WMC 2022

Proceedings of the 9th International Workshop on Mixed Criticality Systems

jointly hold with RTSS 2022, Houston (Hybrid)

Edited by

Xiaotian Dai and Zheng Dong

December 2022

Organizers

Program Co-chairs

Zheng Dong, Wayne State University
Xiaotian Dai, University of York

Steering Committee

Iain Bate, University of York
Arvind Easwaran, Nanyang Technological University
Zhishan Guo, North Carolina State University
Jing Li, New Jersey Institute of Technology

Program Committee Member

Nan Guan, City University of Hong Kong
Geoffrey Nelissen, Tu Eindhoven
Daniel Casini, Scuola Superiore Sant'anna - Pisa
Lea Schönberger (Publicity Chair), Tu Dortmund University
Kecheng Yang, Texas State University
Konstantinos Bletsas, Cister
Mohamed Hassan, Mcmaster University
Renato Mancuso, Boston University
Yasmina Abdeddaim, Université Gustave Eiffel, Esiee Paris
Bryan C. Ward, Vanderbilt University
Jinghao Sun, Dalian University of Technology
Jinkyu Lee, Sungkyunkwan University
Nathan Fisher, Wayne State University
Shuai Zhao, Sun Yat-sen University
Corey Tessler, University of Nevada, Las Vegas
Zhe Jiang, University of Cambridge
Georg von der Brüggen, Tu Dortmund, Germany

Message from the Program Chairs

It is our pleasure to welcome you to the 9th International Workshop on Mixed Criticality Systems (WMC) at the Real-Time Systems Symposium (RTSS) in Houston, USA on 5th Dec 2022.

The purpose of WMC is to share new ideas, experiences, and information about research and development of mixed-criticality real-time systems. The workshop aims to bring together researchers working in fields relating to real-time systems with a focus on the challenges brought about by the integration of mixed-criticality applications onto single-core, multi-core, and many-core architectures. These challenges are cross-cutting. To advance rapidly, closer interaction is needed between the sub-communities involved in real-time operating systems / run-time environments/hypervisor, real-time scheduling, security, safety, and timing analysis. The workshop aims to promote an understanding of the fundamental problems that affect Mixed Criticality Systems (MCS) at all levels in the software/hardware stack and crucially the interfaces between them. The workshop will promote lively interaction, cross-fertilization of ideas, synergies, and closer collaboration across the breadth of the real-time community, as well as attract industrialists from the aerospace, automotive, and other industries with a specific interest in MCS.

For this ninth edition of the workshop, 8 submissions were received. The review process involved 17 Program Committee members, with each submission receiving at least 4 reviews. We decided to accept all submissions (one paper was accepted with shepherding.) for presentation at the workshop, including 5 regular unpublished papers and 3 Journal-Never-Presented papers. We sincerely thank all the Program Committee members for their time and effort in the review process. As well as regular paper presentations, there is a keynote session on "Probabilistic Real-Time Scheduling and its Possible Link to Mixed-Criticality Systems", given by Dr. Jian-jia Chen (TU Dortmund, Germany).

WMC 2022 would not be possible without the hard work of people involved in organizing RTSS 2022, including Drs. Liliana Cucu-Grosjean, Arvind Easwaran, Sebastian Altmeyer and Christopher Gill. In particular, we would like to thank the RTSS 2022 Hot-Topics Day chair Dr. Dionisio de Niz for his excellent organization and great support of the overall workshops program. We also thank the WMC Steering Committee for their guidance and suggestions during the preparation of WMC 2022.

Finally, we would like to thank all of the authors who submitted their work to WMC 2022 and the keynote speaker. We wish you an interesting and exciting workshop and an enjoyable stay in Houston.

Zheng Dong (Wayne State University, USA),
Xiaotian Dai (University of York, UK),
WMC 2022 Program Co-Chairs

Technical Program

Probabilistic Real-Time Scheduling and its Possible Link to Mixed-Criticality Systems

Georg von der Brüggen*, Sergey Bozhko†, Mario Günzel*,
Kuan-Hsun Chen§, Jian-Jia Chen*, and Björn B. Brandenburg†

*TU Dortmund University, Germany
†Max Planck Institute for Software Systems (MPI-SWS), Germany
§University of Twente, The Netherlands

***Abstract*—Proving hard real-time guarantees based on a classical analysis may significantly underutilize the processor in the average case. Therefore, instead of considering a very rare worst-case scenario, a probabilistic scheduling analysis determines the probability of a deadline miss. Such an analysis assumes that task execution times are given by a set of modes representing the range of possible execution scenarios. Considering tasks with multiple modes and different levels of assurances, in this case expressed as different probabilities to miss deadlines, for different tasks provides a natural link to mixed-criticality systems.**

This work summarizes recent results in probabilistic real-time scheduling and some potential problems that should be considered when linking these results to mixed-criticality systems. In addition, possible connections between mixed-criticality systems and probabilistic analysis are detailed. The goal of this work is to start a discussion to determine whether such probabilistic results may be interesting for mixed-criticality research.

I. Introduction

A classical, deterministic scheduling analysis for hard real-time systems examines the question whether, given a set of (recurrent) tasks, all task instances meet their deadline in all circumstances. These analyses assume that jobs are always executed according to their WCET. Proving timing guarantees under these pessimistic assumptions may significantly underutilize the processor in the average case.

Considering this dilemma, the mixed-criticality approach proposed by Vestal [13] in 2007 has started an active research field within the real-time systems community (the latest version of the survey by Burns and Davis [3] lists 660 related papers). In a mixed-criticality system, tasks have multiple execution modes with different related execution-time budgets. For instance, a dual-criticality system can be in high-criticality or low-criticality mode and is comprised of two kinds of tasks, high-criticality and low-criticality tasks. At system start, the system is in low-criticality mode and timing guarantees are provided for all tasks in the system. If one of the high-criticality tasks overshoots its execution time budget, the system switches to the high-criticality mode, where the execution time budget of high-criticality tasks is increased while no guarantees are provided for low-criticality tasks.

This mixed-criticality model, in its most basic form, has received considerable criticism [10], [9], [14] since low-criticality tasks are abandoned once the system switched to high-criticality mode and no return to low-criticality mode was considered. This resulted in research into more realistic mixed-criticality models and graceful degradation of the service of low-criticality tasks. Please see Section 6 in the survey by Burns and Davis [3] for details.

As a result, mixed-criticality research more frequently considered systems where tasks may switch their execution behavior frequently instead of assuming a single mode switch. In such a scenario, it seems natural to consider situations where only a small subset of tasks exhibits larger execution times for a limited time interval. Hence, performing a system mode switch may be both costly and unnecessary. Furthermore, in 2020, an empirical survey by Akesson et al. [1] revealed that 62% of the responding real-time practioners work on systems that include soft or firm real-time tasks and for 45% of the systems even the most critical functions can endure occasional deadline misses. Hence, an alternative approach to provide guarantees in mixed-criticality systems may be to determine how large the probability for a deadline miss actually is if intervals in which larger execution times occur are short or larger execution times are rare and to adjust runtime measures accordingly. For example, a system mode switch might only be performed if the probability that a deadline miss occurs in a critical function exceeds a certain threshold.

The risk of deadline misses can be quantified in a probabilistic schedulability analysis, usually considering either the deadline miss rate (that is, the percentage of deadline misses in the long run) or the worst-case deadline failure probability (WCDFP) (i.e., an upper bound on the probability of the first deadline miss in a busy window). A survey on probabilistic schedulability in real-time systems community has been provided by Davis and Cucu-Grosjean in 2019 [8]. We provide a summary on recent work in the area, point out open research questions, and possible links to mixed-criticality systems[1]. Our goal is to determine whether such probabilistic results are potentially interesting for the mixed-criticality research community.

[1]This submission is an extension of the one presented at the 15th Workshop on Models and Algorithms for Planning and Scheduling (MAPSP) 2022, which only focused on probabilistic scheduling but did not consider mixed-criticality systems. The version submitted to MAPSP can be found at https://mapsp2022.polito.it/Proceedings.pdf, page 143-145.

II. Probabilistic Analysis: Basics and Problems

We assume that a task's execution time is described as a set of possible modes, each defined by a pair of (i) its maximum execution time in that mode and (ii) the related probability, e.g., $\begin{smallmatrix} C_i \\ \mathbb{P}_i \end{smallmatrix} = \left(\begin{smallmatrix} 3 & 5 \\ 0.9 & 0.1 \end{smallmatrix}\right)$ means that τ_i has an execution time of at most 3 with probability 0.9 and an execution time 5 with probability 0.1.

Assuming a given release pattern, the probability that jobs miss their deadline under a given scheduling algorithm can be calculated via job-level convolution. Figure 1 shows an example of job-level convolution under static-priority scheduling.

The example considers 3 jobs, 2 of the higher-priority task τ_1 and 1 of task τ_2. The goal is to determine the probability that the job of τ_2 misses its deadline. We start in an initial state where the execution time is 0 with probability 1, that is, no job has yet been executed. Jobs are convolved one by one with the current states by summing up the ETs while multiplying the probabilities. This iteratively calculates the probability that the job of τ_2 meets its deadline at $t = 8$ or at $t = 14$, since all possible job-cost combinations are considered.

However, one of the main problems in probabilistic analysis can also be observed in Figure 1, namely, that the number of states can be exponential in the number of jobs for a job-level convolution. Therefore, it can only directly be applied if, on the one hand, the number of jobs that must be considered is small and, on the other hand, the number of release patterns that must be considered is small as well. Otherwise, the computational complexity is too high to be feasible in practice. As a result, two important research questions are:

1) How can the number of release patterns that have to be examined be reduced?
2) How can the deadline miss probability for one of these scenarios be determined efficiently?

Especially in the context of mixed-criticality systems, these calculations must also be applicable when the execution times of jobs are not independent due to a mode switch.

In the following, we give a brief overview on the progress on these fundamental research questions.

III. Efficient Approximation of Miss Probabilities

One approach to speed up the calculation using job-level convolution is reducing the number of states by re-sampling [12]. Specifically, states are combined to reduce the number of states as soon as the number of states exceeds a configurable threshold. However, re-sampling also reduces the precision of the calculation in a way that, in a non-trivial manner, depends on the concrete re-sampling scheme. Markovic et al. [11] introduced optimal re-sampling schemes that minimize the precision loss. However, bounding the loss remains an open problem. Markovic et al. [11] also detailed how cyclic convolution can be used instead of direct convolution to improve the calculation efficiency.

Instead of considering all possible job-cost combinations at the same time, the Monte-Carlo Response Time Analysis by Bozhko et al. [2] analyzes job traces individually. In each iteration, one specific trace (for instance, the one indicated with brown arrows in Figure 1) is sampled. Specifically, in each iteration, jobs are considered one by one, each time drawing one of the possible execution times according to the related probabilities. To estimate the deadline failure probability for the job under analysis, the number of observed deadline misses is counted and divided by the number of iterations, and then combined with an estimate of the confidence interval at a configurable level of assurance. The Monte-Carlo Response Time Analysis is scalable to scenarios with a very large number of jobs and is easily parallelizable. It allows to provide estimates with a known precision interval, but may require an infeasible number of samples when this interval must be too small.

Another approach is to not consider the jobs in order of arrival but to instead evaluate all relevant intervals individually. For instance, for the example in Figure 1, first the deadline failure probability for the interval [0,8] and then for the interval [0,14] would be calculated. The main idea of this approach is to make up for always starting from scratch by speeding up the calculation for each interval. The task-level convolution by von der Brüggen et al. [15] utilizes the fact that, when a specific interval is considered, the workload contributed by a specific task only depends on the number of jobs in a specific mode, but not on their specific order.

Analytic bounds estimate the probability for each interval individually as well. They, however, do not consider individual job modes to determine the workload the jobs contributes. Instead, the probability that the workload in a given interval is larger than the interval length is estimated directly, using analytic bounds. The most prominent approach is the line of work from Chen et al. [6], [4] utilizing Chernoff Bounds. While results exploiting Hoeffding's or Bernstein inequalities [15] are preferable regarding runtime, Chernoff Bounds usually provide a better tradeoff between runtime and precision. However, Chernoff Bounds do not provide any precision guarantees.

IV. Determining a Worst-Case Release Patterns

Similar to the idea of the classical critical instant, one approach to reduce the analysis complexity is to determine a certain scenario that always provides the worst case or an upper bound on the deadline miss probability.

When considering the worst-case deadline failure probability under static-priority scheduling, Maxim and Cucu-Grosjean [12] proposed such a scenario in 2013, and Chen and Chen [4] provided an alternate proof in 2017. Unfortunately, the depicted scenario, which is identical to the classical critical instant, has been contradicted with a counterexample by Chen et al. [5] in 2022. Chen et al. [5] also provided two over-approximations for the worst-case release pattern, which can be utilized to over-approximate the worst-case deadline failure probability. The question whether there is one specific release pattern that always results in a worst-case workload for any interval under static-priority scheduling remains open.

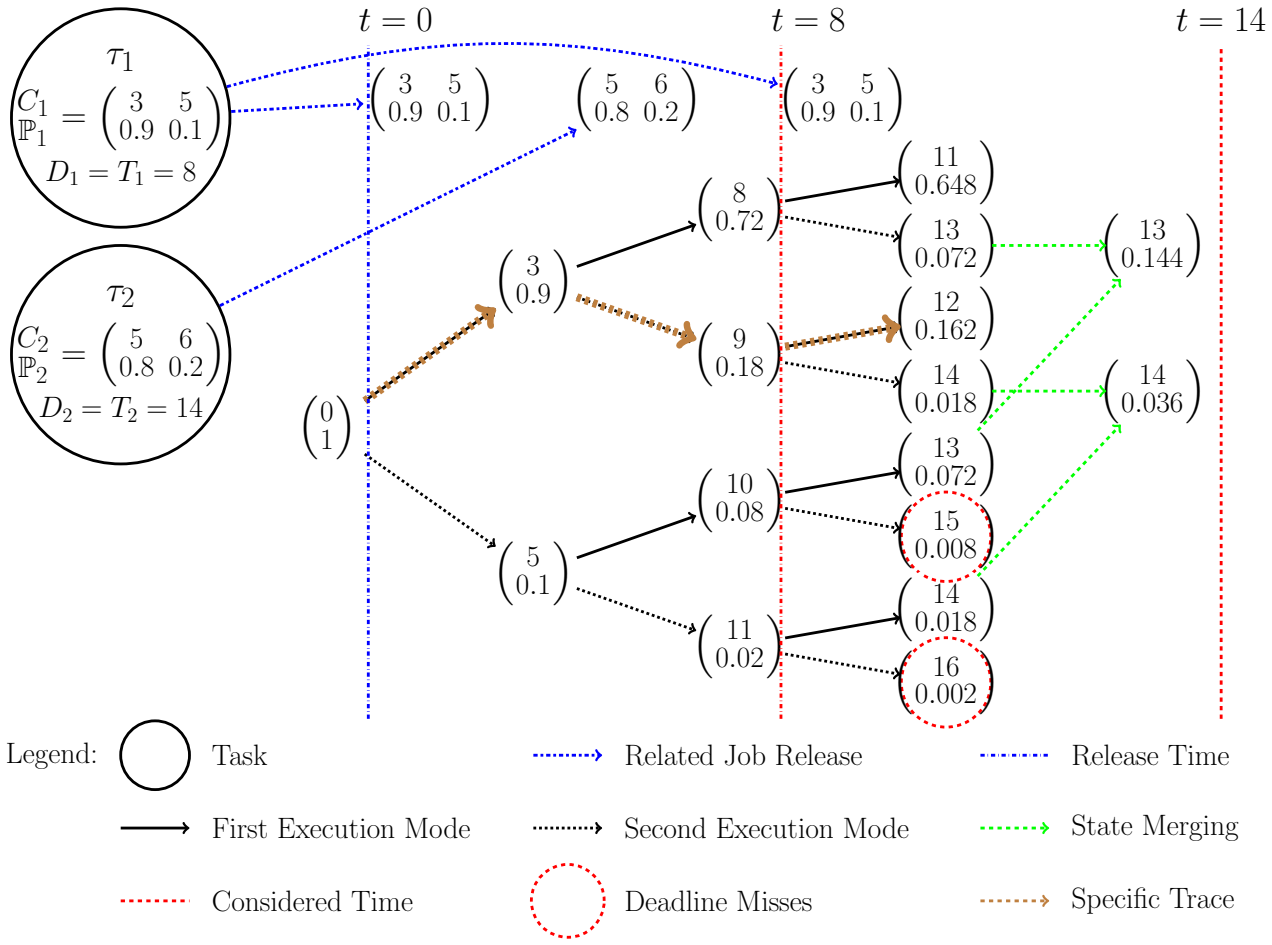


Fig. 1. A convolution example for two tasks under rate-monotonic scheduling.

