at IEEE Real-Time Systems Symposium, RTSS*, 2017.

# Probabilistic parallel real-time tasks model on multiprocessor platform

Slim BEN AMOR
Inria Paris, France, slim.ben-amor@inria.fr

Liliana CUCU-GROSJEAN
Inria Paris, France, liliana.cucu@inria.fr

## I. Introduction

More intelligent and interactive systems expectations increase the need for dedicated embedded electronics in various fields such as avionics, automotive, and telecommunications. Most existing embedded systems have timing requirements and constraints to ensure operational reliability or to maintain a certain level of quality of service with respect to time, making them real-time systems.

Beside timing constraints, many real-time systems present precedence constraints. They are imposed between different processes to ensure the good execution of the data flow. For example, the sensor measurements must be pre-processed and used for further computation. Then, the produced result helps to control the actuators. The order between different computation and tasks guarantees that each function executes on the appropriate data.

Many processor designers incorporate specific architectures and functions that enhance the average performance. For example, multi-core processor and cache memory levels accelerate the execution of programs in general. However, they introduce a larger variability for the execution time and communication delays because of the complex architecture. This variability degrades the performance in the worst case scenario by an increased pessimism. The use of probabilistic approaches may reduce pessimism of schedulability analyses by taking in consideration the variability of system parameters. In this paper we concentrate on the scheduling of real-time task systems with precedence constraints on a multi-core platforms (the most widespread architecture in nowadays).

## II. Task Model

We consider a set of $n$ sporadic parallel real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled according to a fixed-priority and preemptive policy on a multiprocessor system. This system is composed of $m$ uniform processors denoted $\pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ where each processor $\pi_i$ has a speed $s_i$. Each parallel task $\tau_i$ is specified by a 4-tuple $(G_i, O_i, D_i, T_i)$ where $G_i$ is a directed acyclic graph (DAG), and $O_i$, $D_i$ and $T_i$ are positive integers. In fact, each task $\tau_i$ is a recurrent process that releases an infinite sequence of "jobs" $\tau_{i,j},\ j \in \mathbb{N}$. The first job is released after the release offset $O_i$ from the instant zero. While subsequent jobs are released at least after $T_i$ time units. Every job released by $\tau_i$ has to complete its execution within its deadline $D_i$ time units from its release. We assume $D_i \leq T_i$ (constrained deadline).

The internal structure of a task $\tau_i$ is described by the DAG $G_i = (V_i, E_i)$, where $V_i$ is a set of $n_i$ vertices and $E_i \subseteq (V_i \times V_i)$ is a set of directed edges connecting these vertices (it is required that these edges do not form any cycle). Each vertex $v_{i,l} \in V_i,\ l = 1 \ldots n_i$ represents a computational unit that must execute sequentially. This unit, referred as a "subtask", is characterized by a probabilistic worst-case execution time (pWCET) $C_{i,l}$. We assume that the pWCET is already evaluated for each task with another method [1] and it is given by a discrete probability distribution with different possible values for WCET and their corresponding probabilities (see example below).

Example of pWCET distribution:

$$C_{i,l} = \begin{pmatrix} 2 & 3 & 8 \\ 0.5 & 0.3 & 0.2 \end{pmatrix}$$

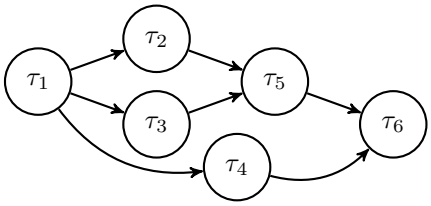



Fig. 1. Example of DAG graph describing precedence constraints

Each directed edge $(v_{i,a}, v_{i,b}) \in E_i$ denotes a precedence constraint between the subtasks $v_{i,a}$ and $v_{i,b}$, meaning that the subtask $v_{i,b}$ cannot start executing before subtask $v_{i,a}$ completes its execution. In this case, $v_{i,b}$ is called a "successor" of $v_{i,a}$, whereas $v_{i,a}$ is called a "predecessor" of $v_{i,b}$. Each subtask could have multiple predecessors and successors. Hence, a subtask is then said to be "ready" if and only if all its predecessors have finished their execution. We call a subtask without

any predecessors or successors, respectively, "source" or "sink" subtask. A direct acyclic graph could have multiple source and sink subtasks.

This task model is coherent with several hardware architectures. It could represent a set of DAG tasks that run on a distributed multiprocessor system where processors belong to different systems and each processor has its own speed $s_i$ (uniform processors). The task model is executed on a distributed architecture according to a partitioned scheme. In this case, if two connected subtasks with precedence constraints are scheduled in different processors, they will induce an inter-processors communication to exchange information.

Moreover, our task model could describe a DAG task set that executes on a multi-core architecture. This architecture contains a cache memory dedicated to every core and a global shared memory. If two dependent subtasks are scheduled on different cores, they will produce communication delays due to the access to the global memory to exchange data between the two subtaks. Even if the two subtasks execute on the same core, a delay may be introduced because of global memory access in case of a cache miss. Hence, the delay caused by precedence edge could be represented by additional subtasks with probabilistic execution time where the probability of the high values is proportional to the cache miss rate.

## III. Motivating Examples

The task model described above could be encountered in several systems on real world applications. For instance, in the design of unmanned aerial vehicle (UAV) for critical mission, the drone has several tasks that include sensor reading, navigation, detection, motors control, etc. These tasks should respect temporal and precedence constraints to ensure safety and stability. They will run on a multi-core embedded processor and they will be affected by the high variation of the execution time caused by inter-tasks interference, cached memory and complex architectures (pipeline, branch prediction, dedicated DSP. . . ). Besides, the drone should communicate with a ground station to send information and receive instructions. The delay of the wireless communication has high variation depending on the distance and the environment conditions. According to our model, this communication step could be seen as subtask on the DAG precedence graph. This subtask has probabilistic execution time which describes well the communication delay variation.

In addition, another application could be described by our task model; The vehicles platoon driving consists of group of cars or trucks that follow a leader vehicle, generally in a highway, using electronics control and wireless communication. Platooning allows to save fuel by reducing air resistance and required safety distance even with high speed driving. In this case, wireless communication become very sensitive to driving speed and distances between vehicles. This sensitivity will introduce an important variation in the communication delays which can be modeled with a probabilistic execution time. When we look at the whole system, we can describe it as distributed multiprocessors system with one processor for each vehicle. These processors are communicating as message passing system via wireless link to exchange data and instructions.

## IV. Open Problems

Similar task models were proposed and resolved in previous work. In [2]–[4] authors work on schedulability of DAG tasks with deterministic parameters on multiprocessor. While [5] studied the schedulability of DAG Tasks in presence of probabilistic execution time on single processor platform. As an extension to these existing model, we consider a probabilistic approach on a multiprocessor system and we could define several open problems according to our task model. For example, in case of a distributed system, we could choose a partitioned scheduling approach that defines which task or subtask will be assigned to a given processor (remote system). Moreover, partitioned and global scheduling policies are two possible choices for multi-core systems with shared memory.

**Problem 1.** We assume that each subtask can be executed only on one processor for a given subtask partitioning. How do we calculate the probabilistic worst case response time (pWCRT) for each task?

**Problem 2.** We assume that each subtask can be executed on any processor. However, a job of a subtask that started its execution on given processor can not migrate. Under these conditions, how do we calculate the probabilistic worst case response time (pWCRT) for each task?

## References


[1] A. Gogonel, C. Maxim, and L. Cucu-Grosjean, "pWCET estimator for real-time systems," in *RTSS 2017 - IEEE Real-Time Systems Symposium*, Paris, France, Dec. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01666342

[2] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *2012 IEEE 33rd Real-Time Systems Symposium*, Dec 2012, pp. 63–72.

[3] J. Fonseca, G. Nelissen, V. Nelis, and L. M. Pinho, "Response time analysis of sporadic dag tasks under partitioned scheduling," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[4] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global edf scheduling of directed acyclic graphs on multiprocessor systems," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS '13. New York, NY, USA: ACM, 2013, pp. 287–296. [Online]. Available: http://doi.acm.org/10.1145/2516821.2516836

[5] S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean, "Schedulability analysis of dependent probabilistic real-time tasks," in *RTNS*, 2016.

# How effective is sensitivity analysis with probabilistic models?

Luca Santinelli, ONERA - DTIS, luca.santinelli@onera.fr

## I. Introduction and motivation

Probabilities are a quite recent modeling paradigm for real-time systems. They offer more flexible representations than classical deterministic-single-value models, by applying multiple values. Each of those values has an associated probability that defines the "confidence" of the value as model. Probabilistic models tents to explore what is beyond the single value worst-case and thus aiming at reducing the pessimism that single value models bring.

The flexibility from probabilistic models can be used into system design, especially in scheduling. The probabilities can be involved into both off-line and runtime decisions to help improving overall performance, especially with soft real-time systems. Actual approaches to probabilistic schedulability analysis have not yet explored such flexibility. This is due to the fact that it is extremely complex to do schedulability analysis with probabilities – all the values from every task have to be combined for reliable results. Existing probabilistic schedulability analyses are empirical solutions and not structured as formal methods; they cannot rely on appropriate mathematical analysis to ease the guarantee of the results. The complexity of the probabilistic schedulability analysis is a big limitation to explore the potential of probabilities. Due to that, it is not yet possible to study the impact that each value in the probabilistic models has on the schedulability, and most of all there are very rare works that apply probabilities into scheduling decisions [6].

The lack of formal-method-based probabilistic schedulability approaches has been the objective of the RTSOPS 2017 abstract 'Markov Chain Modeling of Probabilistic Real-Time Systems'. In there, it has been proposed to use Continuous Time Markov Chains to model jobs executions, with job interactions, and validate the framework with model checking. https://forge.onera.fr/projects/probscheduling and [10] are schedulability analysis implementations and theoretical proof as follow up of the RTSOPS abstract.

**Probabilistic models:** A real-time application is a set $\Gamma$ of tasks, $\Gamma = \{\tau_i\}$. $\Gamma$ it is called probabilistic real-time application whenever at least one of its elements is described with a probabilistic parameter. In this abstract it is assumed to be the probabilistic Worst-Case Execution Time (pWCET), but it can be any task parameter e.g., task minimum inter-arrival time or period. We consider the pWCET as a discrete distribution defined to upper bound any possible execution time the task can have [4]. It composes of multiple values, each with an associated probability; those values are Worst-Case Execution Times (WCET) thresholds for the task, while the probabilities are "confidence" degree on the WCET threshold as the probability that the task execution time is bounded by the WCET threshold.

A task $\tau_i$ of a probabilistic real-time application is the tuple $\tau_i = \{\mathcal{C}_i, T_i, D_i\}$ where $\mathcal{C}_i$ is the pWCET, $T_i$ is the period, and $D_i$ is the deadline of the task; the case of implicit deadline $D_i \leq T_i$ is considered here. The pWCET as a discrete random variable can be represented with the Probability Density Function (PDF) $f_{\mathcal{X}}(x)$, with the Cumulative Distribution Function (CDF) $F_{\mathcal{C}_i}(x) = \sum_0^x f_{\mathcal{C}_i}(y)$, and with the Inverse Cumulative Distribution Function (ICDF) $\overline{F}_{\mathcal{C}_i}(x) = 1 - \sum_0^x f_{\mathcal{C}_i}(y)$ as the the exceeding threshold probability.

For simplicity reasons, we represent the discrete random variable $\mathcal{C}_i$ as two arrays: $\overline{wcet} = \{c_{i1}, c_{i2}, \ldots\}$ for the WCET thresholds, and $\overline{p}_i = \{p_{i1}, p_{i2}, \ldots\}$ for the probabilities with $p_{ij} = f_{\mathcal{C}_i}(c_{ij})$; $P_{ij}$ is the cumulative probability $P_{ij} = 1 - \sum_{k>j} p_{ik}$.

**Resource demand:** We assume the Earliest Deadline First (EDF) scheduling to schedule probabilistic real-time applications; similar reasoning can be done for fixed priority scheduling algorithms. The demand bound function $\mathsf{dbf}_i$ of task $\tau_i$ is the resource requested by $\tau_i$ to fully execute by its deadline. $\mathsf{dbf}_i(t) \stackrel{\text{def}}{=} (\lfloor \frac{t-D_i}{T_i} \rfloor + 1)_0 \times c_{ij}$ and it represents the minimum resource request in order to execute the task by its deadline; $c_{ij}$ is the task WCET. $\mathsf{dbf}_\Gamma$ is the resource demand of the whole task set $\Gamma$: $\mathsf{dbf}_\Gamma \stackrel{\text{def}}{=} \sum_\Gamma \mathsf{dbf}_i$.

With probabilities, it is possible defining multiple demand bound functions for $\tau_i$, depending on the WCET threshold $c_{ij}$ from $\mathcal{C}_i$. $\langle \mathsf{dbf}_i(t, c_{ij}, P_{ij} \rangle$ is a probabilistic demand bound function for $\tau_i$ where to the $\mathsf{dbf}_i$ there is associated the probability $P_{ij}$ as the exceeding probability of the WCET threshold $c_{ij}$ selected. $P_{ij}$ is the probability for $\mathsf{dbf}_i(t, c_{ij})$ to be exceeded at runtime when the task executes for an execution time larger than $c_{ij}$, $P_{ij} = \overline{F}_{\mathcal{C}_i}(c_{ij})$. $\Gamma$ is modeled with a set of probabilistic demand bound functions, each as $\langle \mathsf{dbf}(t, \overline{C}), P \rangle$ from the combination of tasks demand bound functions $\mathsf{dbf}_i$. With $\overline{C} = (c_{1j}, c_{2k}, \ldots)$ the array of WCET threshold values (one for each task), $\mathsf{dbf}(t, \overline{C})$ is computed as the summation of the tasks $\mathsf{dbf}_i(c_{ij})$, and $P$ is the exceeding probability of $\mathsf{dbf}(t, \overline{C})$ such that $P = P_1(c_{1j}) \times P_2(c_{2k}) \times \ldots$[1].

**Resource provisioning:** The computational resource to execute tasks is provided by reservation mechanisms, also known as servers. The resource provisioning $S(t)$ of a server $S$ can be lower bounded in $[0, t]$ with the supply bound function (sbf) $\mathsf{sbf}$: $\mathsf{sbf}_S(t) \stackrel{def}{=} min_{t_0 \geq t} \int_{t_0}^{t_0+t} S(x)dx$ [7]. It is possible defining the $\mathsf{sbf}$ linear approximation $\mathsf{lsbf}$ that lower bounds the resource provisioning in $[0, t]$, $\forall \Delta \;\; \mathsf{lsbf}_S(\Delta) \leq \mathsf{sbf}_S(\Delta)$: $\mathsf{lsbf}(\Delta) \stackrel{def}{=} \max\{0,\ \alpha(\Delta)\}$, with $\alpha \stackrel{def}{=} \lim_{\Delta \to \infty} \frac{\mathsf{sbf}(\Delta)}{\Delta}$ the resource provisioning rate and $\Delta \stackrel{def}{=} \inf\{q \mid \alpha(\Delta - q) \leq \mathsf{sbf}(\Delta)\ \forall \Delta\}$ the longest interval with no resource provisioning [9].

The EDF schedulability of a task set $\Gamma$ within a server $S$ can be guaranteed iff: $\forall \Delta\ \mathsf{dbf}_\Gamma(t) \leq \mathsf{lsbf}_S(t)$; the set of time instances where to check EDF schedulability can be reduced to the set $D$ of deadlines within the task set hyperperiod [1, 7].

**Sensitivity analysis and proposal**: The sensitivity analysis to real-time scheduling aims at investigating the impact that task parameters have on the application schedulability. Classically, the sensitivity analysis applies to deterministic task models in order to study the impact that task parameters have on schedulability, [8, 2]. With some enhancements, it is applicable to probabilistic real-time frameworks and extend the impact evaluation to probabilities.

This abstract is for illustrating possible ways to apply sensitivity analysis with probabilistic models and ease the complexity of actual probabilistic schedulability analysis. We believe that sensitivity analysis would allow exploring the flexibility of the probabilistic models into real-time systems design and development. Moreover, the sensitivity analysis would directly link probabilities to scheduling conditions and effectively apply them into scheduling decisions. This is an open problem since so far there are not proposed effective works on that.

[1]The probability multiplication for $P$ as joint probability is possible due to the worst-case distribution assumption [3]. As $\mathcal{C}_i$ are pWCETs they are independent, then the combination of $\mathsf{dbf}$ is independent. To remind that $P_j$ are cumulative probabilities and their multiplication ends up into cumulative probability.

The abstract illustrates ideas around sensitivity analysis and probabilities which could lead to unexplored contributions for reducing the pessimism and implementing better probabilistic schedulers.

## II. $(\alpha, \Delta)$-SPACE

Given the schedulability condition with demand bound function and $\mathsf{lsbf}$, it exists the $(\alpha, \Delta)$-space where to represent resource provisioning and resource requests/demands, [8]: an application $\Gamma$ can be mapped into the $(\alpha, \Delta)$-space with its feasibility region $\Phi_\Gamma$ as the set of all service supply pairs $(\alpha, \Delta)$ that guarantee the application timing constraints (schedulability). The feasibility region $\Phi_\Gamma$ in case of EDF is such that $\forall\ t \in \mathrm{D}\ :\ \mathsf{dbf}(t) \leq \alpha \cdot (t - \Delta)$, which means that $\forall\ t\ \in\ \mathrm{D}\ :\ \Delta \leq t - \frac{\mathsf{dbf}(t)}{\alpha}$, and $\Delta \leq \min_{t \in \mathrm{D}} \left\{ t - \frac{\mathsf{dbf}(t)}{\alpha} \right\}$. Similarly, with fixed priority and level-$i$ workload, it is possible defining $(\alpha, \Delta)$-space feasibility regions.

With probabilities there exist multiple probabilistic feasibility regions depending on the WCET thresholds $\overline{C}$ applied. Each $\langle \Phi_\Gamma(\overline{C}, P \rangle$ is a feasibility region $\Phi_\Gamma(\overline{C})$ from the WCET thresholds selected $\overline{C}$ and the probability $P$ associated to it. $P$ is the same as the one of $\langle \mathsf{dbf}(t, \overline{C}), P \rangle$ and is the probability of exceeding $\Phi_\Gamma(\overline{C})$; it can be also interpreted as the probability of verifying the condition $\mathsf{dbf}(\cdot, \cdot) \leq \mathsf{sbf}$, thus the 'schedulability probability'.

In $(\alpha, \Delta)$-space, there exists the Euclidean distance $(\delta\alpha = \alpha_2 - \alpha_1, \delta\Delta = \Delta_2 - \Delta_1)$ which defines the distance between two resource supply $\mathsf{lsbf}$; it can be extended with probabilities and used for sensitivity analysis.

With the probabilistic $(\alpha, \Delta)$-space, the sensitivity analysis would:

1) Evaluate the resource demand for probabilistic schedulability answering to: For a specific probability, what are the resource provisioning necessary to guarantee the probabilistic real-time application? Resource evaluation can be applied to develop and guarantee strategies that change resource provisioning and obtain specific probabilistic schedulability levels.

2) Define and explore the resource provisioning-probabilistic schedulability trade-offs. The trade-offs are to explore the effects that the WCET thresholds have on the schedulability and on the resource necessary to achieve certain probabilistic schedulability levels.

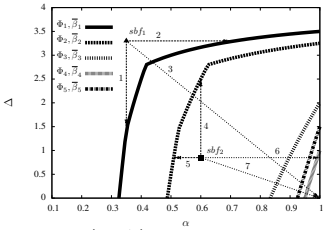



Fig. 1. $(\alpha, \Delta)$-space and the sensitivity analysis that applies to it.

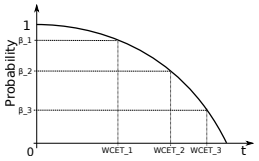



Fig. 2. Task pWCET in its ICDF form with different WCET thresholds from $\beta$.

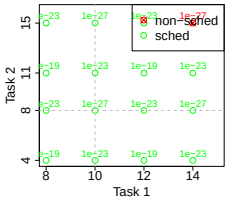



Fig. 3. Probabilistic C-Space 2D representation of a three task probabilistic real-time application.

In Figure 1, an example of $(\alpha, \Delta)$-space for an probabilistic application scheduled under EDF. In there, there are represented 5 feasibility regions as well as 7 possible resource modifications which can be applied in order to achieve the desired probabilistic schedulability level.

We propose to do sensitivity analysis with a parameter $\beta$ applied to pWCETs. $\beta$ defines the cumulative probability at which extract pWCET thresholds from a pWCET. It is possible define a unique $\beta$ for $\Gamma$ or a $\beta_i$ per task $\tau_i$, $\overline{\beta}_i = \{\beta_1, \beta_2, \ldots\}$. $\beta$ is represented in Figure 2 with the multiple possible WCET threshold extracted from a continuous distribution. The more $\beta$ reduces, the more the WCET threshold increases and becomes more "confident" to be the worst-case single-value model. $\beta$ applies the same way to discrete pWCET distributions. In Figure 1 there is an example of that where the 5 feasibility regions are derived for 5 different $\overline{\beta}$. The sensitivity analysis with strategies for changes and the use of $\beta$ is under development. Under definition there are the metrics for quantifying the advantages from probabilistic models (pessimism reduced for different degrees of schedulability). The complexity of the probabilistic analysis and for developing strategies appears reduced with the $(\alpha, \Delta)$.

## III. C-SPACE

The C-space from [2, 5] is another abstract representation for real-time applications on which apply sensitivity analysis.

From probabilistic models (pWCET and probabilistic dbf), it is possible to build the probabilistic version of the C-space, such that each point $\overline{c}$ has a $P$ probability associated. $\overline{c} = \{c_1, c_2, \ldots\}$ is a combination of task WCET thresholds (possible WCET thresholds, one per task, like $\overline{C}$), while the probability $P$ is the probability of exceeding such thresholds $P = P_1(c_{1,j}) \times P_2(c_{2,k}) \times \ldots$. In the probabilistic the C-space, $P$ cannot represent the schedulability probability.

Given the scheduling policy, in the probabilistic C-space there is the feasibility region where every point $\overline{c}$ within the region is a schedulable WCET thresholds configuration. The points outside the region do not represent schedulable WCET thresholds configurations. Figure 3 shows an example probabilistic real-time application with three tasks, and its probabilistic C-space in 2D form; to each point, there is a probability associated.

With the probabilistic C-space it is possible to apply Euclidean distance $(\delta c_1 = c_{1k} - c_{1j}, \delta c_2 = c_{2r} - c_{is}, \ldots)$ combined with probabilities, and evaluate the impact that WCET choices have on the schedulability. The sensitivity analysis for the probabilistic C-space can also work with parameter $\beta$. It is under discussion how to develop effective sensitivity analysis for the probabilistic C-space and with $\beta$. The goal is also to show a reduced complexity of the probabilistic schedulability analysis from the sensitivity analysis with the C-space.

## REFERENCES


[1] S. K. Baruah. Dynamic and static-priority scheduling of recurring real-time tasks. In *Real-Time System*, pages 93–128, 2003.

[2] E. Bini, M. D. Natale, and G. C. Buttazzo. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems*, 39(1-3), 2008.

[3] L. Cucu-Grosjean. Independence - a misunderstood property of and for (probabilistic) real-time systems. Invited paper to the 60th birthday of A. Burns, 2013.

[4] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. *Proceedings of the 25th IEEE Euromicro Conference on Real-Time Systems (ECRTS)*, 2013.

[5] L. George and J. Hermant. Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling. *JACIC*, 6(11):604–623, 2009.

[6] Z. Guo, L. Santinelli, and K. Yang. EDF schedulability analysis on mixed-criticality systems with permitted failure probability. In *21th IEEE International Conference on Embedded and Real-Time Computing System and Applications*, 2015.

[7] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *ECRTS'03, IEEE Computer Society*, pages 151–158, 2003.

[8] L. Santinelli, G. C. Buttazzo, and E. Bini. Multi-moded resource reservations. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 37–46, 2011.

[9] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, page 2. IEEE Computer Society, 2003.

[10] J. Singh, G. Infantes, L. Santinelli, D. Doose, and J. Brunel. Continuous time markov chain modeling for probabilistic schedulability analysis. In *CRTS Workshop at IEEE Real-Time Systems Symposium, RTSS*, 2017.

# A Switch-back Protocol for Task-level Criticality Mode on Mixed-Criticality Systems

Jaewoo Lee*, Hyeongboo Baek†, and Jinkyu Lee†
Dept. of Industrial Security, Chung-Ang University, Republic of Korea*
College of Software, Sungkyunkwan University (SKKU), Republic of Korea†

## I. Introduction

One of trends in embedded systems is toward *Mixed-Criticality (MC) systems*, which integrate multiple components with different criticality levels on a single platform. Their examples are ARINC standard in avionic systems [1] and ISO 26262 standard in automotive systems [2]. The goal of MC systems is to provide different levels of assurance on components of different criticality while achieving efficient resource utilization.

Since Vestal's seminal work [3], most of existing solutions [4]–[10] employ (system-level) criticality mode-switch from low-criticality (LC) mode to high-criticality (HC) mode when a task shows HC behavior (executing for more than its low-confidence Worst-Case Execution Time (WCET) estimate). Upon any task transiting to the HC mode, where all the other tasks show HC behavior, those existing solutions commonly penalize all of the LC tasks, i.e., either by dropping all [4], [6] or degrading all the services offered to them [5], [7]–[10].

Addressing the fact that not all HC tasks necessarily show HC behavior at the same time, recent studies [11]–[13] consider *task-level mode-switch*, where tasks can exhibit HC behavior at different times, independently from each other. Under task-level mode-switch, it is allowed that some HC tasks execute in the HC mode while others remain in the LC mode. This makes it possible to penalize some of the LC tasks selectively in the event of mode-switch, rather than all of them unnecessarily.

Although earlier work considered that mode-switch situation is a very rare case, there is increasing demand applying mixed-criticality system into the domain where mode-switch normally happens [5]. For the domain, developing *switch-back* protocol from HC mode to LC mode is necessary, as well as mode-switch protocol from LC mode to HC mode, which is well studied in the existing work. Baruah et al. [4] introduced a simple switch-back protocol that initializes criticality mode at idle times. There are some recent studies considering switch-back [14], [15], which reduce the time duration of HC mode under system-level mode-switch. The drawback of their approach is the fact that the initialization of criticality mode is possible only when there is no HC behavior inside the system (or component), which potentially degrades the execution of LC tasks due to a larger time duration of HC mode.