Fortunately, the results discussed in the previous section are still applicable, as they provide efficient calculation methods for a given release pattern, but do not utilize any specific property of the critical instant.

For earliest-deadline first scheduling, von der Brüggen et al. [16] showed a worst-case scenario that upper-bounds the worst-case deadline failure probability in 2021.

No approach that can analytically bound the deadline miss rate is known under either static-priority scheduling or earliest-deadline first scheduling, as the result by Chen et al. [7] for static-priority scheduling is not applicable anymore due to the recently provided counterexample [5].

V. Job Dependencies

The previously discussed worst-case scenarios and calculation methods assume that the probabilities for job execution times are probabilistically independent. Therefore, applying them to mixed-criticality systems, where all, or at least some, tasks jointly switch into high-criticality mode is not straight forward. Nevertheless, von der Brüggen et al. [16] provided an over-approximation that, under earliest deadline first, enables dependencies in an restricted scenario. Specifically, they assumed that the dependencies can be modelled as acyclic task chains and that job modes depend on the modes of predecessors in those chains. This scenario is similar to mixed-criticality systems where some tasks triggers high-criticality behavior in all tasks in the system. It thus may be applicable to mixed-criticality systems. Alternatively, this approach may allow to analyze scenarios where a subset of the tasks in the system switch to high-criticality mode.

VI. Possible Links

In addition to the links already mentioned so far, there are a number of possible connections between mixed-criticality systems and probabilistic analysis of real-time systems. The notion of different levels of assurance for high-criticality and low-criticality tasks naturally can be interpreted as different thresholds for acceptable residual risk of deadline misses. Especially since high-criticality (or degraded-mode) behavior is expected to be rare at runtime, if lower-criticality tasks are seen as having a higher tolerance for occasional deadline misses, a probabilistic view would allow considerable resources to be reclaimed.

For example, it might be interesting to explore whether it is possible to extend the Monte Carlo approach by Bozhko et al. [2] to a probabilistic mixed-criticality setup. If it is

possible to identify a small number of relevant job-arrival sequences that must be considered, it may be possible to sample bounds on the deadline failure probability with a configurable, criticality-specific level of confidence. In particular, it may be possible to take mode changes into account simply by sampling (also) schedules in which mode changes occur, so that the final probability bounds reflect not only assurances for high-criticality tasks, but also how low-criticality tasks fare in the event of a mode change. In other words, a probabilistic approach could provide low-criticality tasks with much stronger guarantees than merely "best effort" in the event that a higher-criticality mode is entered.

Probabilistic analyses could also exploit another opportunity related to mode changes. A classic dual-criticality analysis must address (at least) three scenarios: the system in stable low-criticality mode, the system in stable high-criticality mode, and crucially, the system as it transitions from low- to high-criticality. The latter case, the time of mode transition is the most challenging aspect from a scheduling point of view, because increased high-criticality demand coincides with the lingering effects of pre-mode-change low-criticality interference, and hence typically represents the "assurance bottleneck." A probabilistic analysis could exploit that it is unlikely that low-criticality tasks exhibit maximum resource demand (and generate maximum interference) at precisely the moment when a high-criticality task triggers a mode change — at least if tasks of different criticalities are independent. A more refined analysis down the line could then further extend such an analysis to take into account possible dependencies between high- and low-criticality tasks.

In a different direction, it would also be interesting to inject the central notion of mixed-criticality systems into probabilistic modeling. Specifically, the idea that high- and low-criticality task parameters express different levels of assurance can also be seen as different levels of confidence in the correctness of specified mode probabilities. For example, when characterizing the chance that a job enters an "exceptional mode" associated with an increased execution cost (rather than its cheaper "normal mode"), it is reasonable to expect that a more risk-averse estimate would be obtained for a high-criticality task than for a low-criticality task. Consequently, it could be interesting to explore a different kind of what-if analysis: what happens to high-criticality tasks if the probability distribution assumed for low-criticality tasks turns out to be optimistic? This is akin to the classic mixed-criticality question — what happens to high-criticality tasks if low-assurance WCET estimates are optimistic — but comes with a twist that makes it considerably more difficult: whereas it is obvious when a low-assurance WCET estimate is exceeded, it is generally much harder to pinpoint when a low-assurance execution-time distribution is refuted by observations in practice. Thus, this line of exploration faces not only hard stochastic analysis problems, but also open question concerning the design of runtime mechanisms that would be appropriate for probabilistic mixed-criticality systems.

VII. Conclusion

Probabilistic timing analysis may be an interesting approach when considering mixed-criticality systems, as it may provide argumentation to postpone or omit a mode switch if the probability that a high-criticality task may miss a deadline is sufficiently small. Furthermore, even if mode switches become unavoidable, a probabilistic analysis may be able to recover much pessimism at a specified degree of residual risk.

However, the field of probabilistic scheduling itself still holds multiple open research questions, especially for establishing worst-case arrival patterns, when bounding the deadline-miss rate, and when considering probabilistically dependent jobs.

Therefore, extensions to mixed-criticality are not straight forward and will require further advances in the field of probabilistic scheduling. It thus seems interesting to start a discussion on how such extensions could benefit mixed-criticality research.

References

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis. An empirical survey-based study into industry practice in real-time systems. In *41st IEEE Real-Time Systems Symposium, RTSS*, 2020.

[2] S. Bozhko, G. von der Brüggen, and B. B. Brandenburg. Monte carlo response-time analysis. In *42nd IEEE Real-Time Systems Symposium, RTSS*, 2021.

[3] A. Burns and R. Davis. Mixed criticality systems-a review. Technical report, University of York, 2022. 13th edition.

[4] K.-H. Chen and J.-J. Chen. Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In *Symposium on Industrial Embedded Systems*, 2017.

[5] K.-H. Chen, M. Günzel, G. von der Brüggen, and J.-J. Chen. Critical instant for probabilistic timing guarantees: Refuted and revisited. In *Real-Time Systems Symposium*, 2022.

[6] K.-H. Chen, N. Ueter, G. von der Brüggen, and J.-J. Chen. Efficient computation of deadline-miss probability and potential pitfalls. In *Design, Automation & Test in Europe*, 2019.

[7] K.-H. Chen, G. von der Brüggen, and J.-J. Chen. Analysis of deadline miss rates for uniprocessor fixed-priority scheduling. In *24th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA*, 2018.

[8] R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic schedulability analysis techniques for real-time systems. *Leibniz Trans. Embed. Syst.*, 6(1):04:1–04:53, 2019.

[9] R. Ernst and M. D. Natale. Mixed criticality systems - A history of misconceptions? *IEEE Design & Test*, 33(5):65–74, 2016.

[10] A. Esper, G. Nelissen, V. Nélis, and E. Tovar. How realistic is the mixed-criticality real-time system model? In *RTNS*, 2015.

[11] F. Markovic, A. V. Papadopoulos, and T. Nolte. On the convolution efficiency for probabilistic analysis of real-time systems. In *Euromicro Conference on Real-Time Systems, ECRTS*, 2021.

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

[13] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[14] G. von der Brüggen, K.-H. Chen, W.-H. Huang, and J.-J. Chen. Systems with dynamic real-time guarantees in uncertain and faulty execution environments. In *37th IEEE Real-Time Systems Symposium, RTSS*, 2016.

[15] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik. Efficiently approximating the probability of deadline misses in real-time systems. In *Euromicro Conference on Real-Time Systems*, 2018.

[16] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, K. Morik, and B. B. Brandenburg. Efficiently approximating the worst-case deadline failure probability under EDF. In *42nd IEEE Real-Time Systems Symposium, RTSS*, 2021.

Mixed-Criticality Scheduling for Parallel Real-Time Tasks with Resource Reclamation

Qingqiang He[1], Nan Guan[2], Xu Jiang[3]

[1]The Hong Kong Polytechnic University, Hong Kong SAR
[2]City University of Hong Kong, Hong Kong SAR
[3]Northeastern University, China

***Abstract*—This paper considers the mixed-criticality scheduling of parallel real-time tasks. With the purpose of guaranteeing the deadline for hard real-time tasks and reclaiming computing resources for soft real-time tasks, we propose an approach by online monitoring the execution of hard real-time tasks and adjusting the allocated number of cores dynamically. To achieve this, we present a concept called *allocation vector*, which can serve as the interface between hard real-time tasks and soft real-time tasks: for hard real-time tasks, we derive a schedulability test under the interface; for soft real-time tasks, we discuss the design principle of how to determine the interface to reclaim computing resources as much as possible. We demonstrate the usefulness of the interface and the effectiveness of the proposed approach through examples.**

I. Introduction

This paper considers the mixed-criticality scheduling for parallel real-time tasks under the federated scheduling paradigm. The parallel real-time task is characterized by the volume and the length. The volume is the total workload in this task, and the length is the workload of the longest path in this task. In federated scheduling [1], each heavy task (tasks with the volume larger than its deadline) is assigned and executed exclusively on a set of cores. Therefore, we can restrict our attention to the scheduling of one parallel real-time task on a set of cores.

The federated scheduling suffers from the resource-wasting problem [2], [3] due to the pessimism within its analysis techniques and the overly conservative characterization of parallel real-time tasks. To address the resource-wasting within the scheduling of one parallel real-time task, we propose a mixed-criticality approach by online monitoring the execution of the hard parallel real-time task, and dynamically adjusting the allocated number of cores. Our approach can guarantee the deadline of hard real-time tasks and reclaim computing resources for soft real-time tasks at the same time. To achieve this, we present a concept called *allocation vector*, which can serve as the interface between hard real-time tasks and soft real-time tasks: for hard real-time tasks, we derive a schedulability test under the interface; for soft real-time tasks, we discuss the design principle of how to determine the interface to reclaim computing resources as much as possible.

Our approach only relies on the volume and the length of parallel real-time tasks, not requiring the detailed structure of the parallel task. The usefulness of the introduced interface and the effectiveness of the proposed approach are demonstrated through examples.

II. Related Work

Two closely related works to this paper are [4], [5].

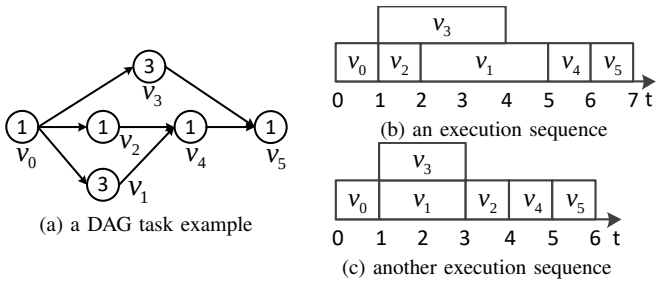
In [4], Agrawal et al. proposed a task model to represent parallel real-time tasks using two pairs of volume and length with different levels of assurance. One pair of volume and length is very conservative and therefore trusted to a very high level of assurance; the other is more representative of the typical execution behavior. The task model enables the scheduling algorithm to dynamically adjust the number of cores assigned to an individual task during the execution of the task. The adjustment of the number of cores is only based on the task model which is obtained by offline profiling the parallel real-time task, so [4] does not utilize the runtime information to reclaim computing resources for soft real-time tasks.

In [5], Baruah extended the method in [4] by combining the worst-case characterizations (i.e., the volume and length) and experimental profiling of execution behavior of parallel real-time tasks. Baruah motivated the work using conditional parallel real-time tasks, since the existence of conditional constructs makes the execution behavior of parallel real-time tasks more complex and the worst-case characterizations more pessimistic.

III. System Model

A. Task Model

A sporadic parallel real-time task is specified as a tuple (G, D, T), where G is the DAG task model, D is the relative deadline and T is the period. We consider constrained deadline, i.e., $D \leq T$. The DAG task model is a directed acyclic graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. Each vertex $v \in V$ represents a piece of sequential workload with worst-case execution time (WCET) $c(v)$. An edge $(v_i, v_j) \in E$ represents the precedence relation between v_i and v_j, i.e., v_j can only start its execution after v_i finishes its execution. A vertex with no incoming (outgoing) edges is called a *source vertex* (*sink vertex*). Without loss of generality, we assume that G has exactly one source (denoted as v_{src}) and one sink (denoted as v_{snk}). In case G has multiple



(a) a DAG task example

(b) an execution sequence

(c) another execution sequence

Fig. 1. An illustrative example.

source/sink vertices, a dummy source/sink vertex with zero WCET can be added to comply with our assumption.

A *path* λ is a set of vertices $(\pi_0, \cdots, \pi_k)$ such that $\forall i \in [0, k-1]$, $(\pi_i, \pi_{i+1}) \in E$. The length of a path λ is defined as $len(\lambda) \coloneqq \sum_{\pi_i \in \lambda} c(\pi_i)$. A *complete path* is a path $(\pi_0, \cdots, \pi_k)$ such that $\pi_0 = v_{src}$ and $\pi_k = v_{snk}$, i.e., a complete path is a path starting from the single source vertex and ending at the single sink vertex. The *longest path* is a complete path with largest $len(\lambda)$ in G. If there is an edge $(u, v) \in E$, u is a *predecessor* of v, and v is a *successor* of u. If there is a path in G from u to v, u is an *ancestor* of v and v is a *descendant* of u. We use $pred(v)$, $succ(v)$, $ance(v)$ and $desc(v)$ to denote the set of predecessors, successors, ancestors and descendants of v, respectively.

Example 1. *Fig. 1a shows a parallel real-time task G where the number inside vertices is the WCET. The deadline and the period $D = T = 7$. v_0, v_5 are the source vertex and the sink vertex, respectively. The longest path is $\lambda = (v_0, v_1, v_4, v_5)$ and $len(\lambda) = 6$. For vertex v_4, $pred(v_4) = \{v_1, v_2\}$, $succ(v_4) = \{v_5\}$, $ance(v_4) = \{v_0, v_1, v_2\}$, $desc(v_4) = \{v_5\}$.*

B. Runtime Behavior

The parallel task G executes on a multi-core platform with identical cores. A vertex v is *eligible* if all of its predecessors have finished, thus v can be immediately executed if there are available cores. The parallel task G is scheduled by any algorithm that satisfies the *work-conserving* property, i.e., an eligible vertex must be executed if there are available cores.

At runtime, vertices of G execute at certain time points on certain cores under the decision of the scheduling algorithm. An *execution sequence* ε of G describes which vertex executes on which core at every time point. For example, two execution sequences of the task in Fig. 1a are shown in Fig. 1b and Fig. 1c. For a vertex v, the *start time* $s(v)$ and *finish time* $f(v)$ are the time point when v first starts its execution and completes its execution, respectively. Note that $s(v)$ and $f(v)$ are specific to a certain execution sequence ε, but we do not include ε in their notations for simplicity. Without loss of generality, we assume the source vertex of G starts execution at time 0, so the *response time* R of G in an execution sequence equals $f(v_{snk})$.

IV. Motivation

This section discusses the scheduling algorithm of a parallel real-time task in federated scheduling, which motivates this work.