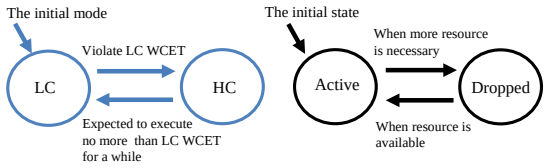
**System Model.** We consider dual criticality: high-criticality (HC) and low-criticality (LC). We consider an MC task set consisting of $n$ MC tasks. Each MC task $\tau_i$ is characterized by $(T_i, C_i^L, C_i^H, \chi_i)$, where $T_i$ is task period, $C_i^L$ is LC WCET, $C_i^H$ is HC WCET, and $\chi_i \in \{HC, LC\}$ is task criticality level. Depending on $\chi_i$, a task is either a LC task or a HC task. The LC and HC utilization of a task $\tau_i$ are defined as $u_i^L \stackrel{\mathsf{def}}{=} C_i^L/T_i$ and $u_i^H \stackrel{\mathsf{def}}{=} C_i^H/T_i$, respectively. For notational convenience, we define $U_L^L \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_L} u_i^L$, $U_H^L \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_H} u_i^L$, and $U_H^H \stackrel{\mathsf{def}}{=} \sum_{\tau_i \in \tau_H} u_i^H$.

We will consider runtime mode of tasks, similar to [13]. For HC tasks, each task has its own task-level criticality mode (*task mode*) (denoted as $M_i$) that indicates its runtime behavior. A task $\tau_i$ is said to be in LC mode ($M_i = \mathsf{LC}$) if no job of the task has executed more than its LC WCET ($C_i^L$), and be in HC mode ($M_i = \mathsf{HC}$) otherwise. Under task mode, we consider task-level mode-switch, where an individual task changes its task mode independently (see Fig. 1(a)). That is, each HC task starts in LC mode, and switches to HC mode when its execution time violates $C_i^L$ (called *mode-switch*). It is also possible that a task can switch from HC-mode to LC mode when its execution time is expected to be less than or equal to $C_i^L$ for a while (called *switch-back*). For LC tasks, we consider runtime execution state of a task (see Fig. 1(b)): each LC task is in either *active* state or *dropped* state. Initially, all LC tasks are active (jobs of the tasks are released sporadically). On mode-switch, some LC tasks are allowed to be dropped in order to support HC tasks with their additional resource requests. When a LC task is dropped, no job of the task is released. On switch-back, some dropped LC tasks are allowed to be re-execute when the system determines that there are enough resources to execute the tasks.

## II. A switch-back protocol to improve the LC performance under task-level criticality mode

**MC-ADAPT Scheduling Framework.** We will develop a switch-back protocol under MC-ADAPT scheduling framework [13]. MC-ADAPT extends EDF-VD [4] under system-level mode-switch and develop a new scheduling algorithm under task-level mode-switch. MC-ADAPT aims to minimally penalize LC tasks by fully reflecting the dynamically changing system behavior into adaptive decision making. MC-ADAPT runtime schedulability analysis can capture the dynamic system state, and MC-ADAPT scheduling algorithm adaptively determines which task to be dropped based on the runtime analysis.

*System state* [13] captures the dynamic system behavior at mode-switch, including the task mode (execution state) of each task: for a given task set $\tau$, a system state $S$ is defined as a four-tuple of disjoint sets: $S = (\tau_{H1}, \tau_{H2}, \tau_{L1}, \tau_{L2})$ where $\tau_{H1}$ is the LC-mode HC task set ($\tau_{H1} = \{\tau_i \in \tau_H | M_i = \mathsf{LC}\}$), $\tau_{H2}$ is the HC-mode HC task set (including the mode-switching




(a) The task mode of a HC task (b) The execution state of an LC task

Fig. 1. The Behavioral Model of an MC Task

task) ($\tau_{H2} = \tau_H \setminus \tau_{H1}$), $\tau_{L1}$ is the active LC task set, and $\tau_{L2}$ is the dropped LC task set (including the dropping LC tasks at mode-switch) ($\tau_{L2} = \tau_L \setminus \tau_{L1}$).

MC-ADAPT can schedule any task set which passes the MC-ADAPT offline schedulability test: $U_L^L + \sum_{\tau_i \in \tau_H} \min(\frac{u_i^L}{x}, u_i^H) \leq 1$ and $xU_L^L + U_H^H \leq 1$. If a task set is accepted by the offline analysis, there are preprocessing steps for MC-ADAPT before runtime scheduling phase: **a)** sort LC tasks in decreasing order of their task utilization; **b)** the VD of each HC task $\tau_i$ is assigned by $V_i = x \cdot T_i$ where $x = \min(1, (1 - U_H^H)/U_L^L)$; and **c)** for each HC task $\tau_i$, the initial task mode of the task is HC mode ($M_i = \mathsf{HC}$) if $C_i^L/V_i > C_i^H/T_i$, and LC mode ($M_i = \mathsf{LC}$) otherwise.

MC-ADAPT schedules the job with the earliest effective deadline and operates under the following rules: **a)** schedule LC tasks with their real deadlines; **b)** for each HC task $\tau_i$, schedule the task with its VD if the task is in LC mode ($M_i = \mathsf{LC}$) and with its real deadline if the task is in HC mode ($M_i = \mathsf{HC}$); **c)** at the mode-switch of a HC task $\tau_i$, set $M_i := \mathsf{HC}$ and drop the LC task with the highest utilization among the active LC task set ($\tau_{L1}$) until the dropped LC task set ($\tau_{L2}$) satisfies the online schedulability test for mode-switch (Eq. (1)):

$$U_{L2}^L \geq \frac{U_{L1}^L + U_{H1}^L/x + U_{H2}^H + U_L^L - 1}{1 - x}. \tag{1}$$

**Switch-back Protocol.** Based on MC-ADAPT framework [13], we will develop a new switch-back protocol. The previous MC-ADAPT employs the simple switch-back protocol used in EDF-VD [4], which initializes task-modes of all HC tasks and execution states of all LC tasks at idle time. The problem of the simple system-level switch-back protocol is larger time duration of HC mode for HC tasks, which potentially degrades the performance of LC tasks. In order to improve the performance of LC tasks, we need to reduce the time duration of HC mode in HC tasks. As fine-grained (task-level) mode-switch mechanism was applied in MC-ADAPT, we can consider fine-grained (task-level) switch-back mechanism.

The first question is how to determine the switch-back of a HC task. A simple approach is to change the task-mode of the HC task whose $k$-th job triggered mode-switch from HC mode to LC mode at the next job release (the release time of the $(k+1)$-th job. If there exists time locality of mode-switch and non-zero mode-switch overheads, the simple approach cannot be a good choice.

The second question is how to develop runtime task resume protocol. The protocol is activated at runtime switch-back situation. Among LO-tasks dropped by previous mode switches, the protocol selects a set of LC tasks to be active. Then, the system changed the execution state of the selected LC tasks from the dropped state to the active state and execute the LC tasks normally. We would like to derive the online schedulability test for switch-back, similar to Eq. (1).

## REFERENCES


[1] *ARINC653 - An Avionics Standard for Safe, Partitioned Systems*. Wind River Systems / IEEE Seminar, 2008.
[2] "ISO/DIS 26262-1 - Road vehicles Functional safety Part 1 Glossary," Tech. Rep., Jul. 2009.
[3] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," in *Real Time System Symposium (RTSS)*, 2007, pp. 239–243.
[4] S. Baruah, V. Bonifaci, G. D"Angelo, H. Li, A. Marchetti-Spaccamela, S. Van der Ster, and L. Stougie, "The Preemptive Uniprocessor Scheduling of Mixed-Criticality Implicit-Deadline Sporadic Task Systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 145–154.
[5] A. Burns and S. Baruah, "Towards A More Practical Model for Mixed Criticality Systems," in *Workshop of Mixed Criticality Systems (WMC)*, 2013.
[6] P. Ekberg and W. Yi, "Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012, pp. 135–144.
[7] O. Gettings, S. Quinton, and R. I. Davis, "Mixed Criticality Systems with Weakly-hard Constraints," in *Real-Time Networks and Systems (RTNS)*, 2015, pp. 237–246.
[8] X. Gu and A. Easwaran, "Dynamic Budget Management with Service Guarantees for Mixed-Criticality Systems," in *Real Time System Symposium (RTSS)*, 2016, pp. 47–56.
[9] M. Jan, L. Zaourar, and M. Pitel, "Maximizing the execution rate of low-criticality tasks in mixed criticality systems," in *Workshop of Mixed Criticality Systems (WMC)*, 2013.
[10] H. Su and D. Zhu, "An Elastic Mixed-Criticality task model and its scheduling algorithm," in *Design, Automation, and Test in Europe (DATE)*, 2013, pp. 147–152.
[11] P. Huang, P. Kumar, N. Stoimenov, and L. Thiele, "Interference Constraint Graph - A new specification for mixed-criticality systems," in *Emerging Technologies and Factory Automation (ETFA)*., 2013, pp. 1–8.
[12] X. Gu, A. Easwaran, K.-M. Phan, and I. Shin, "Resource Efficient Isolation Mechanisms in Mixed-Criticality Scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 13–24.
[13] J. Lee, H. S. Chwa, L. T. X. Phan, I. Shin, and I. Lee, "Mc-adapt: Adaptive task dropping in mixed-criticality scheduling," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, pp. 163:1–163:21, Sep. 2017. [Online]. Available: http://doi.acm.org/10.1145/3126498
[14] J. Ren and L. T. X. Phan, "Mixed-Criticality Scheduling on Multiprocessors Using Task Grouping," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 25–34.
[15] I. Bate, A. Burns, and R. I. Davis, "A Bailout Protocol for Mixed Criticality Systems," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2015, pp. 259–268.

# Resource Augmentation Bounds of EDF for Sporadic Tasks with Constrained Deadlines

Zhishan Guo
Department of Computer Science
Missouri University of Science and Technology, USA
Email: guozh@mst.edu

Xin Han
School of Software Technology
Dalian University of Technology, China
Email: hanxin@dlut.edu.cn

**Abstract**

Even though earliest-deadline-first (EDF) is optimal in terms of uniprocessor schedulability, it is not easy to verify the schedulability of EDF on a uniprocessor for task sets with constrained deadlines—it has been shown that the problem is actually co-NP-hard. The most efficient way to solve this problem in polynomial time is via a partially linear approximation of the demand bound function. On one hand, such an approximation is proven to have a resource augmentation factor no greater than $2 - 1/e \approx 1.632$, where $e$ is the Euler's number. On the other hand, concrete input instances have been provided to show that the lower bound of resource augmentation factors for uniprocessor systems under such approaches is 1.5. This paper studies such strategy and the existing proofs, and present some insights to narrow the gap between the upper and lower bounds of EDF schedulability test with approximate demand bound functions.

## I. Background

Sporadic task sets [1] are widely considered workload model in the real-time systems community in the past several decades. A piece of code (task) $\tau_i$ is characterized by a worst-case execution time (WCET) $C_i$, a minimum inter-arrival separation length (also known as period) $T_i$, and a relative deadline $D_i$. A sporadic task may trigger releases of a number of jobs, where two consecutive such releases should arrive no shorter than the period. The scheduling window of a jobs is determined by its release time and absolute deadline (which is $D_i$ time units after the release time).

It has been shown in [2] that EDF is an *optimal* policy for scheduling a task set upon a uniprocessor in a preemptive manner; i.e., there exists a correct schedule[1] for a given task set if and only if it is correctly scheduled under EDF. The known exact EDF schedulability test[2] for constrained deadline sporadic tasks is via checking the sum of *demand bound functions* ($dbf()$) of all tasks $\tau_i$, as presented in:

$$\forall t : 0 < t \leq lcm\{T_i\} :: \quad \sum_i dbf(\tau_i, t) \leq t, \tag{1}$$

where

$$dbf(\tau_i, t) = max\{0, \lfloor \frac{t - D_i}{T_i} \rfloor + 1\} \times C_i. \tag{2}$$

Although [3] showed that the critical instance is when all tasks release their first job synchronously (at time 0, without loss of generality) and subsequent job arrivals are as rapidly as possible, this is still a pseudo-polynomial time algorithm as Condition (1) must be verified at all time points $t$ within a hyper-period[3].

## II. Preliminary Results

In fact, it is not possible to derive a polynomial-time exact schedulability test unless $P = NP$ since the problem has proven to be co-NP-hard [4]. As a result, approximated polynomial schedulability tests have been derived. Specifically, a linear approximation is proposed in [5], with a special case mentioned in [6]:

$$dbf^*(\tau_i, t) = \begin{cases} 0, & \text{if } t \leq D_i; \\ \left(\frac{t - D_i}{T_i} + 1\right) C_i, & \text{otherwise.} \end{cases} \tag{3}$$

Since $dbf^*(\tau_i, t) \geq dbf(\tau_i, t)$ holds for any $t \geq 0$ and it is a two-piece linear function, as demonstrated in Figure 1, it is obvious that we could use $dbf^*$ instead of $dbf$ in (1) and achieve a sufficient only, yet polynomial time EDF schedulability test.

[1] Under a *correct* schedule, all jobs (released by the tasks) receive enough execution time (up to the WCETs) within their scheduling windows.
[2] Given a certain scheduling policy, the associated *schedulability test* is used to verify whether correctness can be guaranteed under all circumstances.
[3] In practice, it suffices to verify the instances when dbf function changes its value, yet the total number of such time points is still exponentially large.

A *resource augmentation factor* of $\rho$ of an algorithm $A$ guarantees that if a sporadic task set is feasible on $m$ identical processors, the schedule derived from the algorithm $A$ is correct by speeding up the original platform by a factor of $\rho$. In [6], Chen and Chakraborty proved the resource augmentation bound on uniprocessor systems is at most $2 - \frac{1}{e} \approx 1.6322$. Theorem 1 of [6] shows that the resource augmentation factor is at least 1.5.

## III. Insights and On-Going Efforts

We first introduce some techniques used in [6] and then give some new techniques/insights that may be helpful in lowering the upper bound of the resource augmentation factor.

**Normalization.** Given a set of tasks $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ with $D_1 \le D_2 \le \cdots \le D_n$, we can transform it to another set $\tau'$ with property: $dbf^*(\tau, t) = dbf^*(\tau', t)$ for all $t \ge D_n$.

$$C_i' = \left( \left\lfloor \frac{D_n - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i, \quad (4)$$

$$T_i' = \left( \left\lfloor \frac{D_n - D_i}{T_i} \right\rfloor + 1 \right) \cdot T_i, \quad (5)$$

$$D_i' = \left( \left\lfloor \frac{D_n - D_i}{T_i} \right\rfloor \right) \cdot T_i + D_i. \quad (6)$$

The transformation first occurred in [6], here we call it as a normalization. The following results are from [6] and are also demonstrated in Figure 1: $dbf(\tau, t) \ge dbf(\tau', t)$, $dbf^*(\tau, D_n) = dbf^*(\tau', D_n)$ and ${D_n}' < {D_i}' + {T_i}'$ for each ${\tau_i}'$.

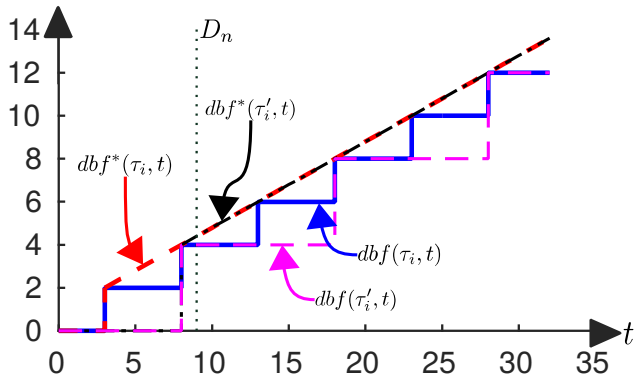



Fig. 1. An example for task transformation and dbf modifications, with task parameters of $C_i = 2, D_i = 3, T_i = 5$, and $D_n = 9$.

**Fixing Deadlines.** Previous work only considered the set of $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ with $D_n < D_i + T_i$ for $1 \le i \le n$.

Assume that $dbf(\tau, t) \le t$ for all $t > 0$ and $\sum_{i=1}^{n} u_i \le 1$, where $\frac{C_i}{T_i} = u_i$. To maximize $dbf^*(\tau, D_n)$, it is proved that there is an optimal solution $\tau$ with the following property: $D_i = \sum_{j=1}^{i} C_j$.

**Analyzing.** They relaxed the constraints $D_i + T_i > D_n$ to $D_i + T_i = D_n$, then there is only one knapsack constraint: $\sum_{i=1}^{n} u_i \le 1$, previous work solved the problem by a greedy method, i.e, $u_i$ gets its maximal value if possible, otherwise $u_i = 0$ by setting $T_i = +\infty$. Such scheme lead to an resource augmentation bound of 1.6322.

**Weakness.** Following the above analysis we found that $dbf(\tau, D_n)$ may get too much larger than $D_n$, which violates the constraint $dbf(\tau, D_n) \le D_n$.

**Observation 1**: Assume $T_1$, $T_2$ are feasible on uniprocessor and each task in $T_2$ has the same execution time. Then $\lim_{|T_1| \to \infty} \sup_{T_1} dbf^*(T_1, D_n) \le \lim_{|T_2| \to \infty} \sup_{T_2} dbf^*(T_2, D_n)$.

**Observation 2**: Assume $T$ is a set of tasks with property: for each task we have $dbf(\tau_i, t) \le t$, where $0 \le t \le D_i + T_i$. Then $\sup_T dbf^*(T, D_n) \le \frac{14 D_n}{9}$.

**Our Result.** We recently proved that the resource augmentation factor of uniprocessor EDF schedulability with sporadic tasks is at least 14/9, which is much close to the lower bound 1.5 given in [6]. We hope eventually an upper bound of 1.5 is proved following the aforementioned insights, and thus the problem can be closed.

## Acknowledgment

The authors would like to thank Prof. Sanjoy Baruah from Washington University at St. Louis and Prof. Xingwu Liu at Chinese Academy of Sciences for the fruitful discussions. Work supported by NSF Award: CNS 1755965.

## References

[1] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[2] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[3] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Real-Time Systems Symposium, 1990. Proceedings., 11th*. IEEE, 1990, pp. 182–190.

[4] F. Eisenbrand and T. Rothvoß, "Edf-schedulability of synchronous periodic task systems is conp-hard," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 2010, pp. 1029–1034.

[5] K. Albers and F. Slomka, "An event stream driven approximation for the analysis of real-time systems," in *Real-Time Systems, 2004. ECRTS 2004. Proceedings. 16th Euromicro Conference on*. IEEE, 2004, pp. 187–195.

[6] J.-J. Chen and S. Chakraborty, "Resource augmentation bounds for approximate demand bound functions," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*. IEEE, 2011, pp. 272–281.

# Priority Assignment in Fixed Priority Pre-emptive Systems with Varying Context Switch Costs

Robert I. Davis
University of York, UK
Email: rob.davis@york.ac.uk

Sebastian Altmeyer
University of Amsterdam (UvA), Netherlands
Email: altmeyer@uva.nl

Alan Burns
University of York,UK
Email: alan.burns@york.ac.uk

**Abstract**

This paper discusses the open problem of priority assignment for systems using Fixed Priority Preemptive Scheduling (FPPS), where the context switch costs are dependent on whether or not the preempting and the preempted tasks belong to the same process and hence share the same address space. Schedulability tests for such systems are not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm. We pose the question: Can optimal (or close to optimal) priority assignments be found efficiently, ideally in a polynomial number of schedulability tests, avoiding the need to check all $n!$ possible priority orderings?

## I. Introduction

The relevant safety standards (IEC61508, DO-178C, ISO26262) for electronics, avionics, and automotive systems require that either all applications are developed to the standard required for the highest criticality application, or that independence between different applications is achieved in both the spatial and temporal domains. One approach to ensuring spatial isolation is to make use of the concepts of *processes* and *threads*, where each process has a separate memory address space. Using a single process for all high criticality applications provides a single memory address space, easing the costs of interaction between high criticality tasks, which are implemented as threads within that process. Low criticality applications and their tasks can similarly be mapped to another distinct process and its threads. This ensures that tasks in low criticality applications cannot corrupt the data or memory space used by high criticality applications. Alternatively, individual applications may each be mapped to a distinct process, providing spatial isolation between applications of the same criticality.

The use of processes and threads gives rise to varying context switch costs [1]. Switching threads within a process (i.e. the context switch between tasks of the same application) has a low cost, since this involves switching only the resources unique to the threads, for example the processor state (program counter, stack pointer, processor registers etc.), which can typically be done in a very short time and may be assisted by hardware support. By contrast, switching between processes (i.e. the context switch between tasks of different criticality applications) may have a much higher cost. It involves switching the resources related to the processes. In particular, switching the memory address space, and can also involve operations on the caches [2], and the Translation Lookaside Buffer (TLB), making process switches a much more costly operation.

In a recent paper [3] at RTAS 2018, Davis et al. derived three flavors of schedulability analysis for FPPS accounting for differing context switch costs, referred to as *simple*, *refined*, and *multi-set* analysis. Here we focus on the *refined* analysis.

## II. System Model

The system model assumed is an extension of the classical sporadic task model. We are interested in tasks executing under FPPS on a single processor. Each of the $n$ tasks $(\tau_1, \tau_j, \ldots, \tau_n)$, is assigned a unique priority. Each task is characterized by its relative deadline $D_i$, worst-case execution time $C_i$, and minimum inter-arrival time or period $T_i$. Tasks are assumed to have constrained deadlines ($D_i \leq T_i$). A task is schedulable if its worst-case response time $R_i$ is less than or equal to its deadline ($R_i \leq D_i$). Each task is assumed to belong to an application mapped to a specific process and hence a specific address space. $A_i$ indicates the address space that task $\tau_i$ is mapped to. If tasks $\tau_i$ and $\tau_j$ belong to the same process and address space, then $A_i = A_j$, otherwise $A_i \neq A_j$. We assume that a context switch from one task $\tau_i$ to another $\tau_j$ has a large cost $C^C$ if it involves switching process and address space (i.e. when $A_i \neq A_j$), and a small cost $C^S$ otherwise.

## III. Analysis

The simple analysis presented in [3] makes the assumption that all context switches incur the large context switch cost. It is thus equivalent to the standard response time analysis for FPPS [4], [5] with the large context switch cost subsumed into each task's WCET. In reality, however, the context switch time depends on both the preempting task and the preempted task. Taking this information into account, the standard analysis is *refined* in [3] as follows. Note we assume that the first job in the busy period always experiences a large context switch time, since the previously running job may be associated with a different process and address space. (We assume that soft real-time tasks may run in a background process at the lowest priority).

$$R_i = C_i + C^C + \sum_{\forall j \in \text{hp}(i)} \left\lceil \frac{R_i}{T_j} \right\rceil (C_j + \gamma_{i,j}) \text{ where } \gamma_{i,j} = \begin{cases} C^C & \text{if } \exists h \in \text{aff}(i,j) | A_h \neq A_j \\ C^S & \text{otherwise} \end{cases} \quad (1)$$

Where $\text{hp}(i)$ denotes the set of tasks with priorities higher than that of task $\tau_i$, and $\text{aff}(i,j)$ denotes the set of *affected* tasks that can execute between the release and completion of task $\tau_i$ and also be preempted by higher priority task $\tau_j$, these tasks have

priorities higher than or equal to that of task $\tau_i$, but lower than that of task $\tau_j$. The term $\gamma_{i,j}$ equates to a large context switch time only if there is some task $\tau_h$ that can execute during the busy period (i.e. the response time) of task $\tau_i$, be preempted by task $\tau_j$, and belongs to a different process and address space to $\tau_j$.

It is easy to construct examples with three tasks $\tau_A$, $\tau_B$, and $\tau_C$ with $A_A = A_C$ and $A_A \neq A_B$ showing that the worst-case response time of task $\tau_C$ depends upon the relative priority order of tasks $\tau_A$ and $\tau_B$; such an example is given in [3]. This means that Deadline Monotonic (DM) priority assignment [6] is no longer optimal. Further, the analysis is not compatible with Audsley's Optimal Priority Assignment (OPA) algorithm [7], since the dependence on the relative priority ordering of higher priority tasks breaks a *necessary* condition for the applicability of Audsley's algorithm [8].

## IV. Open Problems

A priority assignment algorithm or policy $P$ is said to be *optimal* with respect to a schedulability test $S$ and a given task model, if and only if there are no task sets that are compliant with the task model that are deemed schedulable by test $S$ using another priority assignment policy, that are not also deemed schedulable by test $S$ using policy $P$.

We are interested in finding optimal priority assignments for systems with context switch costs that depend on whether the preempting and the preempted task belong to the same process, and so share a common address space. Here, each task belongs to one of two (or more) distinct processes, and the analysis used is the refined test for FPPS given above (or alternatively, the multi-set analysis given in [3]). In each case, an optimal priority assignment could be found by exploring all $n!$ possible priority orderings; however, such an approach becomes intractable even for relatively small task sets (e.g. for $n = 15$, $n! > 10^{12}$). Rather, we are interested in efficient methods of finding an optimal (or close to optimal) assignment; ideally with complexity that is polynomial in the number of schedulability tests.

*Priority assignment toolkit and ideas*

In prior work on priority assignment for fixed priority systems, a number of techniques have proven useful:

(i) Establishing the properties of a priority ordering by considering if schedulability is maintained when the priorities of particular tasks are swapped, for example swapping tasks that have adjacent priorities but are out of DM order [9].

(ii) Work on Robust Priority Assignment [10] has established certain properties (such as the optimality of DM priority ordering) that hold in the presence of general forms of additional interference. It can be useful to disregard subsets of tasks, representing them only as additional interference, to enable a simpler form of reasoning about the optimal priority ordering of the tasks that remain.

(iii) Simple sufficient tests (using only the large context switch costs) and simple necessary conditions (using only the small context switch costs) that are compatible with Audsley's algorithm can be used to guide priority assignment [11]. These techniques enable partial assignments to be found that are certain to be schedulable, while discarding others that are certain to be unschedulable.

As an initial idea, if we could show that if a schedulable priority ordering exists, then a revised ordering with all of the tasks belonging to each specific process in DM partial order is also schedulable, then reasoning along these lines might reduce the overall problem to one of merging DM partial orders for each process – potentially a much simpler problem.

For more background information on techniques for priority assignment in fixed priority systems, see the review on this topic [9]. Finally, we note that solutions to the problems posed may lead to improved solutions to the more complex problem of priority assignment for FPPS with Cache Related Preemption Delays (CRPD) [12] [13].