For a DAG task $G = (V, E)$, two important characterizations of G are the *volume* and the *length*. The volume (denoted as $vol(G)$) is the total workload in this task and is defined as $vol(G) \coloneqq \sum_{v \in V} c(v)$. The length (denoted as $len(G)$) is the length of the longest path in this task. For example, for the task G in Fig. 1a, $vol(G) = 10$ and $len(G) = 6$. The volume can be measured by executing the task in a platform with one core, and the length can be measured by executing the task in a platform with a sufficiently large number (bounded by the number of vertices in this task) of cores [4].

In [6], Graham proposed a well-known response time bound using the volume and the length of a DAG as follows. The response time R of DAG task G scheduled by work-conserving scheduling on m cores is bounded by (1).

$$R \leq len(G) + \frac{vol(G) - len(G)}{m} \tag{1}$$

Therefore, in federated scheduling, the number of cores m allocated to a DAG G can be computed by (2).

$$m = \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \tag{2}$$

Equation (2) computes the minimum number of cores m such that the response time bound in (1) is no larger than the deadline D.

For a DAG task, the computing resources allocated according to (1) and (2) exhibit several types of pessimism and cause a large amount of resources being wasted, as summarized in the following.

- **Analysis Pessimism.** The bound in (1) is derived by constructing an artificial scenario where vertices not in the longest path do not execute in parallel with the execution of the longest path. However, in real execution, many vertices not in the longest path actually can execute in parallel with the longest path. As observed in [3], this type of pessimism may cause the portion of wasted computing resources arbitrarily close to 100%.
- **Execution Pessimism.** Parameters used in (2), such as $vol(G)$, $len(G)$, are based on the worst case execution time. To comply with the hard real-time requirements, these worst case execution times can be overly pessimistic [7], [8], and the actual execution time can be far less than the WCET, leading to severe resource-wasting during execution.

The analysis pessimism can be partially addressed through improved offline analysis. For example, the technique of intra-task priority assignment can be employed to improve system schedulability [9]–[11]. Under this technique, priorities are assigned to the vertices of a DAG task to control the execution order of vertices and runtime behavior of the task. However, the execution pessimism cannot be mitigated through offline

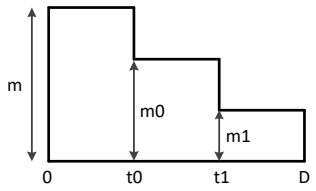


Fig. 2. An illustration of the scheduling for our approach.

analysis, since the required information (such as the actual execution time of vertices) is only available during runtime.

In this paper, we propose an approach to address both two types of pessimism by online monitoring the execution of parallel tasks and adjusting the allocated number of cores dynamically with the target of both satisfying the hard real-time deadline and reclaiming computing resources for soft real-time tasks.

V. Our Approach with Resource Reclamation

In this approach, during the execution of the parallel real-time task, we collect information regarding the execution of the hard real-time task and gradually reduce the allocated number of cores to reclaim computing resources for soft real-time tasks.

A. The Scheduling Algorithm

Definition 1 (Allocation Vector)**.** *For a parallel real-time task (G, D, T), the allocation vector Φ is a set of time points $\{t_0, \cdots, t_k\}$ $(k \geq 0)$ satisfying all of the following conditions.*

1) $\forall i \in [0, k]$, $0 \leq t_i < D$.
2) $\forall i, j \in [0, k]$ *and* $i < j$, $t_i < t_j$.

Given a parallel real-time task (G, D, T) and the allocation vector $\Phi = \{t_0, \cdots, t_k\}$, the scheduling starts at time 0 with the number of cores m computed by (2). During the scheduling, two types of information are monitored.

1) $w(t)$: the volume of the workload executed from time point 0 to time point t.
2) $l(t)$: the cumulative length of time intervals during which some core is idle from time point 0 to time point t.

At each time point t_i, if G does not complete its execution, we adjust the allocated number of cores to m_i. The conditions for m_i will be derived in Section V-B. The scheduling is illustrated in Fig. 2.

B. Schedulability Test for Hard Real-Time Tasks

Definition 2 (Critical Path [9])**.** *The critical path $\lambda^* = (\pi_0, \cdots, \pi_k)$ of an execution sequence is a complete path satisfying the following property.*

$$\forall \pi_i \in \lambda^* \setminus \{\pi_0\} : f(\pi_{i-1}) = \max_{u \in pred(\pi_i)} \{f(u)\} \tag{3}$$

The critical path is specific to an execution sequence of the DAG task G. The critical path of G in an execution sequence is not necessarily the longest path of G.

Example 2. *For the execution sequence in Fig. 1b, the critical path of G is (v_0, v_1, v_4, v_5). In Fig. 1c, the critical path of G is (v_0, v_2, v_4, v_5), which is not the longest path of G. In Fig. 3b, the critical path of G is also (v_0, v_1, v_4, v_5).*

Lemma 1. *In an execution sequence under work-conserving scheduling, when the critical path is not executing, all cores are busy.*

Proof. Suppose that the critical path of this execution sequence is $\lambda^* = (\pi_0, \cdots, \pi_k)$. $\forall i \in (0, k]$, by Definition 2, π_{i-1} is with the maximum finish time among all the predecessors of π_i. This means that when π_{i-1} completes its execution, all predecessors of π_i have completed execution. Therefore, π_i is eligible at $f(\pi_{i-1})$. If some core is idle in $[f(\pi_{i-1}), s(\pi_i)]$, it contradicts the fact that the scheduling is work-conserving. □

Lemma 2. *In an execution sequence under work-conserving scheduling, when some core is idle, the critical path is executing.*

Proof. Lemma 2 is the contrapositive of Lemma 1. □

Theorem 1. *At each time point t_i ($i \in [0, k]$), if the allocated number of cores m_i is computed by (4) and (5), then the parallel real-time task G is schedulable under work-conserving scheduling with the allocation vector $\Phi = \{t_0, \cdots, t_k\}$.*

if $vol(G) - w(t_i) \leq len(G) - l(t_i)$, then

$$m_i = 1 \tag{4}$$

else

$$m_i = \left\lceil \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \right\rceil \tag{5}$$

Proof. Let ε be the execution sequence under analysis of G. At time point t_i, we focus on the remaining graph G_i of G (i.e, the part of G that has not been executed until t_i). We denote the critical path of ε as λ^*. Since l_i is the cumulative length of time intervals before t_i where some core is idle, by Lemma 2, λ^* is executing in these time intervals. Therefore, the length of λ^* in G_i (i.e., the length of λ^* executing after t_i) is at least $len(\lambda^*) - l(t_i)$, which is bounded by

$$len(G) - l(t_i)$$

And the volume of G_i is bounded by

$$vol(G) - w(t_i)$$

By (1), the response time of G_i is bounded by

$$len(G) - l(t_i) + \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{m_i}$$

The new deadline of G_i is $D - t_i$. Let

$$len(G) - l(t_i) + \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{m_i} \leq D - t_i$$

which means (5). □

Note that the volume $vol(G)$ and length $len(G)$ in Theorem 1 are profiled and determined offline. We do not need to monitor these two parameters online.

Corollary 1. *The schedulability test in Theorem 1 dominates the test in [1] (shown in (1) and (2)) in the sense that the computing resource allocated by Theorem 1 is no larger than that of [1].*

Proof. It is sufficient to show that $\forall i \in [0, k]$, the m_i in (5) is no larger than the m in (2), i.e.,

$$\left\lceil \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \right\rceil \leq \left\lceil \frac{vol(G) - len(G)}{D - len(G)} \right\rceil \quad (6)$$

Suppose $t_0 = 0$, we have $w(0) = 0$, $l(0) = 0$, so (6) holds trivially. Therefore, to prove (6), it suffices to show that $\forall i \in [0, k)$, $m_{i+1} \leq m_i$, i.e.,

$$\frac{vol(G) - w(t_{i+1}) - len(G) + l(t_{i+1})}{D - t_{i+1} - len(G) + l(t_{i+1})} \leq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (7)$$

We define

$$\Delta(t) \coloneqq t_{i+1} - t_i$$

$$\Delta(w) \coloneqq w(t_{i+1}) - w(t_i)$$

$$\Delta(l) \coloneqq l(t_{i+1}) - l(t_i)$$

Equation (7) can be rewritten as (8).

$$\frac{vol(G) - w(t_i) - len(G) + l(t_i) - (\Delta(w) - \Delta(l))}{D - t_i - len(G) + l(t_i) - (\Delta(t) - \Delta(l))} \leq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (8)$$

Be aware of the following statement: for $0 < x_1 < x_2$, $0 < y_1 < y_2$,

$$\frac{x_1}{y_1} \geq \frac{x_2}{y_2} \Rightarrow \frac{x_2 - x_1}{y_2 - y_1} \leq \frac{x_2}{y_2}$$

Therefore, to prove (8), it suffices to show that

$$\frac{\Delta(w) - \Delta(l)}{\Delta(t) - \Delta(l)} \geq \frac{vol(G) - w(t_i) - len(G) + l(t_i)}{D - t_i - len(G) + l(t_i)} \quad (9)$$

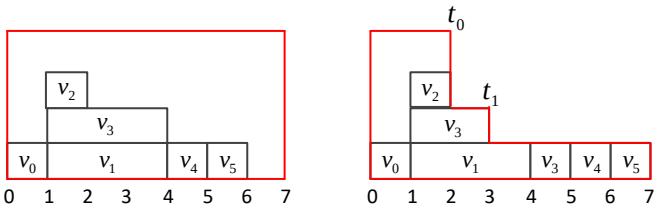
The length of time interval $[t_i, t_{i+1}]$ is $\Delta(t)$. During time interval $[t_i, t_{i+1}]$, the allocated number of cores is m_i. The volume of the workload executed in $[t_i, t_{i+1}]$ is $\Delta(w)$. By the definitions of $l(t)$ and $\Delta(l)$, $\Delta(l)$ is the cumulative length of time intervals during which some core is idle in $[t_i, t_{i+1}]$. Therefore, we have

$$\Delta(w) \geq m_i(\Delta(t) - \Delta(l)) + \Delta(l) \quad (10)$$

which means

$$\frac{\Delta(w) - \Delta(l)}{\Delta(t) - \Delta(l)} \geq m_i$$

Therefore, (9) holds, which means that (6) holds. □



(a) federated scheduling in [1] (b) the proposed approach

Fig. 3. Compare the allocated computing resources between the original federated scheduling and our approach.

C. Design Principle for Soft Real-Time Tasks

This subsection discusses that for a parallel task, how to determine the allocation vector. The objective is that at each time point t_{i+1}, we want to reduce the number of cores m_{i+1} compared to m_i. By Corollary 1, we know that this reduction of cores lies in (10). Therefore, during time interval $[t_i, t_{i+1}]$, the monitoring procedure should observe the type of execution satisfying both of the following conditions.

1) During the execution, at least one core is idle.
2) During the execution, more than one core are busy.

The more this type of execution, the more the number of cores can be reduced. With this guideline, the allocation vector can be determined by offline profiling or dynamically determined during execution.

D. An Example

This subsection provides an example to illustrate the usefulness of the allocation vector interface and the effectiveness of the proposed approach. For the parallel real-time task G in Fig. 1a, suppose that the deadline $D = 7$. The volume $vol(G) = 10$ and the length $len(G) = 6$. Fig. 3 shows the possible execution sequences and the allocated computing resources (circled by red rectangles) under the original federated scheduling in [1] and our approach.

In Fig. 3a, by (2), the allocated number of cores $m = (10-6)/(7-6) = 4$. So the total allocated computing resources (the area of the red rectangle) are $m \times D = 4 \times 7 = 28$. In Fig. 3b, suppose that the allocation vector is $\Phi = \{t_0 = 2, t_1 = 3\}$. At time point 0, same as the original federated scheduling, the allocated number of cores $m = 4$. At time point $t_0 = 2$, the monitored information are $w(t_0) = 4$, $l(t_0) = 2$. $vol(G) - w(t_0) = 6$, $len(G) - l(t_0) = 4$ and $D - t_0 = 5$. By (5), the adjusted number of cores $m_0 = (6-4)/(5-4) = 2$. At time point $t_1 = 3$, the monitored information are $w(t_1) = 6$, $l(t_1) = 2$. $vol(G) - w(t_1) = 4$, $len(G) - l(t_1) = 4$ and $D - t_1 = 4$. Since $vol(G) - w(t_1) \leq len(G) - l(t_1)$, by (4), the adjusted number of cores $m_1 = 1$. So the total allocated computing resources are $4 \times 2 + 2 \times 1 + 1 \times 4 = 14$. Therefore, in this example, our approach reclaims $(28 - 14)/28 = 50\%$ computing resources for soft real-time tasks and guarantees that the hard real-time task meets its deadline.

VI. Conclusion

In this paper, to address the analysis and execution pessimism that lead to the resource-wasting problem in federated scheduling, we propose a mixed-criticality approach by online monitoring the execution of hard parallel real-time tasks, and dynamically adjusting the allocated number of cores to reclaim computing resources for soft real-time tasks. We give an example that illustrates the effectiveness of the proposed approach.

Acknowledgment

The authors would like to thank the anonymous reviewers for their constructive and insightful comments.

References

[1] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of federated and global scheduling for parallel real-time tasks," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, 2014, pp. 85–96.

[2] N. Ueter, G. Von Der Brüggen, J.-J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 482–494.

[3] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and W. Yi, "Virtually-federated scheduling of parallel real-time tasks," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 482–494.

[4] K. Agrawal and S. Baruah, "A measurement-based model for parallel real-time tasks," in *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[5] S. Baruah, "Resource-efficient execution of conditional parallel real-time tasks," in *European Conference on Parallel Processing*. Springer, 2018, pp. 218–231.

[6] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[7] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *Proceedings 22nd IEEE Real-Time Systems Symposium (RTSS 2001)(Cat. No. 01PR1420)*. IEEE, 2001, pp. 215–224.

[8] G. Bernat, A. Colin, and S. M. Petters, "Wcet analysis of probabilistic hard real-time systems," in *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.* IEEE, 2002, pp. 279–288.

[9] Q. He, X. Jiang, N. Guan, and Z. Guo, "Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2283–2295, 2019.

[10] S. Zhao, X. Dai, I. Bate, A. Burns, and W. Chang, "Dag scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency," in *IEEE Real-Time Systems Symposium*. IEEE, 2020.

[11] Q. He, M. Lv, and N. Guan, "Response time bounds for dag tasks with arbitrary intra-task priority assignment," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

A Secure Resilient Real-Time Recovery Model, Scheduler, and Analysis

Abdullah Al Arafat*, Sudharsan Vaidhun†, Bryan C. Ward‡, Zhishan Guo*
*Department of Computer Science, North Carolina State University
†Department of Electrical and Computer Engineering, University of Central Florida
‡Department of Computer Science, Vanderbilt University

***Abstract*—Real-time and embedded systems are increasingly being applied in the command and control of safety- and mission-critical applications such as autonomous vehicles and critical infrastructure. Meanwhile, to enable new capabilities, we are witnessing a rapid growth in the complexity and connectivity of such devices. Unfortunately, such designs often introduce new attack vectors, necessitating inventions that provide stronger security and resilience. This paper presents a secure and resilient scheduling technique for hard real-time applications. Specifically, this approach builds upon the well-known mixed-criticality scheduling framework and demonstrates a new dimension of criticality: security criticality. In the presented model, low-security-criticality workloads are dropped in the event of a malicious event, both to minimize the attack surface, as well as enable the timely scheduling of both a recovery task and the re-execution of the victim task. This paper demonstrates how the existing mixed-criticality scheduling approaches are overly pessimistic in light of this model, and presents a new scheduling algorithm for it. The performance of the presented algorithm and analysis is evaluated through schedulability experiments.**

I. Introduction

Real-time and embedded systems are being employed across society to monitor and control increasingly complex cyber-physical systems. For example, modern automobiles have dozens of onboard computers to control the engine, transmission, braking, driver-assist features, and infotainment systems. In industrial-control applications, the Industrial Internet of Things (IIoT) promises greater efficiencies through increased communication, coordination, and autonomy of industrial processes ranging from power systems to manufacturing.