## References


[1] S. Yamada and S. Kusakabe, "Effect of context aware scheduler on TLB," in *Proceedings IEEE International Symposium on Parallel and Distributed Processing*, April 2008, pp. 1–8.

[2] J. Whitham, N. C. Audsley, and R. I. Davis, "Explicit reservation of cache memory in a predictable, preemptive multitasking real-time system," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 4s, p. 120, 2014.

[3] R. I. Davis, S. Altmeyer, and A. Burns, "Mixed criticality systems with varying context switch costs," in *Proceedings IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2018.

[4] N. C. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[5] M. Joseph and P. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, May 1986.

[6] J.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, no. 4, pp. 237 – 250, 1982.

[7] N. Audsley, "On priority assignment in fixed priority scheduling," *Information Processing Letters 79(1)*, pp. 39–44, May 2001.

[8] R. I. Davis and A. Burns, "Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems," in *Real-Time Systems, Volume 47, Issue 1*, 2010, pp. 1–40.

[9] R. I. Davis, L. Cucu-Grosjean, M. Bertogna, and A. Burns, "A review of priority assignment in real-time systems," *Journal of Systems Architecture*, vol. 65, pp. 64 – 82, 2016.

[10] R. I. Davis and A. Burns, "Robust priority assignment for fixed priority real-time systems," in *Proceedings IEEE International Real-Time Systems Symposium (RTSS)*, Dec 2007, pp. 3–14.

[11] ——, "On optimal priority assignment for response time analysis of global fixed priority pre-emptive scheduling in multiprocessor hard real-time systems," University of York,, Tech. Rep., April 2010.

[12] S. Altmeyer, R. I. Davis, and C. Maiza, "Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems," *Real-Time Systems*, vol. 48, no. 5, pp. 499–526, 2012.

[13] H. N. Tran, F. Singhoff, S. Rubini, and J. Boukhobza, "Addressing cache related preemption delay in fixed priority assignment," in *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015, pp. 1–8.

# Open Problems in FIFO Scheduling with Multiple Offsets

Mitra Nasri[1†], Robert I. Davis[2], and Björn B. Brandenburg[1]
[1] Max Planck Institute for Software Systems (MPI-SWS)
[2] University of York, UK, and Inria, France

## I. Introduction

*First-In-First-Out* (FIFO) is a widely used scheduling policy that can easily be implemented in hardware or software. With FIFO scheduling, the order in which jobs of tasks are executed depends only on their release times. This means that there are no preemptions, since no future job can have a higher priority than one that has already been released. Since tasks are not preempted, their *worst-case execution times* (WCETs) can be estimated with a higher degree of accuracy [1]. Further, on a uniprocessor, non-preemptive execution means that a running task has exclusive access to shared resources. This allows FIFO scheduling to reduce both design complexity and implementation overheads [2]. FIFO scheduling is also *sustainable* w.r.t. a reduction in execution times, i.e., if a task set is schedulable under the FIFO policy when all jobs exhibit their WCETs, then the system remains schedulable even if some jobs require less execution time [3]. As a result, in the absence of release jitter, it is possible to use a *simulation-based* schedulability test to obtain exact response-time analysis for periodic tasks under FIFO scheduling.

The key issue with FIFO scheduling is that it does not take task deadlines into account and so is ineffective at meeting time constraints [3]. This is because once a task enters the FIFO queue, its position in the queue cannot be changed even if a more urgent task is released later. FIFO scheduling of a hard real-time system therefore usually implies a severe under-utilization of the processing resource [2]. In an attempt to improve schedulability, Altmeyer et al. [3] proposed a method where offsets are chosen randomly until an assignment is found that is schedulable using FIFO scheduling; however, this solution does not scale well beyond a few tasks or to task sets with a high utilization [2]. We note that this is an example of the *classic* offset assignment problem, where the goal is to find an offset for each task such that the task set becomes schedulable by the given scheduling algorithm. In this case, the initial offset chosen for each task affects both the release times and the absolute deadlines of its jobs.

Recently, Nasri et al. [2] introduced a novel and different way of using offsets. Here, the initial offsets are assumed to be fixed, and hence the absolute deadlines for the jobs are also fixed. With the approach of Nasri et al. [2], jobs are partitioned into different groups and each group has a different additional relative offset applied. These relative offsets affect only the release times of the jobs, but not their deadlines. Fig. 1-(a) shows a FIFO schedule using multiple offsets. Note that this task set is not schedulable by any work-conserving policy if each task has only one offset (see Fig. 1-(b)). This example illustrates that going beyond one offset per task can yield significant gains in schedulability [2].

Next, we precisely define the relative offset assignment problem with multiple offsets, as explored by Nasri et al. [2]. We assume that a periodic task set is represented by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where each task $\tau_i$ is characterized by its period $T_i$, deadline $D_i$, WCET $C_i$, and initial offset $O_i$. In order to express multiple offsets for a task, we use offset pairs denoted by $OP = (k, o)$, where $k$ is the job index (from the start of the hyperperiod) from which a new offset $o$ is applied. For example, an offset pair $(3, 12)$ means that starting from the third job in the hyperperiod, each job of the task has an offset of $12 + O_i$, until this is changed by a later offset pair (if any). More precisely, the release time and deadline of the $j^{th}$ job of task $\tau_i$ with relative offset $o_{i,x}$ are $r_{i,j} = O_i + (j-1) \cdot T_i + o_{i,x}$ and $D_{i,j} = O_i + (j-1) \cdot T_i + D_i$ respectively. (Note that the relative offset does not alter the deadline).

## II. Challenges and Open Problems

**Open Problem 1:** Given a periodic task set $\tau$, for each task $\tau_i$ (characterized by $C_i$, $T_i$, $D_i$, $O_i$) find a set of offset pairs $\hat{O}_i = \langle (k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i}) \rangle$ such that the resulting task set is schedulable using FIFO scheduling.

Open Problem 1 comes with several challenges. Firstly, this problem is strongly NP-Hard since any general solution could also be used to solve the non-preemptive scheduling problem for periodic tasks, which is known to be strongly NP-hard [4]. Secondly, iterative approaches that attempt to find offsets for tasks one after another cannot easily be applied since changing the offset of one task may change the alignment of the releases of that task w.r.t. all other tasks, resulting in a totally different and potentially infeasible schedule [2]. This issue is more important for non-preemptive scheduling, since due to the blocking effect a change in the schedule of a lower-priority task affects both higher and lower priority tasks as well as the next job of the task itself. This makes the offset-assignment problem more difficult than the commonly studied preemptive case under fixed-priority scheduling. Interestingly, in the case of sets of periodic tasks with non-harmonic periods, it may not be possible

† The first author is supported by a fellowship from the Alexander von Humboldt Foundation.

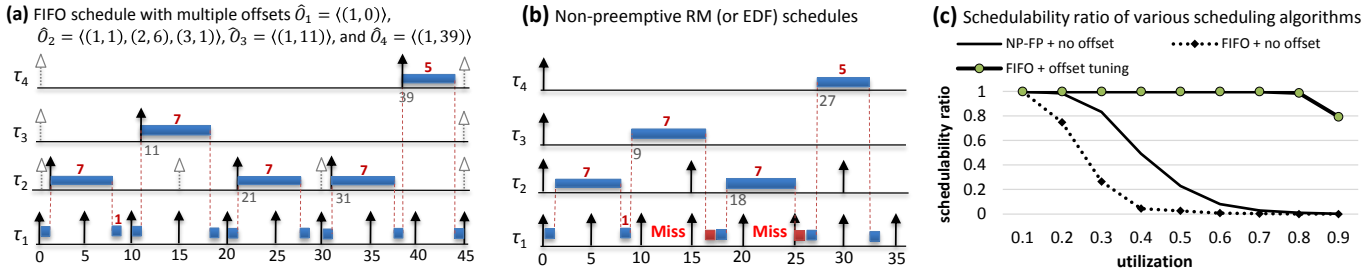



Fig. 1. **(a)** shows a FIFO schedule with multiple offsets, **(b)** shows a non-preemptive rate-monotonic schedule for the same task set, and **(c)** shows a comparison between the schedulability ratio of various algorithms according to the results of Nasri et al. [2] for task sets similar to those used in automotive systems.

to find a single offset per task that ensures schedulability. Fig. 1-(b) shows an example task set that can be scheduled with two offsets for a task, but is infeasible with only one.

An extreme solution would be to assign one offset to each job of a task. In that case, the problem reduces to one of finding a feasible non-preemptive schedule, since the offset of each job can point to its start time in the non-preemptive schedule. This approach, however, requires a large amount of memory to store the offset pairs, which leads to the following open problem.

**Open Problem 2:** Solve Open Problem 1 such that the total number of offset pairs is minimized, i.e., $\min \sum_{i=1}^{n} |\hat{O}_i|$.

Recently, we have proposed a heuristic for solving Open Problem 1 [2], which is based on a reverse method, i.e., instead of proposing an algorithm to find the offset pairs while *building the schedule*, our method starts from a given feasible *reference schedule* and tries to assign offset pairs such that the FIFO scheduler mimics the given reference schedule. This technique, which is called *offset tuning*, finds a small set of offsets such that the resulting FIFO schedule becomes *equivalent* to the reference schedule, i.e., the resulting FIFO schedule has the same *job ordering* as the reference schedule and no job in the FIFO schedule finishes later than the corresponding job in the reference schedule. Thus, offset tuning guarantees FIFO schedulability by design, provided that the reference schedule is feasible. Offset tuning has $O(M \log M)$ computational complexity, where $M$ is the total number of jobs in the hyperperiod. In order to obtain high levels of schedulability, we used the non-preemptive, non-work-conserving *Critical-Window EDF* (CW-EDF) [5] scheduling algorithm to produce the reference schedule. This combination enables high schedulability to be achieved while retaining the low runtime of FIFO scheduling with only a small added memory footprint [2]. Fig. 1-(c) compares the schedulability ratio of FIFO, *non-preemptive fixed-priority* (NP-FP) with *rate-monotonic* priorities, and FIFO with offset tuning (FIFO-OT). It is worth noting that the offset tuning method [2] greedily minimizes the number of offsets used for each individual task; however, this does not necessarily result in the minimum number for the task set as a whole. The proposed solution thus does not optimally solve Open Problem 2, even though empirically it tends to perform well [2]. Further, the performance of the offset-tuning heuristic for other types of reference schedules (e.g., obtained from an ILP solver) is unclear and may be much less efficient.

One possible extension is to consider systems in which tasks exhibit release jitter. Release jitter, which can arise due to timer interrupts, interrupt latency, network delays, etc., can potentially change the order of jobs in the FIFO queue at runtime. Such runtime unpredictability renders Open Problem 1 much more challenging, since the relative offsets used must make the resulting FIFO schedule consistent (or at least schedulable) for any combination of release jitter for all of the different tasks.

**Open Problem 3:** Given a periodic task set $\tau$, for each task $\tau_i$ (characterized by $C_i$, $T_i$, $D_i$, $O_i$, $J_i$) find a set of offset pairs $\hat{O}_i = \langle(k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i})\rangle$ such that the resulting task set is schedulable using FIFO scheduling.

Another possible extension is to relax the constraints that the initial offsets $O_i$ are fixed. This leads to an open problem that builds upon the classic offset assignment problem.

**Open Problem 4:** Given a periodic task set $\tau$, for each task $\tau_i$ (characterized by $C_i$, $T_i$, $D_i$) find an initial offset $O_i$ and a set of offset pairs $\hat{O}_i = \langle(k_{i,1}, o_{i,1}), \ldots, (k_{i,m_i}, o_{i,m_i})\rangle$ such that the resulting task set is schedulable using FIFO scheduling.

## REFERENCES


[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 3, pp. 36:1–36:53, 2008.

[2] M. Nasri, R. I. Davis, and B. Brandenburg, "FIFO with Offsets: High Schedulability with Low Overheads," in *RTAS*, 2018, pp. 271–282.

[3] S. Altmeyer, S. Sundharam, and N. Navet, "The case for FIFO real-time scheduling," University of Luxembourg, Tech. Rep., 2016.

[4] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *RTSS*, 1991, pp. 129–139.

[5] M. Nasri and G. Fohler, "Non-work-conserving non-preemptive scheduling: motivations, challenges, and potential solutions," in *ECRTS*, 2016, pp. 165–175.

# DAG Scheduling Algorithm Considering Large-scale Calculation Tasks Using Many-core Architecture

Yuto Kitagawa
Graduate School of Engineering Science
Osaka University, Japan

Takuya Azumi
Graduate School of Science and Engineering
Saitama University, Japan

## I. Introduction

In recent years, computing platforms that support embedded systems have become increasingly multicore and many-core because such embedded systems have become automated and increased in terms of size and complexity. For example, many-core hardware for embedded systems including Kalray MPPA-256 [1] and Tilela TILE-Gx [2] is suitable for large-scale calculations and therefore the primary solution for implementing today's embedded systems. Such many-core hardware for embedded systems can run an application of the autonomous driving system [3]. In general, these systems are good at running different applications simultaneously because they have multiple instruction, multiple data (MIMD) architectures.

The autonomous driving system includes such a variety of different applications (i.e. localization and object recognition) using a variety of data (e.g. onboard cameras, GPS, infrastructure-to-vehicle (I2V), vehicle-to-vehicle (V2V)). In these applications, there are multiple tasks and multiple dataflows between the tasks. In its entirety, the autonomous driving system can be described as a directed acyclic graph (DAG) by modeling tasks as nodes and dataflows as edges. Furthermore, each task has a potentially different deadline; therefore, the problem of meeting each deadline can be expressed as a DAG scheduling problem. To meet deadlines, shortening the schedule length (i.e. makespan) of the entire DAG is a powerful approach, however, scheduling a large number of tasks on multiple processors to minimize the overall scheduling length is recognized as an NP-complete optimization problem. Therefore, heuristics are utilized to obtain acceptable near-optimal solutions.