As a society, we are increasingly reliant upon such systems and enjoy the capabilities and efficiencies such interconnected systems offer. However, the complexity of these systems increases the attack surface, and their increasing connectivity makes vulnerabilities even more accessible to attackers. Furthermore, embedded systems are often not developed with the same cyber-security scrutiny that is common in general-purpose systems. For example, the Mirai botnet [2] exploited the unchanged universal factory-default password to co-opt webcams into a powerful botnet. While many attacks can be detected and/or prevented via known defensive techniques, it is critical that a system is able to respond and recover from such threats while preserving real-time constraints.

Common cyber-security defenses are often eschewed in real-time and embedded systems. For example, address-space layout randomization (ASLR) is employed ubiquitously across general-purpose computing systems and is enabled by default in Windows, Mac OSX, and Linux. But randomization-based defenses are often avoided in real-time systems for predictability reasons as they can significantly increase worst-case performance [7], [9].

The most common class of vulnerabilities are *memory-corruption* vulnerabilities, which are bugs that an attacker can exploit to corrupt regions of memory. Microsoft and Google have reported such vulnerabilities account for approximately 70% of vulnerabilities in their codebases [19], [22]. While there are techniques to eliminate such vulnerabilities, they are expensive (Softbound [18] has overheads over 100%), or impractical (rewrite all code in a memory-safe language such as Rust). Therefore, most runtime defenses protect against memory-corruption-based attacks by seeking to prevent exploitation by crashing the process. For example, control-flow integrity (CFI) [1], [24] performs checks at control-flow transitions to ensure valid branch targets, and crashes the process upon invalid control flow. Importantly, runtime defenses are *integral* to the protected task, that is, they are executed within the protected process, not in a separate process as in monitoring-based security approaches [10], [11], [12], [13]. We note that runtime defenses are designed to prevent exploitation, while monitoring-based approaches detect anomalies and evidence that the system has been compromised. Therefore, monitoring-based security approaches are orthogonal to the runtime defenses we consider, but are outside the scope of this work.

Processes in real-time and embedded systems often control physical devices, and hence cannot simply crash to prevent an attack—such an approach could itself compromise the system. Instead, in such applications, computation could be restarted to maintain continuous safe control of the physical process. However, restarting a real-time job may significantly impact its ability to complete before its deadline, and may introduce additional demand that may in turn compromise the temporal integrity of other tasks in the system, if not properly mitigated.

These observations motivate the need for new task models, scheduling algorithms, and analysis to enable resilience to cyber attacks, *i.e.*, the ability to maintain some safe level of operation while recovering from an attack. While there may be simple or naïve means of supporting such behavior in existing scheduling and analysis frameworks, fully maximizing the platform utilization while enabling such resilience requires fundamentally new models, algorithms, and analysis.

Mixed criticality. This problem has several important commonalities with mixed-criticality (MC) scheduling, specifi-

cally, the ability to operate in a degraded mode of execution. Critically, however, MC scheduling models have principally been developed to handle one aberrant behavior—temporal overruns—not security incidence. However, Burns has recently argued that work on MC systems should be generalized to multi-mode systems [6]. This work is an exemplar of this argument, and we demonstrate a multi-mode system in which mode switches are triggered by security events rather than timing overruns.

There are several important *similarities and differences* between the standard Vestal-model [23] for MC scheduling and the needs of a resilient real-time recovery model. For example, when a security event is detected, it is useful to shed less-critical workloads to ensure the continued correct operation of high-criticality work. Shedding work is especially useful for security as it can also reduce the attack surface of the system. There are, however, several important differences. First, when a defense prevents an attack it crashes the process, requiring re-execution of the job and additional processing time. Another key difference with security criticality is that memory-corruption attacks are most likely to target only a single task, not all high-criticality tasks simultaneously. Memory-corruption attacks target vulnerabilities in code, and because different tasks have different code, they are not likely to be vulnerable to the same exploit payloads.

As a result, adapting existing MC system analysis results will be too pessimistic. In addition, in an MC environment, the system often returns to normal mode when a transient overload condition subsides. In contrast, returning to a normal mode of execution after detection of a cyber threat may require additional recovery processing for computations such as (i) adding the malicious input to a blocklist to ensure the re-executed task will not be attacked, [17] (ii) forensic analysis, (iii) human-operator communication, and/or (iv) other actions to harden the security posture of the system, such as substituting binaries with stronger-defended ones, *etc.* Such additional computation time must also be modeled and analyzed. Notably, shedding less-critical workload, with the proper analysis, frees computation time to enable such recovery processing without affecting the utilization of the normal mode.

Related Works. Previous work has studied co-scheduling security monitor tasks [10], [11], [12], [13] with real-time tasks in fixed-priority partitioned multi-core systems with/without allowing migration of monitor tasks. These papers assume that the security tasks monitor security events and potentially detect the attacks (*i.e.*, works as intrusion detection system (IDS)). However IDS does not *stop* attacks, they merely attempt to detect malicious activity, while run-time defenses (*e.g.*, CFI [8], [24], [27], data flow integrity (DFI) [4]) prevent attacks from succeeding by crashing the process. Note that, unlike IDS, runtime defenses are integral to the task itself, and are *not independently scheduled.* Therefore, detection using runtime defenses is real-time and does not have any scheduling overhead. In SR^3, tasks are instrumented with a runtime defense instead of IDS.

Contributions. Based on these observations, we present the first resilient recovery scheduling model and analysis for secure real-time systems. We identify that temporal criticality and security criticality are orthogonal dimensions of criticality and that by designing a system of differing security criticalities enables both efficient recovery after an attack, as well as the minimization of the attack surface in the presence of a cyber threat. We make the following contributions:

- We propose SR^3, a secure and resilient real-time recovery task model that can recover from an attack at runtime while maintaining the real-time correctness of high-security-critical tasks.
- We develop a scheduling algorithm for the presented task model using earliest-first deadline (EDF) with modified virtual deadlines for security-critical tasks.
- We conduct schedulability evaluations that demonstrate the effectiveness of our scheduling algorithm over adapted existing scheduling schemes.

II. Model and Problem

A. Threat Model

We assume a threat model consistent with other prior works on run-time defenses [8], [24], [27]. Specifically, we assume a write-what-where vulnerability that an attacker can leverage to corrupt code pointers[1] to hijack control flow to attacker-specified location(s). Significant research has shown that even such simple and common vulnerabilities can be exploited using return-oriented programming (ROP) [21] or other attack techniques (*e.g.*, [26]) to completely hijack control flow and implement Turing-complete attacker-controlled logic. This is a very common and powerful threat model.

We assume the system is instrumented with a real-time runtime defense such as control-flow integrity (CFI) [8], [24], [27], [20], data-flow integrity [4], or an address-randomization defense [7], [9]. Notably, all of these defense techniques prevent further exploitation of a task by crashing the process.

Attacks on the scheduler or RTOS itself are outside the scope of our threat model. We note, however, that the scheduler and RTOS can be made trustworthy if using a verified RTOS, (*e.g.*, seL4 [15]), or by using a trusted execution environment (*e.g.*, ARM TrustZone) [25]. Notably, however, attacks related to the *confidentiality* (*e.g.*, side-channel attacks) of the workloads are out of the scope of this work. We also note that IDS as additional security tasks scheduled with regular workloads [10], [11], [12], [13] are outside the scope and, in fact, these models are orthogonal to SR^3.

B. System Model

Let $\tau' = \{\tau_1, \tau_2, ..., \tau_n\}$ be a set of n sporadic and implicit-deadline tasks scheduled on a uniprocessor. Each task τ_i can be represented by three tuple $\{C_i, T_i, \varsigma_i\}$, where C_i is the worst-case execution time (WCET), T_i is the minimal inter-arrival separation as well as the relative deadline (*i.e.*, $D_i = T_i$) of the task instances (jobs). We assume that τ_i is instrumented with runtime security defense(s), and that their overheads are

[1]A code pointer is any address stored in a data section that points to executable code. Return addresses on the stack are a frequent attacker targets.

TABLE I: Tasks of differing temporal and security criticalities.

		Temporal Criticality	
		High	Low
Security Criticality	High	Safety-critical Control Processing	Encryption key management software, or IDS
	Low	Processing non-mission-critical sensor inputs	Infotainment

included within C_i. We assume each task can potentially release an infinite sequence of jobs. Let $\varsigma_i \in \{0, 1\}$ denote whether task τ_i is of high or low security criticality. We use $\tau_\varsigma = \{\tau_i | \varsigma_i = 1\}$ and $\tau_\zeta = \{\tau_i | \varsigma_i = 0\}$ to denote the set of high-security-criticality (HI-security) tasks and low-security-criticality (LO-security) tasks, respectively. We model HI- and LO-security tasks based on the observation that some tasks are not essential to maintain safe or secure operation of the system, especially when the system may be under attack. This is depicted in Table I. For example, in an automotive system, infotainment services are not mission-essential functions, and should neither interfere with high security- or temporal-criticality tasks. Some tasks are also high criticality with respect to both security and temporal criticality, as they support mission-critical functionality. However, there are some tasks that could be critical to the security of the system, but be less critical to the temporal correctness of the system. For example, intrusion detection software or key management for encrypted communication may be critical to the security of the system, even if their timing is not mission critical. Alternatively, some sensor readings may support optional or non-mission-critical functionalities, which could be disabled in the presence of a security threat. However, in order to maintain consistent state, their processing is timing critical.

Note that LO-security tasks may themselves contain vulnerabilities. When the system is under attack, minimizing the attack surface is a valuable defense in and of itself. Given this motivation and model, we define the following terms:

Definition 1. ***(Victim Task and Targeted Task)*** *Any task $\tau_v \in \tau'$ is a victim task when it is attacked during runtime. As control-flow-hijacking attacks leverage one or more vulnerabilities within a single process only, we assume that an attack will target a single task. We assume the attack is detected by the process crashing as a result of a defensive mechanism such as CFI [1], [8], [24], as described in our threat model. Consequently, the attack is detected at or before the task completes its execution budget. We further denote a* HI*-security victim task $\tau_v \in \tau_\varsigma$ as a* Targeted Task*, τ_t, with an execution budget of C_t.*

Definition 2. ***(System Modes)*** *The system will begin its execution under* **normal mode***, during which no attack to security task is detected. During runtime, once a victim task is identified, the system will immediately switch into* **recovery mode***. Proper* actions *(see below) will be taken during recovery mode to prevent the system from further exploitation.*

When transitioning to the recovery mode, additional actions may be taken to facilitate recovery, for example, additional monitoring or validation of the system, forensic analysis, communication with human operators, *etc*. We model this additional workload as a *recovery task*. This task is in addition to the regular HI- and LO-security tasks as defined below:

TABLE II: Workload considered in Example 1.

Task ID	C_i	T_i	ς_i
τ_1	1	3	0
τ_2	2	9	1
τ_3	5	25	1
τ_R	1.5	15	–

Definition 3. ***(Recovery Task)*** *The recovery task $\tau_R = \{C_R, T_R\}$ is a task that is activated/released upon detection of an attack (which only leads to execution failure) during runtime, where C_R is its execution budget and T_R is the period. The release time of the recovery task, r_R, is equal to the system mode switch instant. Note that each* HI*-security task could have an individual recovery task. However, from the analytical perspective, taking $C_R = \max\{C_R^i\}$ covers the worst-case where C_R^i is the WCET of recovery task corresponding to i^{th}* HI*-security task.*

The whole SR^3 system workload contains the HI- and LO-security tasks, as well as the recovery task, *i.e.*, $\tau = \{\tau', \tau_R\}$.

Correctness Criteria. Given the SR^3 system, which contains a set of HI- and LO-security tasks and the recovery task, a *correct* scheduler must

1) guarantee that all HI- and LO-security tasks receive enough execution and meet their deadlines during normal mode;
2) ensure that all HI-security tasks (that are not experiencing any failure, *i.e.*, except the Targeted task) continue to receive normal execution budget and meet their deadlines during recovery mode;
3) ensure that if the victim task is a Targeted task (the failing HI-security[2] task being attacked), then the victim task will receive another full re-execution budget (of its original WCET, C_i) beyond the mode switch point and meets its original deadline;
4) provide the recovery task with enough execution budget before its deadline during recovery mode;

Our objective is to identify a correct online scheduling mechanism and derive an offline schedulability test. Note that once there is a detected attack (and thus a mode switch), guarantees to service of LO-security tasks are no longer required, and this workload is dropped to minimize the attack surface. We assume that the malicious input can be placed in a blocklist, and that either a known-safe or sanitized input is used by the re-executed job. This assumption is consistent with prior work [17].

Unfortunately, the standard uniprocessor scheduling algorithms (*e.g.*, Earliest Deadline First (EDF)) cannot correctly schedule the task set. An illustrative example is given below:

[2]When the victim task is a LO-security one, a mode switch is triggered immediately, while no re-execution budget will be allocated, as no guarantees are provided to LO-security tasks in recovery mode.

Example 1. *Consider a task set* $\tau = \{\tau_1, \tau_2, \tau_3, \tau_R\}$ *with parameters presented in Table II. This regular sporadic task set is schedulable on a uniprocessor system under the earliest deadline first (EDF) scheduler as the utilization* ($\sum \frac{C_i}{T_i}$) *of the task set is* 0.855 *(including the utilization of recovery task).*

Now let us map the SR^3 *system workload to a sporadic task model for EDF scheduling by doubling the execution of* HI*-security tasks,* ($C'_2 = 4, C'_3 = 10$) *and keep the recovery task always active. After mapping the task set to a sporadic task model for EDF scheduling, the utilization of the mapped task set becomes* 1.277*. Therefore, the mapped task set with security awareness is not schedulable by EDF. We will later see that the task set is schedulable under our proposed scheduling algorithm.*

III. Scheduling Algorithm

In this section, we present our proposed scheduling algorithms for the SR^3 system workload. We need to re-execute the targeted task after an adversarial attack following the task model. As demonstrated by Example 1, directly employing EDF scheduler may lead to a *too narrow* scheduling window for HI-security tasks upon attack and thus lead to deadline misses. Therefore, we proposed to adopt the concept of *virtual deadline*, such that the HI-security tasks receive proper 'promotion' under the normal mode.

Let us start with a general overview of our proposed algorithm. At any instant in normal mode, we aim to promote the execution of jobs of HI-security tasks over LO-security tasks, maintaining the deadline constraints of all tasks. To do so, we compute a virtual deadline $D_i^v = x \cdot D_i$ for each HI-security task such that the virtual deadline is less than or equal to the original deadline of the tasks (*i.e.*, no task exceeds the original deadline). After computing a suitable virtual deadline for each HI-security task, HI-security tasks are scheduled using their virtual deadline and LO-security tasks with their original deadline following the EDF algorithm. The window between the virtual and actual deadlines for each HI-security job is 'reserved' for the HI-security task's potential re-execution upon attack/mode switch. Further, in recovery mode, all LO-security tasks are dropped immediately at system mode-switch instant. Then, in recovery mode, all HI-security tasks and the recovery task are scheduled following the EDF algorithm using the original deadlines.

We present a way of determining the virtual deadline of the HI-security tasks and so the schedulability test of the scheduling algorithm. In Subsection III-A, we present the utilization-based schedulability analysis of the proposed algorithm (we denote it as sEDF-VD to distinguish it from EDF-VD [3]).

A. sEDF-VD

Given a sporadic implicit-deadline task set τ, one needs to perform a schedulability test for the task system prior to the runtime to determine whether the task system is schedulable or not. If the task system is schedulable, the sEDF-VD finds a virtual deadline $D_i^v = x \cdot D_i$ for all HI-security tasks via a common 'shrinking factor' $x \in (0, 1]$. In Algorithm 1, we present an offline schedulability test for the task set. The algorithm returns a common shrinking factor x for each HI-security task if the task set is schedulable, or FAILURE if the task set is not schedulable under the correctness criteria presented in Section II. The initial shrinking factor x in Line 1, Algorithm 1 comes from Lemma 1 and the conditional statement in Line 3, Algorithm 1 from Lemma 2. Both Lemmas are discussed in the following.

Algorithm 1: sEDF-VD based Schedulability Test (Virtual Deadline Setting)

Input: A SR^3 system workload $\tau = \{\tau_\varsigma, \tau_\natural, \tau_R\}$

1 $x \leftarrow \frac{U_\varsigma}{1-U_\natural}$; // common shrinking factor for all $\tau_i \in \tau_\varsigma$

2 **for** $\forall \tau_t \in \tau_\varsigma$ **do**

3 **if** $xU_\natural + U_\varsigma + u_t + u_R > 1$ **then**

4 return FAILURE; // no x that satisfy Theorem 1

5 **end**

6 **end**

7 return x;

We need to determine a feasible range of x and demonstrate its correctness. First we introduce additional notation.

Utilization parameters. The *utilization* of an implicit-deadline sporadic task is the ratio of its WCET (C_i) to the period (T_i). The utilization of the task system is the summation of all individual tasks in the set.

- $u_i = \frac{C_i}{T_i}$ is the utilization of task τ_i.
- $U_\natural = \sum_{\tau_i \in \tau_\natural} u_i$ is the utilization of LO-security task set.
- $U_\varsigma = \sum_{\tau_i \in \tau_\varsigma} u_i$ is the utilization of HI-security task set.

We will now derive the schedulability constraints of the SR^3 system workload considering the presence of a targeted task ($\tau_t \in \tau_\varsigma$) instead of any victim task through Lemma 1 and 2.

Lemma 1. *In* **normal mode**, *all jobs of* LO*-security tasks meet their actual deadline and all jobs of* HI*-security tasks meet their virtual deadline under sEDF-VD for the following sufficient inequality condition,*

$$x \geq \frac{U_\varsigma}{1 - U_\natural} \tag{1}$$

Proof Sketch. Let us consider a fluid schedule [14] (a conceptual scheduling scheme where each task gets a uniform execution rate over the scheduling period and the scheduler is schedulable if the total execution rate of all tasks is less than or equal to one), where each task in the task set is continuously assigned an execution rate of u_i for each LO-security task. Now, for HI-security tasks, we use virtual deadline deduced by multiplying the actual deadline by $x(\leq 1)$. Therefore, the fluid scheduler will assign a continuous execution rate of $\frac{u_i}{x}$ for each HI-security task. So, using the utilization bound of EDF [16], the task set will be schedulable if,

$$\sum_{\tau_i \in \tau_\natural} u_i + \sum_{\tau_i \in \tau_\varsigma} \frac{u_i}{x} = U_\natural + \frac{U_\varsigma}{x} \leq 1$$

So, the fluid schedule is feasible as the total utilization is less or equal to one. As the EDF in preemptive uniprocessor is optimal, the fluid schedule is also feasible by EDF.

Lemma 2. *In* **recovery mode**, *all* HI*-security tasks meet their actual deadlines, and the recovery task meets its deadline under EDF, if*

$$xU_{\setminus} + U_{\varsigma} + u_t + u_R \leq 1 \tag{2}$$

where u_t and u_R is the utilization of targeted task ($\tau_t \in \tau_{\varsigma}$) and recovery task, respectively.

Proof Sketch. Let us consider contrapositive. Suppose a job misses its deadline. Being in recovery mode, all LO-security jobs have been dropped and cannot miss a deadline. Therefore, the job missing its deadline must be a HI-security task. Let us consider a minimal instance[3] of jobs released by the task set, I, on which one job missed the deadline. Without loss of generality, we consider the earliest release time of a job in I (the last idle instant) as zero (0), and the deadline missed instant of a job in I as t_d. Let t^* denote the mode switch instant triggered by the job of targeted task τ_t. Note, the mode switch instant must be no later than the job's virtual deadline, *i.e.*, $t^* \leq D_t^v$.

Note that all jobs in I must experience some execution in $[0, t_d)$ except the job missed deadline at t_d. Let us consider the earliest release time of the job J amongst those executed in $[t^*, t_d)$ is a, and its deadline d. We will calculate the total executions of jobs in set I for four mutually exclusive subsets—subset of jobs from LO-security tasks, other HI-security tasks[4], the targeted task, and the recovery task separately.

Subset-1. Any LO-security task $\tau_i \in \tau_{\setminus}$ in I has an execution w_i in the considered scheduling window,

$$w_i \leq (a + x(t_d - a))u_i \tag{3}$$

Subset-2. Any *other* HI*-security task* $\tau_i \in \tau_{\varsigma} \setminus \tau_t$ in I has an execution w_i in the considered scheduling window,

$$w_i \leq \frac{u_i}{x}a + (t_d - a)u_i \tag{4}$$

Subset-3: The *targeted task* $\tau_t \in \tau_{\varsigma}$ has an execution,

$$w_t \leq \frac{a}{x}u_t + (t_d - a)2u_t \tag{5}$$

Subset-4. The *recovery task* τ_R has an execution of,

$$w_g \leq (t_d - a)u_R \tag{6}$$

Now, total execution of the jobs in I would be greater than the scheduling window length, t_d to miss a deadline.

$$\left(\sum_{\tau_i \in \tau_{\setminus}} w_i\right) + \left(\sum_{\tau_i \in \tau_{\varsigma} \setminus \tau_t} w_i\right) + w_t + w_g > t_d$$

$$\Rightarrow xU_{\setminus} + U_{\varsigma} + u_t + u_R > 1; \quad \text{(simplified using eq. 3,4,5,6)}$$

Thus, the minimal job instances, I will be schedulable if,

$$xU_{\setminus} + U_{\varsigma} + u_t + u_R \leq 1$$

[3]By minimal instance, we mean any set of jobs from which reducing one job would make the jobs set schedulable.

[4]By 'other HI-security tasks', we refer all HI-security tasks but the targeted task ($\tau_{\varsigma} \setminus \tau_t$).

Hence *Lemma 2* follows.

Using the Lemma 1 and 2, we get following scheduling test for sEDF-VD:

Theorem 1. *A* SR^3 *system workload τ, where the victim task is a targeted task, can be successfully scheduled by sEDF-VD on an uniprocessor if the following (sufficient) conditions hold:*

$$(A) : x \geq \frac{U_{\varsigma}}{1 - U_{\setminus}}; \quad \textit{[from Lemma 1]}$$

$$(B) : x \leq \frac{1 - U_{\varsigma} - u_t - u_R}{U_{\setminus}}, \forall \tau_t \in \tau_{\varsigma}; \quad \textit{[from Lemma 2]}$$

Example 2. *Let revisit the task set given in Example 1. Using Theorem 1 (A), we get $x \geq 0.633$ and for Theorem 1 (B), $x \leq 0.766$ for the task set. Note that, in condtion (B) of Theorem 1, we need to find the lowest value of x which can be found using $\max\{u_t | \tau_t \in \tau_{\varsigma}\}$ in calculation of x. So there is an x that satisfy both of the conditions of Theorem 1. Therefore, the task set is schedulable for sEDF-VD.*

Note that if we map the SR^3 system workload of Table II to mixed-criticality model [23] doubling the execution time of all HI*-security tasks in recovery mode ($C_i^n = C_i, C_i^e = 2C_i, \forall \tau_i \in \tau_{\varsigma}$) and the recovery task, τ_R as ($C_R^n = 0, C_R^e = C_R$), then the lower limit of shrinking factor x by Theorem 1 of [3] is 0.6333 and the upper limit by Theorem 2 of [3] is 0.1666. Therefore, there is no x that satisfy the schedulability constraints of mixed systems by EDF-VD presented in [3] for this mapped task set.*

IV. Evaluation

In this section, we evaluate our proposed algorithm. We will first explain the baseline algorithms that we consider. Next, we will present the workload generation procedure used to generate random task sets. Finally, we present the simulation results and discuss observations.

Baselines. The first baseline algorithm that we consider is the earliest deadline first (EDF) algorithm [16]. The EDF algorithm is used to schedule a workload that follows the sporadic task model and therefore to utilize the EDF algorithm, we map our proposed SR^3 system to the standard sporadic task model by doubling the utilization of each HI-security task in the system to account for the worst-case. Additionally, we also include the recovery task in the task set. With our proposed task model mapped to the sporadic task model, we use the utilization-based schedulability test for EDF algorithm to determine the schedulability.

The second baseline algorithm that we considered is the earliest deadline first with virtual deadline (EDF-VD) algorithm. EDF-VD algorithm is a widely accepted scheduler for the Vestal's mixed-criticality task model. We map our SR^3 system to the mixed-critical task model by allocating twice the utilization in the recovery mode for the HI-security tasks. We also add the recovery task as a HI-criticality task to the system where the normal execution budget of the recovery task is assumed to be 0. With these modifications, we apply the

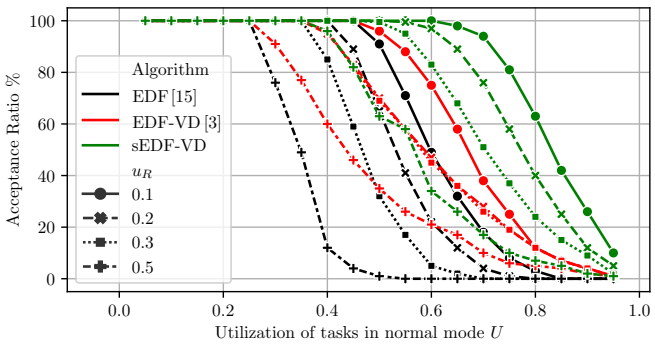


Fig. 1: Acceptance ratio of three different algorithms under multiple utilization settings

EDF-VD schedulability test [3] to determine the schedulability of the task set.

Workload generation. The SR^3 task set generation is controlled by the following parameters, where the default values are represented in bold.

- $n = \{5, \mathbf{10}, 15, 20\}$: Number of tasks in a task set
- $u_R = \{0.1, 0.2, \mathbf{0.3}, 0.5\}$: Utilization of the recovery task
- $U = U_\iota + U_\varsigma = \{x/20 \mid 1 \leq x < 20\}$: Total utilization of the task set in normal mode
- $P = \{0.1, 0.2, \mathbf{0.5}, 1.0\}$: Probability of a task being HI-security task

The task set generation begins with a target value for normal mode utilization given by U. Using the UUniFast algorithm [5], we derive the set of task utilizations in normal mode. The recovery task utilization in recovery mode is given by the u_R parameter. For each setting, we generate 1000 task sets and present the results below.

Figure 1 reports the variation in acceptance ratios for varying system utilizations under different recovery task utilizations.

Observations. When applying EDF, it is seen in Figure 1 that, the acceptance ratio begins to drop as U increases beyond 0.5, irrespective of the recovery task utilization. This can be explained by the added pessimism to be considered by the EDF algorithm in the recovery mode. As the recovery task utilization increases, the performance further decreases. This behavior can be attributed to the added workload contributed by the increasing utilization of the recovery task.

When the EDF-VD algorithm is modified to schedule the proposed task model, the performance follows a similar trend as the EDF algorithm as shown in Figure 1. This pattern is consistent for all values of the recovery task. Similar to EDF, this observation can be attributed to the added pessimism in the higher criticality level due to the re-execution in the recovery mode. The pessimism arises from the need to cover the worst-case scenarios where a task re-execution can be triggered.

V. Conclusion

We have presented SR^3, a secure and resilient real-time attack recovery model, scheduler, and analysis. This model demonstrates that security criticality is an orthogonal dimension of criticality than has been studied in prior work on mixed-criticality scheduling. Our model is an example of a multi-mode mixed-criticality system, in which there are two modes, normal and recovery, and tasks are either high- or low-security criticality. Additionally, to facilitate recovery from a security event, a recovery task executes during recovery mode.

To avoid pessimism when adapting existing MC analysis, we developed a uniprocessor scheduling algorithm with modified virtual deadline for each HI-security task, and provided utilization-based schedulability test. Finally, we experimentally show that SR^3 performs better than the existing uniprocessor scheduling schemes such as EDF and EDF-VD for mixed-criticality systems upon model transformation via simulation on synthetic workload.

References

[1] M. Abadi et al. Control-flow integrity. In *ACM Conference on Computer and Communications Security*, CCS, 2005.

[2] M. Antonakakis et al. Understanding the mirai botnet. In *USENIX Security '17*. USENIX Association, Aug. 2017.

[3] S. Baruah et al. The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems. In *ECRTS '12*, 2012.

[4] N. B. Bellec et al. RT-DFI: Optimizing data-flow integrity for real-time systems. In *ECRTS '22*, 2022.

[5] E. Bini and G. C. Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

[6] A. Burns. Multi-model systems — an mcs by any other name. In *8th International Workshop on Mixed Criticality Systems*, 2020.

[7] N. Burow et al. Moving target defense considerations in real-time safety- and mission-critical systems. In *Proceedings of the 7th ACM Workshop on Moving Target Defense*, pages 81–89, 2020.

[8] Y. Du et al. Holistic Control-Flow protection on Real-Time embedded systems with kage. In *USENIX Security '22*, 2022.

[9] J. Fellmuth et al. Instruction caches in static WCET analysis of artificially diversified software. In *ECRTS '18*, 2018.

[10] M. Hasan et al. Exploring opportunistic execution for integrating security into legacy hard real-time systems. In *RTSS '16*. IEEE, 2016.

[11] M. Hasan et al. Contego: An adaptive framework for integrating security tasks in real-time systems. *ECRTS '17*, 2017.

[12] M. Hasan et al. A design-space exploration for allocating security tasks in multicore real-time systems. In *DATE '18*. IEEE, 2018.

[13] M. Hasan et al. Period adaptation for continuous security monitoring in multicore real-time systems. In *DATE '20*. IEEE, 2020.

[14] P. Holman and J. H. Anderson. Adapting pfair scheduling for symmetric multiprocessors. *Journal of Embedded Computing*, 2005.

[15] G. Klein et al. seL4: Formal verification of an OS kernel. In *SOSP*, 2009.

[16] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, Jan. 1973.

[17] J. S. Mertoguno et al. A physics-based strategy for cyber resilience of cps. In *Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2019*, 2019.

[18] S. Nagarakatte et al. SoftBound: Highly compatible and complete spatial memory safety for C. PLDI, 2009.

[19] C. Project. Memory safety, 2020.

[20] G. Serra et al. PAC-PL: Enabling control-flow integrity with pointer authentication in FPGA SoC platforms. In *RTAS '22*, 2022.

[21] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *CCS '07*, 2007.

[22] G. Thomas. A proactive approach to more secure code, 2019.

[23] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *RTSS*, 2007.

[24] R. J. Walls et al. Control-flow integrity for real-time embedded systems. In *ECRTS '19*, 2019.

[25] J. W. Wang et al. RT-TEE: Real-time system availability for cyber-physical systems using ARM TrustZone. In *IEEE S&P*, 2022.

[26] B. C. Ward et al. The leakage-resilience dilemma. In *ESORICS 2019*, page 87–106.

[27] J. Zhou et al. Silhouette: Efficient protected shadow stacks for embedded systems. In *USENIX Security '20*, 2020.

Mixed-Criticality Wireless Communication for Robot Swarms

Sven Signer
Department of Computer Science
University of York
York, United Kingdom
sven.signer@york.ac.uk

Alan G. Millard
Department of Computer Science
University of York
York, United Kingdom
alan.millard@york.ac.uk

Ian Gray
Department of Computer Science
University of York
York, United Kingdom
ian.gray@york.ac.uk

***Abstract*—In recent years the mixed criticality systems model has been adapted for use in shared-medium communication protocols, but it has not seen deployment into swarm robotics. This paper discusses ongoing work in the application of such a model to this domain, and argues for the benefits of such an approach. In many applications, reliability of communications is essential for the correct and safe operation of the robots. Given the inherently unreliable nature of wireless inter-robot communications, this paper argues for the application of timing- and criticality-aware communication protocols to be able to provide more reliable task-level performance of swarm robotics applications. In this work we define two illustrative swarm applications with two tasks at different criticality levels. Using simulation results we show that in the presence of wireless faults, standard best-effort protocols will cause application errors unpredictably, but a mixed-criticality wireless protocol can maintain important tasks at the cost of less important ones for longer.**

***Index Terms*—real-time wireless, mixed-criticality, swarm communication**

I. Introduction

Distributed autonomous systems rely on wireless communications to implement their functionality. If such systems are to be deployed in high-integrity environments, such as autonomous vehicles, then it is necessary to be able to reason about the performance of the system in situations where such communications are not reliable. Existing approaches in the field of swarm robotics rarely consider timing-aware communication, instead relying on mechanisms such as self-organisation and emergence for information propagation [1]. Failed communications can be corrected through retransmissions, but these also reduce available bandwidth and can cause further transmission failures. A standard best-effort protocol like WiFi does not allow the system integrator to analyse system performance ahead of time.