Most heuristic scheduling algorithms are based on list scheduling [4], [5]. List scheduling consists of two phases, i.e. (i) a task prioritizing phase in which tasks in a DAG are listed in order by their priority and (ii) a processor-selection phase in which tasks with the highest priorities are scheduled. List scheduling provides high-quality scheduling and is generally accepted owing to its low complexity. Of particular relevance here, the research work presented in [4] and [5] proposes DAG scheduling algorithms for automotive applications; however, these algorithms do not consider processing tasks that require a large amount of computation, for example, image processing algorithms. To ensure quick response times, these tasks must be processed using separate hardware resources, including graphics processing units (GPUs) and many-core processors.

Given the above state-of-the-art and corresponding limitations, we study a DAG scheduling algorithm that considers processing tasks that require a large amount of computation. As expected, such tasks are offloaded to large-scale computing resources for processing. The DAG scheduling algorithm studying now is designed specifically for the many-core Kalray MPPA-256. Taking advantage of the characteristics of a DAG, offloading is modeled, and task scheduling is performed.

## II. System model

### A. Target many-core hardware

The Kalray MPPA-256 processor [1] is based on an array of computing clusters (CCs) and I/O subsystems (IOSs) that are connected to network-on-chip (NoC) nodes using a toroidal two-dimensional topology. The MPPA many-core chip integrates 16 CCs and four IOSs via NoC nodes. In addition, there are 16 processing elements in one CC, and are four processing elements in one IOS. Kalray MPPA-256 architecture is shown in Fig. 1.

### B. DAG notations

An application used for automotive embedded systems can be represented by a DAG as shown in the left side of Fig. 2 [5]. In the figure, sources that generate sensor data include on-board sensors, such as cameras, radar, and GPS systems, or external communication systems, such as V2V and I2V communication systems. Various applications also use stream processing results, e.g., collision warnings, automotive navigation systems, and vehicle-infrastructure cooperative right-turn collision caution signals. These streaming processes can be represented as shown in the right side of Fig. 2, because they can be represented by nodes and edges of a DAG. Localization, distance calculation, and collision judgment of Fig. 2 on the left contain multiple tasks. Therefore, the number of nodes is large as shown in Fig. 2 on the right. In general, a directed edge

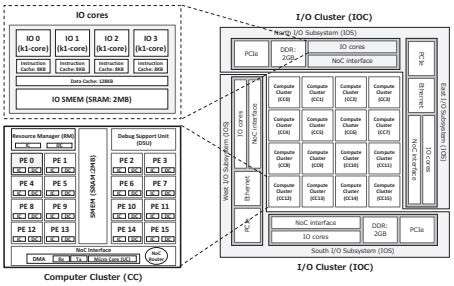



Fig. 1. Illustrating the architecture of Kalray MPPA-256.

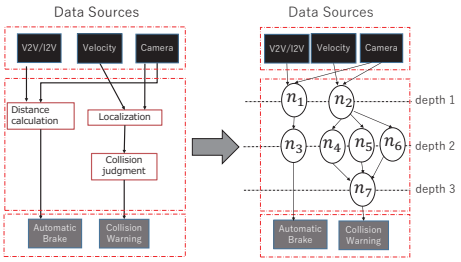



Fig. 2. Illustrating the structure design of an automotive embedded system (left), as well as its corresponding DAG representation (right).

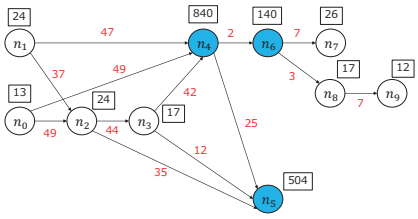



Fig. 3. An example of a target DAG.

from node A to node B indicates that task A has execution precedence over task B. Further, a DAG is periodic and its period is same as a hyper period of on-board sensors.

## III. Problem Description

An example of a DAG to be studied is shown in Fig. 3. Numbers enclosed in rectangles near each node represent required calculation times and red numbers represent communication times. The calculation times of the blue nodes are large, thus requiring parallel calculation; as noted previously, these nodes must be offloaded and processed. Communication time occurs when two nodes are placed on different cores. From now on, we call a node that does not need parallel computation as a non-parallel node and call a node that needs parallel computation as a parallel node. We assume that the parallelizable percentage of parallel nodes is not 100 percent. That is, there are parts that must be calculated sequentially. This is known as Amdahl's law. In addition, we assume the parallelizable percentage of non-parallel nodes is zero percent. Under these assumptions, we consider a scheduling algorithm that processes both parallel and non-parallel nodes with Kalray MPPA-256.

**Open Problem 1:**

How much computing resources should be given to parallel nodes in consideration of Amdahl's law. Considering Amdahl's law, if the computing resources are properly allocated to the parallel nodes, the entire processing time of the DAG will be shortened. In order to solve this problem, we are planning to determine the number of core allocation using two parameters, the position of the parallel node to the DAG and the computation time. In addition, a core allocation must be performed in consideration of an unique architecture of Kalray-MPPA 256.

**Open Problem 2:**

After determining the number of cores allocated for the parallel node, we decide the start and end time of node processing. We let parallel nodes process with cores in CCs, and let non-parallel nodes process with cores in IOSs because it is easy to offload to CC cores as the topology shows. We are trying to solve this scheduling problem using list-scheduling which is a heuristic method because a problem of allocating multiple processors to multiple nodes considering the communication delay is known as NP complete problem.

**Open Problem 3:**

We assume that a DAG task acquires data from multiple on-sensors and its period is same as a hyper period of on-board sensors. Therefore, the DAG task must be scheduled under multirate periodic conditions. We would like to propose a scheduling method by giving the multirate period as a parameter to tasks that a DAG has. Pipeline processing is one of the solutions for DAG scheduling with multirate period. We would like to discuss how to incorporate it into our proposed scheduling algorithm.

## IV. Acknowledgement

This work was partially supported JST PRESTO Grant Number JPMJPR1751 and Hitachi, Ltd.

## References

[1] Y. Maruyama, S. Kato, and T. Azumi, "Exploring Scalable Data Allocation and Parallel Computing on NoC-Based Embedded Many Cores," in *Proc. of IEEE International Conference on Computer Design (ICCD)*, 2017, pp. 225–228.

[2] A. Munir, S. Ranka, and A. Gordon-Ross, "High-performance energy-efficient multicore embedded computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 4, pp. 684–700, 2012.

[3] Shinpei, Kato and Shota, Tokunaga and Yuya, Maruyama and Seiya, Maeda and Manato, Hirabayashi and Yuki, Kitsukawa and Abraham, Monrroy and Tomohito, Ando and Yusuke, Fujii and Takuya, Azumi, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," in *Proc. of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS2018)*, 2018.

[4] J. Rho, T. Azumi, M. Nakagawa, K. Sato, and N. Nishio, "Scheduling Parallel and Distributed Processing for Automotive Data Stream Management System," *Journal of Parallel and Distributed Computing*, vol. 109, pp. 286–300, 2017.

[5] N. Mayo, T. Azumi, K. Yuto, and N. Nishio, "HVMD_CC : Heterogeneous Value with Multiple Deadlines and Communication Contention," in *Proc. of International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS 2017)*, 2017, pp. 13–20.

# Hard Instances of Mixed-Criticality Scheduling

Kunal Agrawal
Washington University in St. Louis
Email: kunal@wustl.edu

Sanjoy Baruah
Washington University in St. Louis
Email: baruah@wustl.edu

**Abstract**

## I. Introduction

In *mixed-criticality* systems, tasks with different criticality levels share a computing platform and demand different levels of assurance in terms of real-time performance. For mixed-criticality systems, the Vestal model [1] has been studied extensively (see [2] for a survey). It is known that mixed-criticality scheduling is a difficult problem and strictly more difficult than traditional (non mixed-criticality) real-time scheduling. In particular, given a collection of jobs with deadlines, it is straightforward to check if this collection of jobs is schedulable on a single processor — one can just check if EDF can schedule the jobs.[1] On the other hand, it is known [3, Theorem 1] that determining whether a given collection of independent mixed-criticality jobs is feasible on a uniprocessor is NP-hard in the strong sense. Similarly, non mixed-criticality implicit deadline sporadic task sets are schedulable on uniprocessors using EDF as long as their total utilization is smaller than 1 — a fact that is easy to check in linear time. However, it was recently shown that it is NP-Hard to check if a collection of dual-criticality implicit deadline tasks are schedulable on a single processors [4].

We would further like to characterize the structural differences and/or similarities between non-mixed criticality task systems and mixed-criticality task systems — in particular, we want to understand whether periodic or sporadic instances are the "hard instances" when it comes to mixed-criticality scheduling on uniprocessors.

## II. System Model

Analogously to traditional (non-MC) recurrent tasks, an MC recurrent task $\tau_k$ is characterized by a five-tuple $(\chi_k, C_k^L, C_k^H, D_k, T_k)$, with the following interpretation. Task $\tau_k$ generates an unbounded sequence of jobs. Each such job has a deadline that is $D_k$ time units after its release. $T_k$ is the minimum inter-arrival time between two consecutive job arrivals. The criticality of each such job is $\chi_k$, and it has LO-criticality and HI-criticality WCET's of $C_k^L$ and $C_k^H$ respectively.

When a job from task $\tau_k$ arrives in the system, its actual execution time, say $\gamma$ is unknown. If the job's $\gamma \leq C_k^L$, then the job exhibits LO-*criticality behavior*; if $C_k^L < \gamma \leq C_k^H$, then the job exhibits HI-*criticality behavior*; otherwise, it exhibits erroneous behavior.

**Correctness criteria.** We define an MC scheduling algorithm to be *correct* if it is able to schedule any system such that

- If all jobs exhibit LO-criticality behavior, all jobs complete by their deadlines; and
- If any job exhibits HI-criticality behaviors, but no jobs exhibit erroneous behavior, all jobs of HI-criticality tasks complete by their deadlines.

**Types of Tasks** We say that a task system is *implicit-deadline* if the deadline of each job $D_k$ is equal to its minimum interarrival time $T_k$. If $D_k < T_k$ for all tasks, then we call the system *constrained deadline*; otherwise the system has arbitrary deadlines.

We say that a task system exhibits strictly *periodic* behavior if consecutive jobs of each task arrive exactly $T_k$ time apart. If some jobs can arrive later (more than $T_k$ time after the previous job of the same task arrived), then we say that the task system exhibits sporadic behavior.

[1] We are assuming preemptive scheduling in this paper.

## III. Context: Periodic vs Sporadic Tasks

For traditional (non mixed-criticality) uniprocessor systems, we know that the worst case for schedulability of tasks is with strictly periodic arrivals. In particular, if a task system is schedulable when jobs of task $\tau_k$ arrive exactly $T_k$ time apart, then the task system is also schedulable if some inter-arrival times are larger [5] for both implicit deadlines and constrained deadlines. Therefore, the schedulability test for both instances are identical.

However, for multiprocessor systems, this is not true. For implicit deadline tasks, it is still true that all task systems with utilization at most $m$, where $m$ is the number of machines, are feasible; therefore, if a particular task system is schedulable with strictly periodic arrivals, then it is also schedulable with sporadic arrivals. On the other hand, for constrained deadline task systems, this is no longer true; there exist task systems which are schedulable with strictly periodic arrivals which are not schedulable with sporadic arrivals [6].

## IV. Open Problems

In this abstract, we are concerned with understanding whether mixed-criticality systems are similar to traditional uniprocessor systems with respect to periodic vs sporadic arrivals.

**Open Problem 1: Does periodic feasibility imply sporadic feasibility for implicit deadline tasks?**

Consider a dual-criticality task system and say that this task system is feasible with strictly periodic arrivals of jobs. Can we also say that this system is feasible with sporadic arrivals? It would be interesting to see if the hard instance for mixed-criticality tasks is still periodic arrivals like it is for both uniprocessor and multiprocessor traditional asks. In addition, we know that checking feasibility is NP-Hard for both periodic and sporadic tasks. However, in some cases, it may be possible to spend additional time to check feasibility for periodic tasks, but explicitly checking feasibility of sporadic instances may remain more difficult. If periodic feasibility implies sporadic feasibility, then this might help with some applications.

If the answer to this question is yes, it leads us to the second question:

**Open Problem 2: Does periodic feasibility imply sporadic feasibility for constrained deadline tasks?**

The problem is identical to the previous problem, just for constrained deadlines. The answer might be different. For instance, as mentioned above, for multiprocessor systems, periodic feasibility implies sporadic feasibility for implicit deadlines but not for constrained deadlines. It would be interesting to know mixed-criticality tasks also exhibit similar patterns.

**Open Problem 3: Are there classes of schedulers where periodic schedulability implies sporadic schedulability?**

Given a mixed-criticality task system $\tau$ and a scheduler $A$, say we have verified that $A$ can schedule $\tau$ correctly assuming strictly periodic arrivals. Can we then say that $A$ can also schedule $\tau$ if the arrivals are sporadic? The answer is obviously no for arbitrary schedulers — for instance, a scheduler could just choose to behave badly as soon as it sees a sporadic arrival. However, one can imagine that reasonable schedulers exist for which this claim is true. Could we characterize and understand what properties a scheduler must have for this to be true?