This paper argues that by implementing timing-aware communications protocols and by adopting a mixed criticality system model, it becomes possible to provide hard timing guarantees within a specified fault model, and to reason about system degradation in a controlled way when that model is exceeded. Our results show this translates into better task-level performance for an example swarm robotics application.

Section II begins by defining a motivating problem for this work to address. We then introduce mixed criticality in Section III and examine existing wireless protocols in section IV. We define our system model (Section V) and our experiment in Section VI, followed by results (Section VII), limitations (Section IX) and conclusions.

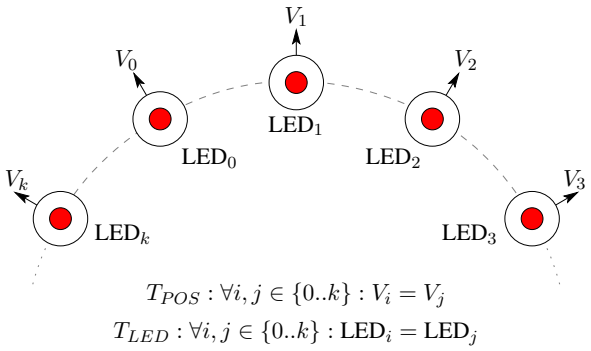


Fig. 1. k robots, each having a velocity V and an LED colour. Task T_{LED} requires all nodes to show the same colour. Task T_{POS} requires all nodes to move with the same velocity. At any given time, both conditions should be satisfied.

II. Motivating Problem

Consider an autonomous wireless swarm robotics platform in which the robots have two tasks:

- T_{LED}: Communicate to form a consensus about what LED colour to display. Each agent may choose to ask the swarm to display a different colour at any time, for example as a response to external stimuli.
- T_{POS}: Coordinate to maintain a circle formation. Each agent may choose to adjust the formation and the others must maintain relative positioning.

This system has two metrics of quality: timing error for task T_{LED}, and positional error for task T_{POS}. The system is trying to minimise both errors. Errors are discussed in more detail in Section VI. Communications are wireless, and so any given transmission has a probability of successful transmission, which decreases as traffic volume increases (due to collisions) and decreases as inter-agent distance increases (see Section V). A naive approach to this problem simply has each robot communicate with every other robot for either task, retransmitting if a transmission fails. However, this means that as the robots get further from each other, transmissions begin to dominate the available bandwidth and errors increase.

If we now introduce the constraint that T_{POS} is a high-integrity task that must be guaranteed to avoid the potential for injury, we need to be able to guarantee a given level of performance for T_{POS}. This guarantee becomes impossible (or at least incredibly pessimistic) due to unbounded interference from the rest of the system.

III. Mixed Criticality Systems

Given the problem introduced in Section II, this work argues that future autonomous swarm robotics systems should be viewed as a mixed criticality system (MCS). The MCS system model was initially motivated by the expectation of deadline overruns caused by imprecise timing analysis, however in a swarm robotics system a common source of issues is that of wireless communication delays. Even in an otherwise well-formed network, transient faults can cause unpredictable sporadic delays to task execution.

In traditional automotive or avionics systems, criticality levels are kept insulated from each other on dedicated control units and networks to make it easier to demonstrate that lower-criticality tasks cannot interfere with the execution of higher-criticality tasks. In the motivating example from Section II, this is not possible due to the shared communication medium.

This work uses the most commonly deployed form of MCS in which the system has two criticality levels, *LO* and *HI*, although many refinements have been made since [2]. In the basic model, each task is assigned to one of these levels, resulting in a set of *LO* tasks and a set of *HI* tasks. Each task also has a WCET value (C) for each criticality level. A task's *LO* WCET (C_{LO}) might come from a simple measurement-based approach and so therefore may be optimistic. Its C_{HI} on the other hand might come from an analytical approach and so is safe, but pessimistic. For all tasks, $C_{LO} < C_{HI}$.

Whilst this approach was initially developed in the context of tasks executing on a CPU, it can be applied to wireless communications [3]. This paper shows that by thinking of the overall behaviour of a robot swarm in terms of criticality levels, the system performance can be gracefully degraded (for example as communication becomes less reliable) in a way that allows key functionality to be retained for longer.

IV. Related Work

Ad-hoc mesh networking is a well-studied problem in the field of autonomous systems. Early work such as BATMAN [4] looked at the problem of communicating nodes with a topology that is not known *a priori*. However it only provided limited support for mobility and little control over traffic prioritisation.

Real-Time Wireless Multi-hop Protocol (RT-WMP) [5] was a major expansion to the research space through the introduction of a real-time wireless protocol that could support a limited form of end-to-end guarantees on a multi-hop wireless network. RT-WMP includes a significant consensus phase in which nodes can coordinate to determine which has the most important traffic and over what routes it should be sent. This leads to predictable performance, but causes significant bandwidth overheads and requires nodes to expend a lot of power when compared to simpler protocols. Some of these problems have been later addressed through Beluga [6] which exploits flow periodicity to reduce coordination overheads, although this work still assumes fixed traffic flows and cannot support support simultaneous transmissions in different parts of the network.

WirelessHART [7] is an extension of the HART protocol focused on wired communication in industrial automation and process control. It uses time-division multiple access (TMDA) at the physical layer with centralised route planning. This allows it to achieve excellent levels of predictability, but in some situations it suffers from poor utilisation because it also cannot support simultaneous transmissions.

Glossy [8], and the wireless protocols that build upon it such as the Low-Power Wireless Bus [9] and Blink [10], use a different approach that assumes no network topology information is available. Instead, all packets are flooded through the network using simultaneous retransmissions within a single timeslot. With sub-microsecond clock synchronisation, the frames interfere constructively rather than destructively. Since network topology is not considered, Glossy requires transmission slots sufficiently large for such a flood to reach all nodes in the network. While this is acceptable for networks with a small number of maximum hops, it prevents efficient operation in larger networks. Further, the protocol cannot handle simultaneous transmissions.

AirTight [11] is a decentralised mixed-criticality protocol that provides real-time guarantees that are resilient to network interference. Access to the network is mediated by a slot table that is assigned ahead-of-time, but each node uses local scheduling decisions to determine which frame is sent during an assigned transmission slot. Unlike CPU scheduling applications, where the criticality level influences the worst case execution time, AirTight assumes that the size of a frame is constant. Instead, it is the level of interference a task must be able to endure that varies by criticality level. A "criticality-aware" fault model bounds the maximum level of interference for a given criticality level. The protocol then guarantees that the worst-case response times computed at a given criticality level will not be exceeded so long as the actual number of transmission failures experienced does not exceed the value predicted by the fault-model for that criticality level. If, at runtime, the failure bound computed by the fault model at a given criticality level is exceeded, the system moves to the next higher criticality level. Nodes in high criticality mode drop low criticality transmissions in order to increase the number of slots available for the high criticality flows. AirTight allows simultaneous transmissions [12], both from spatially separated nodes on the same channel, and through the use of multiple channels.

Inter-robot wireless communication in the field of swarm robotics often defaults to protocols such as WiFi, ZigBee, and Bluetooth. These all can offer potentially very high throughput with excellent robustness, but they rely on protocol or MAC-layer features like random backoff and retransmissions, and

result in priority inversion (where high-importance flows can be interrupted or blocked by low-importance flows). Additional robustness is sometimes added at the application layer, such as through distributed data structures that are resistant to imperfect communication [13], but these are typically not suitable for timing critical data. Prior work [14] illustrates how realistic, i.e. imperfect, communications can prevent swarm robotics applications from functioning correctly.

This paper will explore how swarm applications can fail due to wireless communication without timing awareness, and discuss how the application of AirTight and a mixed criticality system model can aid robustness.

V. Simulated Transmission Model

In order to demonstrate the use of these models, we employ an established swarm robotics simulator, ARGoS [15]. The ARGoS simulator provides integrated support for radio communication through the *simple_radio* interface, but this assumes perfect communications within a given range and does not handle packet collisions. In order to better model realistic communications, we have developed a custom radio communication plugin using an alternative transmission model

We assume that each simulation step is equivalent to one transmission slot, such that each node can only send or receive a single frame during a simulation step. If a robot is able to 'hear' multiple frames within a single slot, we define that the frames interfere destructively such that no frame can be correctly decoded.

Our model is such that the effective packet delivery rate of a link is inversely proportional to the square of the distance between two nodes, and that successful or unsuccessful delivery is determined independently for each link and transmission. We note that is an extremely simple model which does not capture the true complexity of real wireless communications. Previous experiments [16]–[19] have produced conflicting results, but generally show a weak correlation between node distance and packet reception rates, which is highly dependent on the specific testing environment.

Attempting to accurately model a wireless radio's physical layer and resulting radio performance is beyond the scope of this paper, however our experiments do not rely on the specifics of the fault model beyond determining when packets are received. Rather we use this model to show how swarm behaviour changes as packet reception rates decrease. Therefore, we believe the results of the simulation should be broadly applicable regardless of the simplification to the fault model.

Using this simulation, we compare the performance of AirTight with two baseline protocols:

- Broadcast: Nodes broadcast each message a fixed number of times using carrier sensing to reduce collisions.
- Point-to-Point: Nodes transmit messages to each other node in turn, a CSMA/CA like protocol using carrier sensing and random backoff between retransmissions until an acknowledgement is received or a maximum number of retries has been reached.

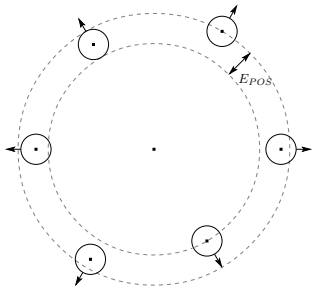


Fig. 2. Circle formation showing positional error E_{POS}, the maximum difference in distance from the circle's central point of any two nodes.

A. AirTight Fault Model

If arbitrarily many transmissions can fail over an indefinite time period, it is not possible to provide any timing guarantees. Therefore, the AirTight protocol [11] requires the number of failures within a time period to be bounded by a fault model that is provided ahead of time during response time analysis.

When considering networks with stationary nodes, the original AirTight specification considered failed transmission slots as a result of external interference causing blackout periods. Here, we instead consider failed transmission due to a reduced packet delivery rate (PDR), presumably as a result of distance between nodes.

Since the PDR is simply the delivery probability for a single packet, and the delivery of each packet is defined to be independent, our fault model must be probabilistic. The probability of a given number of failed transmissions occurring within a busy-period of t transmissions can be modelled using a simple binomial distribution. For a given criticality level L, minimum assumed PDR and a desired confidence bound, the fault model $F(L,t)$ computes a maximum number of failed transmission slots as the smallest integer m satisfying inequality 1, in which we ensure the desired confidence bound is met by accumulating the probability of exactly k failures occurring for all $k \leq m$.

$$\sum_{k=0}^{m} \binom{t}{k} \cdot \left(1 - \text{pdr}(L)^2\right)^k \cdot \left(\text{pdr}(L)^2\right)^{t-k} \geq \text{conf}(L) \quad (1)$$

Note that the PDR is squared in the above equation to account for lost acknowledgements, which are assumed to occur with the same probability as lost data frames. We do not consider dependent PDRs, i.e. blackout periods due to external interference, in this model but note that it would be possible to extend this fault model to include them. In the most basic case, an additional fault-model accounting for static blackout periods with a length and period could simply be summed to the existing fault-model, albeit at the cost of some pessimism.

VI. Experimental Setup

Building on the motivating problem in Section II, we contrive a concrete instantiation. A set of 6 mobile nodes with multicoloured LEDs are arranged in a circle facing outwards,

1m apart, and pre-programmed to assume the same initial LED colour. To satisfy tasks T_{LED} and T_{POS}, all nodes should show the same LED colour and must move radially outward at the same speed in order to maintain the circle formation.

The LED colour and movement speed is determined in a decentralised manner by the nodes. Each node may change the LED colour for the entire swarm if it has been at least one second since it last initiated an LED colour change. This reflects real-world tasks such as a swarm responding to distributed sensing of an environment. When a node proposes an LED colour change, the swarm must coordinate to change to the new colour in unison within two seconds. Multiple such colour changes can be queued to take effect at different future times. If multiple nodes propose an LED colour change for the same time, the conflict is deterministically resolved by a predetermined static priority ordering. The movement speed for the swarm is determined in an equivalent manner, except that these events may only be generated once every five seconds, and take effect after ten seconds. We assume that each frame is only large enough to contain either exactly one LED colour change message or one movement speed change message.

The performance of the system is quantified by measuring two errors: E_{POS}, the maximum difference in effective circle radius of any two nodes, as shown in Figure 2, and E_{LED}, the number of nodes showing an incorrect LED colour. If all nodes could communicate perfectly, E_{POS} and E_{LED} would remain zero.

In a real application, these changes would be triggered by local sensing onboard the nodes. For the purposes of this simulation, we simply assume that after the minimum inter-arrival period has been reached, there is a uniform probability of 2% per node per simulation step for LED changes, and a uniform probability of 1% per node per simulation step for movement speed changes. New LED colours are chosen as a uniformly random RGB value, whereas movement speed is chosen with each node having a bias towards a slower/faster speed, such that positions will diverge if communication fails.

As in Section II, the task T_{POS} is a high-integrity task that should be protected in the event of a system degradation. The AirTight protocol allows this by setting the packet flows of T_{POS} as high-criticality, while those of T_{LED} as low-criticality. The other two protocols have no notion of criticality[1], thus all packets are handled equally.

Since the circle setup results in a constantly increasing distance between agents, the packet delivery ratio is constantly decreasing according to our transmission model. Thus, regardless of protocol, communication between the nodes will eventually fail, causing E_{POS} to increase indefinitely. The effectiveness of a protocol can therefore be determined by the length of time that E_{POS} remains below a threshold value.

[1]Note that some carrier sense protocols have the notion of *priority* that allows smaller inter-frame times for packets that need low latency. This is not same as *criticality* which affects how the system degrades under failures.

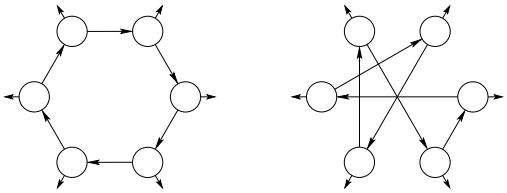


Fig. 3. Node setup showing optimal routing and an example of possible randomised routing.

A. AirTight

An AirTight deployment will analyse the flows and topology ahead of time to ensure that flows are schedulable [11]. For this example the protocol is set up with a slot-table of equal length to the number of nodes, with each node assigned a single unique transmission slot of 10ms. For each node, there are LED and movement flows that begin at that node and proceed around the outside of the circle. We assume that the system integrator wants to prioritise movement accuracy, so assigns LED flows to low-criticality and movement instructions to high-criticality. The overall LED flow is partitioned with the manually selected per-hop deadlines of 190ms, 310ms, 370ms, 430ms, and 550ms (giving an end-to-end deadline of 1850ms), whereas the movement flow is partitioned with the manually selected per-hop deadlines of 1570ms, 1750ms, 1870ms, 1990ms, and 2110ms (giving an end-to-end deadline of 9290ms). Note that these end-to-end deadlines are within the timing requirements as specified at the beginning of this section.

The AirTight protocol in this scenario has the inherent advantage that it has been pre-programmed with *a priori* knowledge of the network topology. To observe its behaviour when this advantage is removed we also test the AirTight protocol where the ordering of the nodes has been randomised, as shown in Figure 3.

The fault model (described in Section V-A) is configured to require confidence bound of 99% given a minimum PDR of 96.3% (corresponding to a distance between nodes of 2 metres) in low criticality mode, and a confidence bound of 99.999% given a minimum PDR of 79.2% (corresponding to a distance between nodes of 3 meters) in high criticality mode.

B. Broadcast

The broadcast protocol is setup to transmit each message 13 times. This number is chosen as the maximum fixed number of transmissions without exceeding the available bandwidth.

C. Point-to-Point

The point-to-point protocol is setup to transmit each message to each other node in turn, until it receives and acknowledgement frame or it has sent the frame 5 times. This maximum number of transmission was determined experimentally as a value that suitably balances the need for retransmissions without causing excessive collisions or triggering a continuous buildup of frames in the transmission buffers.

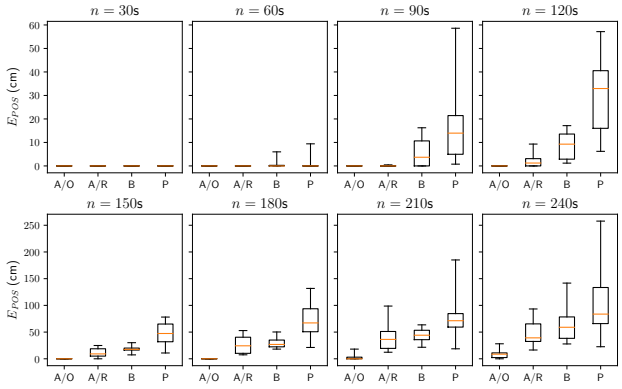


Fig. 4. E_{POS} after n seconds for simulated nodes using the AirTight protocol with optimal (A/O) and randomised (A/R) routing, Broadcast protocol (B), and Point-to-point protocol (P) over 10 different random seeds. Note that the y-axis scaling changes between rows.

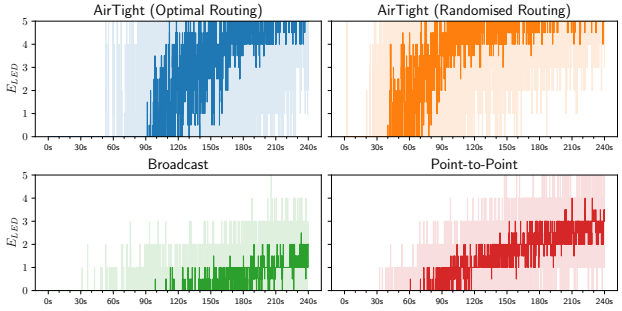


Fig. 5. Median E_{LED} after n seconds over for 10 different random seeds. Shaded area shows range from minimum to maximum E_{LED} value at the given step over the runs.

VII. Simulation Results

The simulation results for E_{POS} demonstrate that all protocols are able to keep $E_{POS} = 0$ for at least 30 seconds of simulation time (see Figure 4). After 60 seconds, both of the comparison protocols have accumulated small positional errors in some of the simulation runs, and by 120 seconds both show an error in the minimum case. The AirTight implementation with optimal routing maintains E_{POS} until after 180s. We note that by 170 seconds the circle radius has exceeded 4m in all simulation runs, meaning the input assumptions to the fault model (maximum distance of 3m) have been significantly exceeded. Thus, the errors that show in later time steps are as a result of "incorrect" input data rather than a violation of the protocols timing guarantees. The AirTight implementation with randomised routing shows a very small but non-zero maximum error at 90 seconds and larger errors from 120 seconds.

For the lower criticality flow, both comparison protocols show an initial LED error after approximately 30 seconds, with the first instance of a median error greater than zero after 60 and 100 seconds for the "point to point" and "broadcast" protocols respectively. With optimised routing the AirTight protocol first shows an LED error after 50s, with a non-zero median first occurring after 90s. With randomised routing the first LED error occurs almost immediately[2], and regular errors start after 40s.

The AirTight protocol results in both higher median and maximum LED error values, particularly in later simulation steps. The protocol has allowed the system designer to prioritise motion control messages in the presence of errors, resulting in lower positional error at the cost of less critical tasks.

VIII. Flocking Application

The circle problem clearly demonstrates the advantages that a mixed criticality system model can bring to unreliable shared-medium wireless communications in a swarm robotics context. Rather than causing unpredictable application-level faults, the mixed criticality model can be used to maintain service for longer in tasks of most importance. To illustrate this in a more realistic application, we present a flocking system in which a group of 10 nodes need to communicate to form and move as a flock. Each node is assumed to know its own position and orientation through some localisation system. Every five seconds, node should transmit their location and orientation to all other nodes such that each can compute it is own velocity to maintain its position in the flock [20].

We assume that each node is also performing some environmental sensing task, that generates up to three data frames per second that should be sent to an a priori designated sink node. We again assume that mobility control is the more important task, and so define this as a low priority flow in the AirTight implementation. We use the same AirTight fault model configuration as for the circle problem, but here define point to point flows from each node to all other nodes for communication. In the broadcast implementation the maximum number of retransmissions is reduced to 3, as this is here the maximum value while ensuring the available bandwidth is not exceeded. The configuration of point-to-point implementation is unchanged.

At each simulation time step, we sum the velocity vectors of all nodes, and compute the mean length of this vector over each simulation run of 30s. Since these vectors add destructively if nodes are moving in different directions (i.e. not as a flock), this value serves as proxy for the stability of flock. Since the distances between nodes once a flock has been formed remain relatively small, we further scale down the packet delivery rate of the fault model presented in section V by a constant factor. The results present in figure 6 show that AirTight is able to maintain an almost optimal flock velocity of approximately 6 cm/s at lower PDR scaling values than the two baseline comparison protocols. We note that this advantage is, as in the circle problem, a result of prioritising the positional information. As shown in figure 7, the performance of the data collection task under AirTight decreases rapidly when packet transmission is made less reliable.

[2]We note that with a starting radius of 1m, the maximum distance between robots does not exceed the input to fault model. This early error is the other 1% from the requested 99% confidence bound provided to the low criticality fault model.

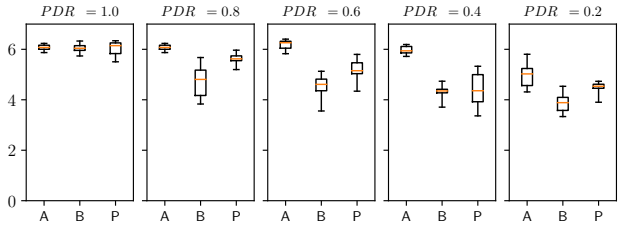


Fig. 6. Mean length of summed node velocity vectors at each time step over 10 simulation runs with different random seeds. PDR is modified by a constant factor from 1.0 to 0.2.

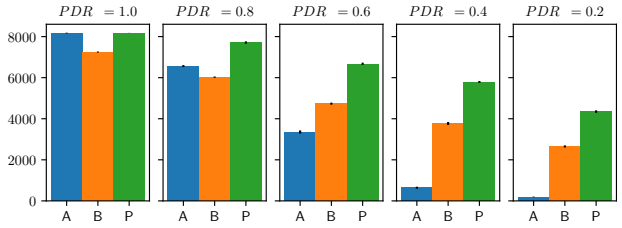


Fig. 7. Mean number of data samples received by the designated sink node over 10 simulation runs (error bars show min/max value). Note that broadcast protocol results in a lower number of received data samples even under good network conditions, as a lack of timing guarantees on delivery mean some frames are not yet delivered by the end of the simulation.

IX. Limitations and Future Work

In order to get the most benefit from the AirTight protocol [11], it requires *a priori* knowledge of the network topology and slot tables in order to determine packet routing and perform schedulability analysis of the flows. This requirement significantly impacts the ability of AirTight to be applied in many swarm robotics applications. The illustrated application in this paper presents a best case scenario since the network topology remains constant. The degraded performance observed when the node positions are randomised demonstrates the impact of the network topology not matching the prior assumptions, which would also occur if robots were permitted to move in ways that changes the network topology. In order to be able to apply the AirTight protocol to more general swarm robotics applications, the protocol must be extended such that routing can be updated at runtime. We intend to address this extension in future work.

X. Conclusion

In this paper we have demonstrated the value of using mixed criticality timing-aware wireless protocols in swarm robotics applications. Wireless communications will always be subject to error, but protocols such as AirTight allow the system designer to trade off errors intelligently, according to the relative importance of the various tasks in the system. Furthermore, such protocols allow timing behaviour to be analysed ahead of time, which allows guarantees to be made as to the timing correctness of a swarm robotics application within parameters provided by a fault model. In future work we intend to extended existing real-time protocols like AirTight in ways that allow it to specifically target swarm robotics applications.

References

[1] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo, "Swarm robotics: a review from the swarm engineering perspective," *Swarm Intelligence*, vol. 7, no. 1, pp. 1–41, 2013.

[2] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, nov 2017.

[3] C. Xia, X. Jin, L. Kong, and P. Zeng, "Bounding the demand of mixed-criticality industrial wireless sensor networks," *IEEE Access*, vol. 5, pp. 7505–7516, 2017.

[4] D. Johnson, N. Ntlatlapa, and C. Aichele, "Simple pragmatic approach to mesh routing using batman," in *2nd IFIP International Symposium on Wireless Communications and Information Technology in Developing Countries*, 2008.

[5] D. Tardioli and J. L. Villarroel, "Real time communications over 802.11: Rt-wmp," in *2007 IEEE international conference on mobile adhoc and sensor systems*, pp. 1–11, IEEE, 2007.

[6] D. Tardioli, "A wireless communication protocol for distributed robotics applications," in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 253–260, 2014.

[7] I. E. Commission *et al.*, "Industrial networks—wireless communication network and communication profiles—wirelesshart (iec 62591: 2016)," *IEC: Geneva, Switzerland*, vol. 3, pp. 1–1043, 2016.

[8] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 73–84, 2011.

[9] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, (New York, NY, USA), p. 1–14, Association for Computing Machinery, 2012.

[10] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive real-time communication for wireless cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, feb 2017.

[11] A. Burns, J. Harbin, L. Indrusiak, I. Bate, R. Davis, and D. Griffin, "Airtight: A resilient wireless communication protocol for mixed-criticality systems," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 65–75, 2018.

[12] J. Harbin, A. Burns, R. I. Davis, L. S. Indrusiak, I. Bate, and D. Griffin, "The airtight protocol for mixed criticality wireless cps," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, dec 2019.

[13] C. Pinciroli, A. Lee-Brown, and G. Beltrame, "A tuple space for data sharing in robot swarms," *EAI Endorsed Transactions on Collaborative Computing*, vol. 2, 5 2016.

[14] M. Selden, J. Zhou, F. Campos, N. Lambert, D. Drew, and K. S. J. Pister, "Botnet: A simulator for studying the effects of accurate communication models on multi-agent and swarm control," in *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 101–109, 2021.

[15] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella, and M. Dorigo, "ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm Intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[16] A. Cerpa, N. Busek, and D. Estrin, "Scale: A tool for simple connectivity assessment in lossy environments," tech. rep., September 5 2003.

[17] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, SenSys '03, (New York, NY, USA), p. 1–13, Association for Computing Machinery, 2003.

[18] N. Baccour, A. Koubâa, M. Ben Jamâa, D. do Rosário, H. Youssef, M. Alves, and L. B. Becker, "RadiaLE: A framework for designing and assessing link quality estimators in wireless sensor networks," *Ad Hoc Networks*, vol. 9, no. 7, pp. 1165–1185, 2011.

[19] K. Brun-Laguna, A. L. Diedrichs, D. Dujovne, R. Léone, X. Vilajosana, and T. Watteyne, "(not so) intuitive results from a smart agriculture low-power wireless mesh deployment," in *Proceedings of the Eleventh ACM Workshop on Challenged Networks*, CHANTS '16, (New York, NY, USA), p. 25–30, Association for Computing Machinery, 2016.

[20] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *SIGGRAPH Comput. Graph.*, vol. 21, p. 25–34, aug 1987.

Precise Scheduling Mixed-Criticality Gang Tasks with Reserved Processors

Tianning She
Department of Computer Science
Texas State University
San Marcos, TX, USA
t_s374@txstate.edu

Zhishan Guo
Department of Computer Science
North Carolina State University
Raleigh, NC, USA
zguo32@ncsu.edu

Kecheng Yang
Department of Computer Science
Texas State University
San Marcos, TX, USA
yangk@txstate.edu

***Abstract*—To mitigate the analytic pessimism that is often necessary to provide the worst-case guarantees for real-time systems, mixed-criticality (MC) scheduling has been proposed, where a task parameter may be associated with multiple estimates corresponding to multiple system runtime modes. While a large body of work on MC scheduling is directed at dropping or degrading low-critical tasks at the mode switch, a recent model, called precise MC scheduling, aims at preserving the full execution of all tasks instead. In precise MC scheduling, the additional workload due to high-critical tasks at the mode switch should be dealt with by increasing the capability of the processing platform. In this paper, we investigate the problem of precise MC scheduling of gang tasks, which may require simultaneously occupying multiple processors to commence any execution. In particular, we focus on the global earliest-deadline-first with virtual deadlines (GEDF-VD) scheduling. We derive a sufficient schedulability test for precise scheduling MC gang tasks by GEDF-VD. By the schedulability test, we also present an analysis of how many processors can be safely reserved in a given system.**

***Index Terms*—mixed-criticality tasks, precise scheduling, gang tasks, virtual deadlines.**

I. Introduction

In order to provide the worst-case guarantees for real-time systems, task parameters, such as the execution time, for schedulability analysis are usually estimated with significant pessimism, which results in less efficient runtime performance. Mixed-criticality (MC) scheduling has been proposed to mitigate such pessimism, where the worst-case execution time (WCET) of a task may be associated with multiple estimates corresponding to multiple system runtime modes.

In particular, the research attention in MC scheduling has often focused on the two-mode setting, where a mode switch is triggered by a high-critical task overrunning its smaller estimate. In the majority of existing work on MC scheduling, once a mode switch is triggered by such one or more high-critical tasks, the low-critical tasks are (fully or partially) sacrificed in order to accommodate the additional workload by those high-critical tasks. More recently, a new direction, called precise MC scheduling, has been proposed and investigated [8]. In the precise MC scheduling paradigm, after a mode switch, low-critical tasks continue to execute up to their WCET estimates as they do before the mode switch. In the meanwhile, the additional workload by the high-critical tasks are supposed to be covered by increasing the executing platform capacity, *e.g.*, boosting the speed of the processor [8].

Besides processor speed, increasing the number of available processors on a multiprocessor platform is another natural way for increasing capacity. In other words, in the normal mode, a certain number of processors are reserved and only the remaining number of processors are deemed available for the real-time tasks of our interest. Upon a mode switch, these reserved processors become available and dedicated to the real-time tasks so that the executing platform capacity is boosted from the real-time tasks' point of view. Such reserved processors in the normal mode could serve for several different purposes, such as to enter a sleep state for power and energy benefits, to be devoted to non-real-time tasks for spatial separation between real-time and non-real-time tasks (the latter will be dropped or deprioritized upon a mode switch), *etc.*, depending on the scenarios and needs of the system.

In this precise MC scheduling scheme with reserved processors, it is the number of processors that varies. Compared to traditional sequential tasks, *parallel* tasks are naturally more tuned to this particular system parameter. In this paper, we focus on one particular kind of parallel tasks, namely the (rigid) gang task model, where a task may *require* simultaneously occupying multiple processor to commence any execution. The real-time scheduling of gang tasks have received some attention [24, 16, 17, 14]. It has also been studied in the context of MC scheduling [9] but only in the conventional sense (*i.e.*, dropping tasks upon a mode switch).