Finally, if the answer is yes for any of the above questions, then the questions can be generalized to mixed-criticality systems with more than two criticality levels.

## References


[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *Proceedings of the Real-Time Systems Symposium*. Tucson, AZ: IEEE Computer Society Press, December 2007, pp. 239–243.

[2] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, pp. 82:1–82:37, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3131347

[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," *IEEE Transactions on Computers*, vol. 61, no. 8, pp. 1140–1152, 2012.

[4] K. Agrawal and S. Baruah, "Intractability issues in mixed-criticality scheduling," in *Proceedings of Euromicro Conference on Real Time Systems (ECRTS)*, 2018.

[5] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proceedings of the 11th Real-Time Systems Symposium*. Orlando, Florida: IEEE Computer Society Press, 1990, pp. 182–190.

[6] S. K. Baruah and K. Pruhs, "Open problems in real-time scheduling," *Journal of Scheduling*, vol. 13, no. 6, pp. 577–582, 2010.

# Nested Locks in the Lock Implementation: The Real-Time Read-Write Semaphores on Linux

Daniel B. de Oliveira[1,2,3], Daniel Casini[3], Rômulo S. de Oliveira[2], Tommaso Cucinotta[3], Alessandro Biondi[3], and Giorgio Buttazzo[3]

[1]RHEL Platform/Real-time Team, Red Hat, Inc., Pisa, Italy.
[2]Department of Systems Automation, UFSC, Florianópolis, Brazil.
[3]RETIS Lab, Scuola Superiore Sant'Anna, Pisa, Italy.
Email: bristot@redhat.com, romulo.deoliveira@ufsc.br,
{daniel.casini,tommaso.cucinotta, alessandro.biondi, giorgio.buttazzo}@santannapisa.it

## I. Introduction

Linux is a general purpose operating system (GPOS) that has gained many real-time (RT) features over the last decade. For instance, nowadays Linux has a *fully preemptive* mode and a deadline-oriented scheduler [1]. Although some of these features are part of the official Linux kernel, many of them are still part of an external patch set, the PREEMPT-RT [2]. The PREEMP-RT changes the locking methods of Linux to prevent unbounded priority inversion. This is achieved by using the Priority Inheritance Protocol [3] on in-kernel mutexes, which bounds the activation delay in high priority tasks. Indeed, the *latency* is the main evaluation metric for the PREEMPT-RT Linux: for example, the Red Hat Enterprise Linux for Real-time [4] (based on PREEMPT-RT) shows a maximum latency of 150 $\mu$s on certified hardware. However, due to Linux's GPOS nature, RT Linux developers are challenged to provide the predictability required for an RTOS, while not causing regressions on the general purpose benchmarks. As a consequence, the implementation of some well known algorithms, like read/write semaphores, has been done using approaches that were not well explored in academic papers.

## II. Read-write semaphores on Linux

On Linux, the read-write semaphores provide concurrent readers and exclusive writers for a given critical section. For example, since the memory mapping information of a process is read very often but rarely changes during its execution, it is protected by a read-write semaphore.

The API of the read-write semaphores is composed by four main functions. Readers call `DOWN_READ()` before entering in the read-side, calling `UP_READ()` when leaving the read-side of the critical section. Writers should call `DOWN_WRITE()` before entering in the write-side of the critical section, calling `UP_WRITE()` when leaving. These functions take only one argument, which is a pointer to a structure `rw_semahore`. The `rw_semaphore` structure is presented in Figure 1[1].

The `readers` variable is an atomic type that counts how many concurrent readers are inside the critical section. This variable is also used to store `READER BIAS` and `WRITER BIAS` flags, which are used to define if there are either readers or a writer in the critical section. Whenever a task should block in the semaphore, it will do by blocking in the real-time mutex `rt_mutex` of the semaphore. The rt_mutex is defined as shown in Figure 2[1]:

```
1 struct rw_semaphore {
2   atomic_t                readers;
3   struct rt_mutex         rtmutex;
4 };
```

Fig. 1: Read-write Semaphore structure

```
1 struct rt_mutex {
2   raw_spinlock_t          wait_lock;
3   struct rb_root_cached   waiters;
4   struct task_struct      *owner;
5   int                     save_state;
6 };
```

Fig. 2: Real-time Mutex structure

In order to protect the fields of the `rt_mutex` struct from concurrent accesses, the spin lock `wait_lock` is used whenever the internal fields of the mutex are modified. The `wait_lock` of the real-time mutex is also used to avoid two writers setting the `WRITE/READ BIAS` concurrently in the `rw_semaphore` structure.

The pseudo-code of each operation is presented in Figure 3 and 4, respectively.

## III. Open Problems

Considering our example, when `DOWN_WRITE()` is called, the task that is trying to acquire the read/write semaphore for writing has to lock two nested resources, a regular *mutex* (acquired at line 8, Figure 4) and a *spin lock* (acquired at line 14, Figure 4), thus creating a **heterogeneous nested lock** (e.g., a suspension-based lock with a nested spin-based lock or vice-versa). This case study, taken from the Linux kernel, highlights two open issues. The first one concerns the need for implementing

[1]Debug fields removed from structure's definition.

```
1: function UP_READ(rw_sem) /* using atomic operations */
2:     if −−rw_sem->readers == 0 then
3:         if a writer is holding the rw_sem->rtmutex then
4:             wake-up the writer
5:         end if
6:     end if
7: end function
8:
9: function DOWN_READ(rw_sem)
10:    if ++rw_sem->readers > 1 then /* using atomic operations */
11:        return /* enter the critical section */
12:    else
13:        rw_sem->readers−−
14:    end if
15:    take rw_sem->rtmutex.wait_lock /* might block busy (spinlock) */
16:    if WRITER BIAS is not set then
17:        rw_sem->readers++
18:        release rw_sem->rtmutex.wait_lock
19:        return /* enter in the critical section */
20:    end if
21:    release rw_sem->rtmutex.wait_lock
22:    take rw_sem->rt_mutex /* might block suspended (rt mutex) */
23:    rw_sem->readers++
24:    release the rw_sem->rt_mutex
25:    return /* enter in the critical section */
26: end function
```

Fig. 3: Read-side operations

```
1: function UP_WRITE(rw_sem)
2:     clear WRITER BIAS
3:     set READER BIAS
4:     release sem->rtmutex
5: end function
6:
7: function DOWN_WRITE(rw_sem)
8:     take rw_sem->rtmutex /* might block suspended (rt mutex) */
9:     clear READER BIAS
10:    if rw_sem->readers != 0 then
11:        suspend waiting for the last reader
12:    end if
13:    while 1 do
14:        take sem->rtmutex->wait_lock /* might block busy (spinlock) */
15:        if sem->readers == 0 then
16:            set WRITER BIAS
17:            release rw_sem->rtmutex->wait_lock
18:            return /* enter in the critical section */
19:        end if
20:        release rw_sem->rtmutex->wait_lock.
21:        suspend waiting for the last reader
22:    end while
23:    return
24: end function
```

Fig. 4: Write-side operations

in Linux state-of-the-art protocols for (possibly heterogeneous) nested locks and developing novel analysis techniques. To the best of our knowledge, only few works on shared-memory multiprocessor synchronization targeted nested critical sections. Two notable examples are the work by Biondi et al. [5], in which a graph abstraction is introduced to derive a fine-grained analysis (i.e., not based on asymptotic bounds) for FIFO *non-preemptive* spin locks, and the one by Ward and Anderson [6], in which the *real-time nested locking protocol* (RNLP) is proposed, with the related *asymptotic* analysis. Later, Nemitz et al. [7] proposed an optimization for the average-case of RNLP. However, to the best of our knowledge, only the extension of RNLP proposed in [8] is explicitly conceived to deal with heterogeneous nested critical sections. The protocol is presented with the related asymptotic analysis, and an experimental study aimed at assessing schedulability. Future research work could target the issues in implementing the extended RNLP [8] in Linux. Also, it is worth considering the possibility of extending the graph abstraction proposed by Biondi et al. [5] to allow fine-grained analysis for nested heterogeneous locks.

The second open problem concerns the design of specialized analysis techniques accounting for specific implementations of complex types of locks (e.g., the aforementioned read/write lock in Linux). Considering the problem previously presented for the `DOWN_WRITE` function, an implementation-aware analysis would account for the contention on the heterogeneous nested critical section, considering it when a blocking-bound for the reader/writer semaphore is derived. The analyses for reader/writer semaphores that have already been proposed (e.g., the protocol proposed by Brandenburg and Anderson [9], or R/W RNLP [10], a variant of RNLP conceived to deal with nested, spin-based, read/write locks) could be integrated with implementation-specific aspects. The availability of blocking-bounds conceived considering the specific implementation adopted in the Linux kernel may help it to be more suitable for real-time contexts. Finally, a third open research area consists in finding more efficient locking protocols (with the related implementation), accounting for both general purpose benchmark performance (i.e., average-case behavior, needed by the GPOS nature of Linux) and predictability.

## REFERENCES


[1] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016.

[2] D. B. de Oliveira and R. S. de Oliveira, "Timing analysis of the PREEMPT RT linux kernel," *Softw., Pract. Exper.*, vol. 46, no. 6, pp. 789–819, 2016.

[3] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, Sep. 1990.

[4] Red Hat, Inc., "Red Hat Enterprise Linux for Real Time," Available at: https://www.redhat.com/it/resources/red-hat-enterprise-linux-real-time [last accessed 28 March 2017].

[5] A. Biondi, A. Weider, and B. Brandenburg, "A blocking bound for nested fifo spin locks," in *Real-Time Systems Symposium (RTSS)*, 2016, pp. 291–302.

[6] B. C. Ward and J. H. Anderson, "Supporting nested locking in multiprocessor real-time systems," in *Real-Time Systems (ECRTS), 2012 24th Euromicro Conference on*, 2012, pp. 223–232.

[7] C. E. Nemitz, T. Amert, and J. H. Anderson, "Real-time multiprocessor locks with nesting: Optimizing the common case," in *Proceedings of the 25th International Conference on Real-Time and Network Systems (RTNS 2017)*, 2017.

[8] B. C. Ward and J. H. Anderson, "Fine-grained multiprocessor real-time locking with improved blocking," in *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, ser. RTNS '13, 2013.

[9] B. B. Brandenburg and J. H. Anderson, "Reader-writer synchronization for shared-memory multiprocessor real-time systems," in *2009 21st Euromicro Conference on Real-Time Systems*, July 2009, pp. 184–193.

[10] B. C. Ward and J. H. Anderson, "Multi-resource real-time reader/writer locks for multiprocessors," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014, pp. 177–186.

# Towards temporal constraints in self driving cars

E. Ntaryamira, C. Maxim, C. Flores and L. Cucu-Grosjean
INRIA Paris, France, firstname.lastname@inria.fr

## I. Introduction and Context

The increased number of functionalities requires an evolution of the technologies embedded within critical systems that, as a result, become more complex every day. A clear application of these systems belongs to the transportation industry, where designing safe and reliable control of automated vehicles strings. Thereby, accident numbers and their severity is expected to be reduced as well as the saved energy together with reduced harmful exhaust emission [1]. Such systems are composed by an important number of interacting sub-systems which make the associated feasibility problem often intractable.

In this paper we focus on the cooperative adaptive cruise control (CACC) which is an extension of the adaptive cruise control (ACC) [2]. The main advantage of the CACC over ACC is the incorporation of a V2V communication layer that enables faster vehicle responses and shorter inter-distances within the platoon. The term platoon refers to an automated vehicles formation where all members regulate their spacing gap towards their preceding vehicles, behaving as a whole interconnected system.

According to the principle of the platoon, vehicles (the leader, the first or the following vehicle) are equipped with longitudinal automation, ranging sensors and can also profit from the mentioned V2V links.

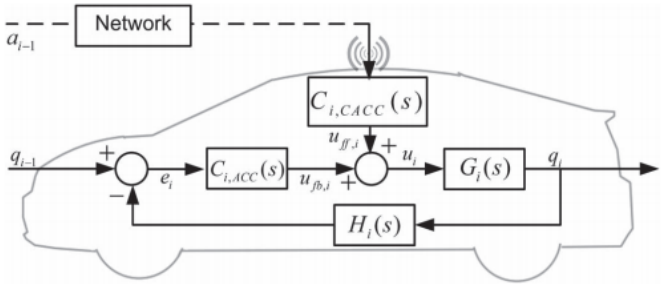



Fig. 1: Control structure block diagram of a single CACC-equipped vehicle [3]

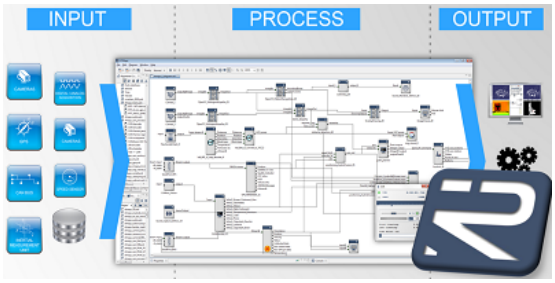



Fig. 2: RTMaps modeling tool [4]

Each car has a two layers automation: the high level is in charge of regulating the spacing during acceleration, and the low level manages the vehicle actuators for tracking the reference acceleration. To achieve the functional behavior, the high level require variables from ego-vehicle and preceding(s) while the low level one requires only information from the ego-vehicle.

Provided with this information, the car adjusts some physical parameters (the steer bar torque, break pedal torque, acceleration power,etc) to reach the reference values timely, setting the system subject to precedence constraints. Graphical block diagramming tools such as simulink or RTMaps are used (see Figure 2).

To provide the platoon string stability, existing results set the operating limits in terms of network latency, considering heterogeneity of technologies and their problems (CAN, V2V, etc).

## II. System Model

In literature, CAN network is well presented and several papers as [5] outlines how the worst-case response times for CAN messages are computed. Though, existing results do not explain formally how compute the end-to-end timing parameters for a system which, in addition to CAN network, takes into account the communication delays due to vehicle-to-vehicle communication and radar/lidar. From the above, this set the system to be unpredictable and existing solutions more pessimistic.

Over the last decades, standards like *AUTOSAR* [6] have been proposed in order to conceive a common platform for the development of automotive software. These standards allow designers to build applications with different

requirements, including timing ones as beside the correctness of the computation result, the result is expected to be obtained within a time interval. In other to evaluate the overall system, time requirements containment on the end-to-end functionality of the task chains [7] are mandatory.

Despite the above condition, however do not provide the information how to compute end-to-end timing parameters for the system of interest in our paper. The use of multicore processors would provide the system with extended possibilities to deal with the complexity of ACC and CACC technologies. However, this is currently a challenge since there is no formal model to describe the interactions between components taking into account functional and non-functional(timing) behavior.

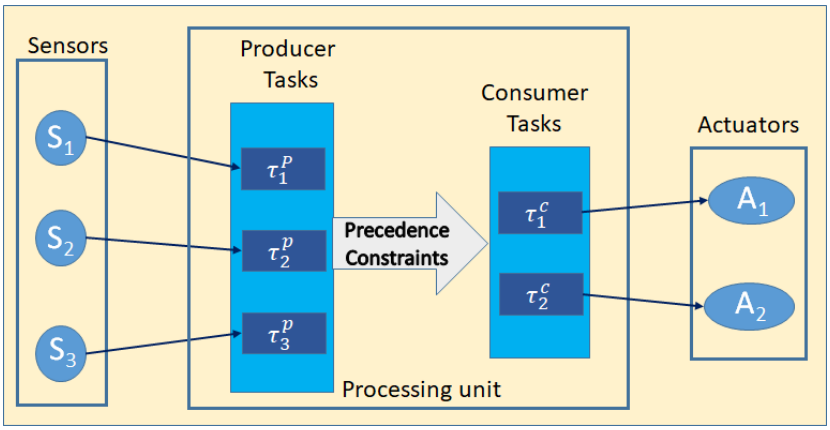



Fig. 3: Automated car model

We consider a vehicle system $S$ composed of a set of $n$ tasks $\tau_i = (C_i, T_i, D_i)$, where $i = \{1, 2, \cdots, n\}$. Every task $\tau_i$ is characterized by a worst-case execution time $C$, a minimal inter-arrival time $T$ and a deadline $D$. Tasks can be split in two different classes: producer ($\tau^p$) and consumer ($\tau^c$). The producer tasks are receiving information from exterior (sensors, wifi, CAN) with different frequencies and can be treated by a modeling tool like RTMaps, while the consumer tasks are executed by the ECU and will transmit commands to the actuators (brake, throttle, gearbox, steering, etc.) with certain frequency. Since the execution of the consumer tasks require data sampled by producers, $S$ is subjected to precedence constraints. An example of the presented model can be seen in Figure 3. We consider in this work global scheduling on the multicore processors. We understand by global scheduling that instances of different programs may be scheduled in different cores of the processor.

For the considered model, we list the associated **open problems** that are scheduling-oriented:

- *How is the execution time of each task estimated/computed?*
- *How are the tasks with precedence constraints scheduled on a multicore processor such that the system behaves correctly functionally and also satisfying the time constraints?*
- *How we provide a worst-case bound on the response time of the system?* We understand by response time of the system the time elapsed between the time instant when the system receives an information until the time instant when an associated action is finished.

## REFERENCES


[1] S. Z. Gaspar Peter and A. Szilard, "Highly automated vehicle systems," in *BME MOGI*, 2014. [Online]. Available: http://www.mogi.bme.hu/TAMOP/jarmurendszerek$_i ranyitasa_a ngol/index.html$

[2] wiki, "Adaptive cruise control," *Wikipedia*. [Online]. Available: https://en.wikipedia.org/wiki/Autonomous$_c ruise_c ontrol_s ystem$

[3] N. v. d. W. Sinan Oncu, Jeroen Ploeg and H. Nijmeijer, "Cooperative adaptive cruise control: Network-aware analysis of string stability," *IEEE Transactions on Intelligent Transportation*, 2014.

[4] N. du Lac, "Intempora, de nombreux projets de rd au cours des 15 dernires annes," 2014. [Online]. Available: https://pole-moveo.org/success-story/intempora-de-nombreux-projets-de-rd-au-cours-des-15-dernieres-annees/

[5] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, no. 3, pp. 239–272, Apr. 2007.

[6] AUTOSAR, "Last access october and specifications of timing extensions," *AUTOSAR*, 2016. [Online]. Available: www.autosar.org

[7] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture - Embedded Systems Design*, 2017.

# Deep Neural Networks for Safety-Critical Applications: Vision and Open Problems

Daniel Casini, Alessandro Biondi, Giorgio Buttazzo
Scuola Superiore Sant'Anna
Pisa, Italy
Email: {daniel.casini, alessandro.biondi, giorgio.buttazzo}@santannapisa.it

## I. Introduction

Autonomously-driving cars may become part of our everyday lives in a few years. One of the key factors that will likely enable the development of autonomous systems is artificial intelligence and, in particular, deep neural networks (DNNs). According to the results of a popular challenge for assessing the performance of DNNs [1], the error rate in image classification reduced from 28% in 2010 to 2.3% in 2017 [2], surpassing the human capability, assessed between 5% and 10% (see Figure 1). Neural networks are computing system inspired by the structure of mammalian brains, composed of many neurons (represented by nodes) exchanging signals through synaptic channels (represented by weighted edges). Synaptic weights are tuned during a supervised learning procedure, in which the network is trained by examples to match a set of expected outputs when specific inputs are presented. The adoption of neural networks in safety-critical scenarios presents many issues. For instance, the difficulty of understanding the meaning of weights configurations and the difficulty of predicting the network output to similar input patterns represents a crucial problem for certification: who would certificate something whose behavior is not completely understandable at design time and cannot be predictably replicated? A possible solution may consist in adopting algorithms based on *hard computing* (i.e., with a well-defined behavior) aimed at acting as *supervisors* for DNNs. The supervisor would detect misbehaviors from the neural network, redirecting the actuation to some defined safe actions. A further improvement (but still in terms of average-case behavior) can be achieved using redundant neural networks in conjunction for performing the same task, and merging the result by means of a voting mechanism.

The complexity of a neural network also makes it prone to attacks and malfunctionings, representing a security threat: what if an attacker exploits the weakness of a DNN to take control over the related, safety-critical, steering system? Morever, DNNs are typically executed on top of general purpose operating systems, whereas actuations take place in a Real-Time Operating System (RTOS). For these reasons, guaranteeing isolation between DNNs and safety-critical tasks represents an important issue.

A popular and effective technique to achieve isolation using a single heterogeneous multi-processor platform is to use a *hypervisor* to separate a computationally intensive processing (i.e., DNNs) and safety-critical activities (e.g., AUTOSAR compliant) in different domains. In this way, the two domains can safely co-exist, also adopting different operating systems. This solution is shown in Figure 2.

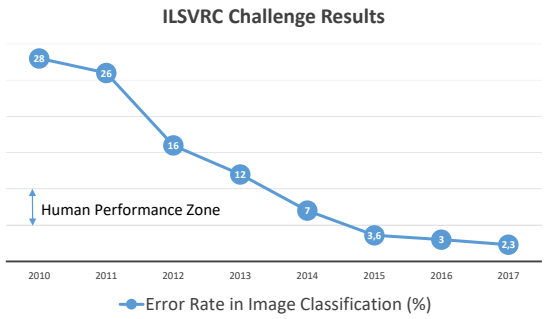



Fig. 1. Evolution of DNNs over the years in terms of error rate in image classification tasks.

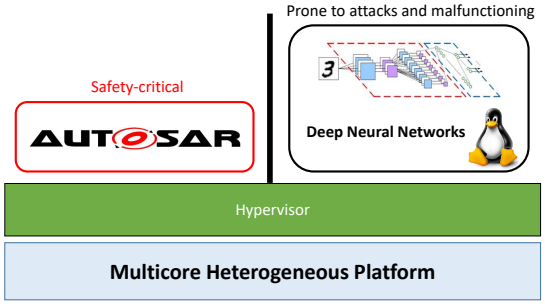



Fig. 2. Sample architecture for achieving isolation between DNNs and safety critical tasks when the underlying hardware platform is shared.

Another key issue is represented by the temporal predictability of DNNs. However, predictability may be a requirement only when DNNs are used in their *inference phase* (i.e., the network is already trained and it is only queried). Although a DNN workload can be represented with a direct acyclic graph[1] (DAG), finding a representative task model (e.g., that accounts

[1]In its classical form: for instance the Tensorflow programming model provides some extensions for branching and looping, which leads to different task models [3].

for memory accesses due to tensors exchanged between directly connected nodes) is fundamental to correctly characterize its temporal behavior. Deep Neural Networks execute on hardware platforms composed of heterogeneous processing units, such as (possibly asymmetric) multi-core CPUs, GPUs and portions of reconfigurable FPGAs. Recently, ad-hoc application specific integrated circuits (ASICs) have been also produced and may be commonly included in commercial heterogeneous platforms soon. The tensor processing unit (TPU) [4] produced by Google and compatible with Tensorflow, one of the most used frameworks for developing Deep Neural Networks, is a notable and promising example. Accounting for the underlying hardware architecture is unavoidable when designing more tailored analysis techniques, but not always simple. This is the case of NVIDIA GPGPUs, whose internal structure and mechanisms are not publicly documented. For these platforms, research can only rely on black-box characterizations based on experimentation results [5]. The support of FPGAs is not yet available for most of the DNNs frameworks (e.g., Tensorflow), but efforts have been done to make them supported [6]. An interesting area of research may also evaluate the execution of DNNs with dynamic partial reconfiguration [7], which recently have been made available (as a prototype) under Linux [8]. Using this approach, computationally intensive nodes may be accelerated by executing them on a FPGAs, exploiting dynamic partial reconfiguration to avoid having a fixed binding between a portion of FPGA area and one or more DNN operations. The support of heterogeneous platforms opens to research on partitioning neural network nodes to different types of devices. For instance, the placement algorithm (the one aimed at deciding where each node should be executed, e.g., in a GPU or in a CPU) still needs to be improved in TensorFlow [3]. Among the usual frameworks for developing and executing DNNs (e.g., Caffe [9], Tensorflow [10], Torch [11], etc.), novel frameworks specialized for inferencing DNNs have been recently developed. For instance, it is worth mentioning TensorFlow Serving [12], an inference framework aimed at increasing the throughput of a DNN by grouping individual inference requests into batches for joint execution on a GPU, and NVIDIA TensorRT [13], which implements a pre-processing phase aimed at optimizing the neural network for a future inference in an NVIDIA GPU. In this way, latency can be decreased and throughput increased. Predictability of optimized inference engines for DNNs should also be evaluated as a future research work.

## II. Open Problems: Categorization and Key Questions

### A. Certification

1) How to support DNNs in a safety-critical context to build a system that is prone for being certified?
2) Which type of *supervision algorithms* are needed to detect wrong outputs and redirect actuation to safe actions?

### B. Isolation

1) How to avoid that the complexity of DNNs may lead to security threats for a safety-critical system running on top of a shared hardware platform?
2) Which mechanism have to be provided to allow them interacting while running in different operating systems?

### C. Predictability in the execution

1) Which is a suitable task model to describe and analyze the temporal behavior of a DNN?
2) How to account for novel (highly heterogeneous) computing platforms?
3) How to account for inference engines that affect the DNN's execution?

## References


[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[2] [Online]. Available: http://image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf

[3] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vigas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2015. [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf

[4] [Online]. Available: https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

[5] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the NVIDIA TX2: Hidden details revealed," in *Proceedings of the 38th IEEE Real-Time Systems Symposium (RTSS 2017)*, Paris, France, December 5-8 2017.

[6] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 152–159.

[7] A. Biondi, A. Balsini, M. Pagani, E. Rossi, M. Marinoni, and G. Buttazzo, "A framework for supporting real-time applications on dynamic reconfigurable FPGAs," in *Real-Time Systems Symposium (RTSS), 2016 IEEE*, Porto, Portugal, November 29 - December 2 2016.

[8] M. Pagani, A. Balsini, A. Biondi, M. Marinoni, and G. Buttazzo, "A linux-based support for developing real-time applications on heterogeneous platforms with dynamic fpga reconfiguration," in *System-on-Chip Conference (SOCC), 2017 30th IEEE International*. IEEE, 2017, pp. 96–101.

[9] [Online]. Available: http://torch.ch/

[10] [Online]. Available: https://www.tensorflow.org/

[11] [Online]. Available: http://caffe.berkeleyvision.org/

[12] [Online]. Available: https://github.com/tensorflow/serving

[13] [Online]. Available: https://developer.nvidia.com/tensorrt