Contributions. In this paper, we present the first study on the precise MC scheduling of gang tasks. With a certain number of reserved processors that are hidden from the real-time tasks in the normal mode but are able to fully dedicate to real-time tasks upon a mode switch, we formally define a specific scheduling problem in this direction. We then devise a virtual-deadline based scheduling algorithm to address this problem, and present a schedulability test for this algorithm. By the schedulability test, we also present an analysis of how many processors can be safely reserved in a given system.

Organization. In the rest of this paper, we introduce our system model and problem statement (Sec. II), present a common deadline-based scheduling algorithm for general gang tasks and its variant by virtual deadlines for MC gang tasks

(Sec. III), provide a proof and analysis for our schedulability test (Sec. IV), briefly survey related work (Sec. V), and conclude (Sec. VI).

II. System Model and Problem Statement

We consider the sporadic gang task model, which differs from the conventional sporadic task model by allowing a single task to simultaneously occupy multiple processors for execution. Furthermore, the allowed parallelism can be categorized as follows [17]: a job (invocation of a task) is

- *rigid* if the number of processors assigned to this job is specified externally to the scheduler a priori, and does not change throughout its execution;
- *moldable* if the number of processors assigned to this job is determined by the scheduler, and does not change throughout its execution
- *malleable* if the number of processors assigned to this job can be changed by the scheduler during the job's execution.

In this paper, we focus on the *rigid* parallelism model.

Specifically, we consider a system consisting of n implicit-deadline sporadic MC gang tasks $\mathcal{T} = \{\tau_1, \tau_2, \cdots, \tau_n\}$, where each task τ_i has a rigid parallelism of m_i processors. Each task τ_i is invoked recurrently with a minimum separation of T_i time units. Each invocation is called a *job* of τ_i, and we use $\tau_{i,j}$ to denote the j^{th} job of τ_i. T_i is called the *period* of τ_i, and we restrict our attention to implicit deadlines. In other words, T_i is also the *relative deadline* for each task τ_i, and every job of τ_i has an absolute deadline T_i time units after its release. The WCET of each task τ_i is estimated a two criticality levels: a low-criticality (L-) estimate C_i^L and a high-criticality (H-) estimate C_i^H, where it is assumed that $\forall i, 0 < C_i^L \leq C_i^H \leq T_i$. Furthermore, if $C_i^L = C_i^H$ for task τ_i so that τ_i *cannot* trigger a *mode switch* as to be described next, then we say τ_i is a LO-task; by contrast, if $C_i^L < C_i^H$ for task τ_i so that τ_i *could* trigger a *mode switch* as to be described next, then we say τ_i is a HI-task. We denote the set of LO-tasks (HI-tasks, respectively) by $\mathcal{T}_{\text{LO}}$ ($\mathcal{T}_{\text{HI}}$, respectively). We also refer to a job of a LO-task (HI-task, respectively) as LO-job (HI-job, respectively) for short. In summary, an implicit-deadline MC sporadic gang task τ_i is specified by a 4-tuple (C_i^L, C_i^H, m_i, T_i).

Reserving processors and mode switch. We consider a multiprocessor platform consisting of M^H identical processors, each of which has a normalized speed 1.0. In the runtime, if the L-estimates are respected, *i.e.*, *all* jobs are finished within their L-WCETs, then we say the system is in *L-mode*; if the L-estimates are exceeded, *i.e.*, *some* jobs need to execute beyond their L-WCETs and up to their H-WCETs, then we say the system is in *H-mode*. Note that the H-estimates are assumed to be always respected. In other words, any job that has cumulatively executed for its H-WCET, *i.e.*, C_i^H, yet still not completed, is considered as *erroneous* and would be terminated immediately. That is, only HI-tasks, for which $C_i^L < C_i^H$, could trigger a mode switch. The system begins with L-mode and the amount of execution completed for each job is being monitored during runtime. If any job has cumulatively executed for its L-WCET, *i.e.*, C_i^L, but still requires further execution, then the system is immediately notified and switched to H-mode. The system can recover to L-mode once *all* processors become idle. We require that only M^L, where $M^L < M^H$, processors are used to actively execute tasks in $\mathcal{T}$ in L-mode, while the remaining $M^\delta = (M^H - M^L)$ processors are reserved. Nonetheless, once the system is switched to H-mode, all M^H processors are devoted to execute tasks in $\mathcal{T}$.

Note that, in contrast to the majority of existing works on MC scheduling, in this work *no task is entirely or partially dropped* upon a mode switch, and every job must meet its absolute deadline in any system mode. The difference between the two WCET estimates upon mode switch, *i.e.*, $C_i^H - C_i^L$, is compensated by the additional M^δ active processors.

In this paper, we assume that the preemption and migration overheads, *e.g.*, due to memory interference, are negligible. Or, equivalently, we assume these overheads are pessimistically taken into account in the WCET estimates.

Moreover, we denote the utilization of a task τ_i in L- and H-modes, respectively, by

$$u_i^L = \frac{C_i^L \cdot m_i}{T_i} \quad \text{and} \quad u_i^H = \frac{C_i^H \cdot m_i}{T_i}.$$

Since $C_i^L = C_i^H$ holds for every LO-task, it also holds $u_i^L = u_i^H$ for such task. We further denote the total utilization of the set of LO-tasks and the set of HI-tasks in L- and H-modes, respectively, by

$$U_{\text{LO}} = \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u_i^L = \sum_{\tau_i \in \mathcal{T}_{\text{LO}}} u_i^H,$$

$$U_{\text{HI}}^L = \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u_i^L, \text{ and } U_{\text{HI}}^H = \sum_{\tau_i \in \mathcal{T}_{\text{HI}}} u_i^H.$$

We further denote the total utilization of all tasks in L- and H-modes, respectively, by

$$U^L = \sum_i u_i^L = U_{\text{LO}} + U_{\text{HI}}^L \quad \text{and} \quad U^H = \sum_i u_i^H = U_{\text{LO}} + U_{\text{HI}}^H.$$

Problem Statement. We address the problem of scheduling the MC rigid Gang tasks on M^H unit-speed processors to meet all deadlines in *all* scenarios with the potential of reserving M^δ processors, where $M^\delta = M^H - M^L > 0$. We say the system is *precise-MC schedulable* if all deadlines are guaranteed to be met and the following constraints are respected.

- Tasks in τ only execute on M^L processors if *all* jobs finish within C_i^L time units of execution;
- Tasks in τ may execute on all the M^H processors if a *any* job (of a HI-task) executes for more than C_i^L time units (yet finishes within C_i^H time units of execution).

III. Scheduling MC Gang Tasks

In this section, we propose an earliest-deadline-first (EDF) based scheduling algorithm to address the precise MC scheduling problem of gang tasks, which has been formalized in the prior section.

Algorithm 1: Selecting Jobs to Schedule under GEDF

input : Ready(t), which is the ready job set at time t
output: Sched(t), which is the scheduled job set at time t

Sched$(t) \leftarrow \emptyset$
for each $\tau_{i,j} \in$ Ready(t) in deadline increasing order **do**
  if $\sum_{\tau_{k,\ell} \in \text{Sched}(t)} m_k \leq M - m_i$ **then**
    Sched$(t) \leftarrow$ Sched$(t) \cup \{\tau_{i,j}\}$
  end
end

A. GEDF *Scheduling for Gang Tasks*

We consider the *preemptive* global EDF (GEDF) scheduling algorithm for *ordinary (non-MC)* gang tasks as follows[1]:

Definition 1. Under GEDF scheduling, the priority of each job is determined by its deadline — the earlier the deadline, the higher the priority. We also assume deadline ties are broken arbitrarily but consistently, and therefore there is no *priority* ties while *deadline* ties may exist. Letting Ready(t) denote the set of *ready* jobs at an arbitrary time instant t, the set of jobs Sched(t) being scheduled at time t is determined by Algorithm 1. m_k is the degree of parallelism of τ_k, which has a job in Sched(t). Please note that, in practice, Algorithm 1 does not need to be evaluated at every time instant but only needs to be invoked when a job is completed and when a new job is released.

Please note that the GEDF scheduling cannot be defined by simply considering the summation of parallelism of all ready jobs with higher priorities, but rather it needs to go through Algorithm 1. This is because some higher-priority (but not the highest) ready jobs may not be scheduled due to lack of available processors while some lower-priority ones with less parallelism may actually be scheduled.

Furthermore, an important parameter Δ_i for a gang task τ_i under GEDF scheduling was introduced in [14] and is defined as follows.

Definition 2. Let Δ_i denote the maximum possible number of idle processors at any time during τ_i's non-executing intervals in which τ_i has pending jobs but does not execute. In other words, if τ_i is not being executing but has pending jobs, the number of idle processors is at most Δ_i.

Clearly, $(m_i - 1)$ would be a *safe* upperbound on Δ_i. Nonetheless, a dynamic programming algorithm has been introduced in [14] to calculate Δ_i in a more accurate manner. This Δ_i identification algorithm runs in polynomial time with a time complexity of $\mathcal{O}(M^2 \cdot n)$, where M is the total number of available processors and n is the number of gang tasks.

With pre-calculating the Δ_i parameter for every task prior to runtime for any given (non-MC) gang task system, a sufficient schedulability test has been proven in [14] as follows.

[1]This formal description of GEDF for gang tasks was introduced in [15].

Theorem 1 (Theorem 2 in [14])**.** *A (non-MC) gang task system, where each task is specified by (C_i, m_i, T_i), is schedulable on M identical processors by* GEDF*, if*

$$\forall i, U_{\text{SUM}} \leq (M - \Delta_i)(1 - \frac{u_i}{m_i}) + u_i \tag{1}$$

where

$$u_i = \frac{C_i}{T_i} \quad \textit{and} \quad U_{\text{SUM}} = \sum_i u_i.$$

B. *Algorithm* GEDF-VD *for MC Gang Tasks*

With the prior work on GEDF scheduling of non-MC gang tasks as discussed above, we introduce a virtual-deadline-based scheduling algorithm, GEDF-VD, to address the precise MC scheduling problem considered in this paper.

Under GEDF-VD, a system wide constant x is selected such that $0 < x < 1$. Each task is associated to a *relative virtual deadline* of $x \cdot T_i$, in addition to the *relative actual deadline* T_i. That is, every job of task τ_i has a virtual deadline at $x \cdot T_i$ time units after its arrival time and has an actual deadline at T_i time units after its arrival. Note that, because under precise MC scheduling LO-tasks are not dropped in the H-mode, the virtual-deadline setting does also apply to LO-tasks as well as HI-tasks. Then, in the runtime, GEDF is applied to schedule all tasks by taking virtual deadlines as deadlines in the L-mode and taking actual deadlines as deadlines in the H-mode, respectively.

Recall that for non-MC gang task set, we have a Δ_i parameter that can be obtained before runtime. The Δ_i identification algorithm from [14] needs to take the required parallelism m_i for every task and the total number of available processors M as inputs. In the problem considered in this paper, although m_i remains the same in the L- and H-modes for every task τ_i, the total number of available processors does differ in the two modes, being M^L and M^H in L- and H-modes, respectively. Therefore, we need to apply the Δ_i identification algorithm twice, and then for each task τ_i, we obtain Δ_i^L and Δ_i^H for L- and H-modes, respectively.

Having Δ_i^L and Δ_i^H for every task, we claim the following sufficient schedulability test for GEDF-VD and this test is to be proven next in Sec. IV.A.

A set of precise MC gang tasks with implicit deadlines are schedulable by GEDF-VD using at most M^L processors in the L-mode and using at most M^H processors in the H-mode, if

$$K^L + K^H \leq 1,$$

where

$$K^L = \max_i \left\{ \frac{m_i U^L + (M^L - \Delta_i^L - m_i) u_i^L}{m_i \cdot (M^L - \Delta_i^L)} \right\}$$

$$K^H = \max_i \left\{ \frac{m_i U^H + (M^H - \Delta_i^H - m_i) u_i^H}{m_i \cdot (M^H - \Delta_i^H)} \right\},$$

so that scaling factor x for setting virtual deadlines in GEDF-VD can be (arbitrarily) chosen from the range

$$[K^L, 1 - K^H].$$

IV. Precise-MC Schedulability Analysis

In this section, we first prove the correctness of the schedulability presented in the prior section, with given M^L and M^H (and therefore given Δ_i^L and Δ_i^H). Then, we provide a method to obtain a safe lower bound on M^L that guarantees the schedulability for given task system and given M^H.

A. Schedulability Test

To derive the schedulability test, we consider the schedulability in L- and H-modes, respectively, in the following two lemmas. We first derive a sufficient schedulability condition for L-mode (Lem. 1). Then, assuming the schedulability in L-mode, we derive a sufficient schedulability condition for H-mode (Lem. 2). They together result in Thm. 2.

Lemma 1. *Under* GEDF-VD *scheduling, all (*LO- *and* HI-*) tasks must meet their virtual deadlines in L-mode, if*

$$\forall i, x \geq \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i \cdot (M^L - \Delta_i^L)}.$$

Proof. In L-mode, tasks are scheduled by their virtual deadlines. By treating the relative virtual deadline as both the relative deadline and the period, every task τ_i in L-mode can be viewed as a (sporadic, implicit-deadline) non-MC gang task specified as (C_i^L, m_i, xT_i). Under this view, the utilization of each task τ_i is

$$\frac{C_i^L}{xT_i} = \frac{u_i^L}{x},$$

and the total utilization is

$$\sum_i \frac{C_i^L}{xT_i} = \frac{1}{x} \cdot \sum_i \frac{C_i^L}{T_i} = \frac{U^L}{x}.$$

Therefore, by Thm. 1, these non-MC gang tasks must meet their deadlines under GEDF, *i.e.*, original MC-gang tasks must meet their virtual deadlines in L-mode, if

$$\forall i, \frac{U^L}{x} \leq (M^L - \Delta_i^L)(1 - \frac{\frac{u_i^L}{x}}{m_i}) + \frac{u_i^L}{x} \tag{2}$$

Given the facts that $0 < x < 1$, $m_i > 0$, and $\Delta_i^L < M^L$,

$$\begin{aligned}
(2) \Leftrightarrow & \forall i, \frac{U^L}{x} \leq M^L - \Delta_i^L - (\frac{M^L - \Delta_i^L}{m_i} - 1) \cdot \frac{u_i^L}{x} \\
\Leftrightarrow & \forall i, \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i x} \leq M^L - \Delta_i^L \\
\Leftrightarrow & \forall i, x \geq \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i \cdot (M^L - \Delta_i^L)}.
\end{aligned}$$

Thus, the lemma follows. □

Lemma 2. *Under* GEDF-VD *scheduling, assuming all virtual deadlines are met in L-mode, all (*LO- *and* HI-*) tasks must meet their actual deadlines in H-mode, if*

$$\forall i, x \leq 1 - \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i \cdot (M^H - \Delta_i^H)}.$$

Proof. By assuming all virtual deadlines are met in L-mode, it follows that the first job that reaches its virtual deadline but has not completed must trigger a mode switch. As a result, any job of τ_i that is released in L-mode has not completed by the mode switch must have its actual deadline (as well as next job release of τ_i) at least $(1-x)T_i$ time units after the mode-switch time instant. In the meanwhile, every job of τ_i cannot execute for more than C_i^H time units in any circumstance and all subsequent releases in H-mode must separate by $T_i > (1-x)T_i$ time units, according to the task model. So, by treating any unfinished job at the mode switch as a new job release at the mode-switch time instant, every task τ_i in L-mode can be viewed as a (sporadic, implicit-deadline) non-MC gang task specified as $(C_i^H, m_i, (1-x)T_i)$. Under this view, the utilization of each task τ_i is

$$\frac{C_i^H}{(1-x)T_i} = \frac{u_i^H}{1-x},$$

and the total utilization is

$$\sum_i \frac{C_i^H}{(1-x)T_i} = \frac{1}{1-x} \cdot \sum_i \frac{C_i^H}{T_i} = \frac{U^H}{1-x}.$$

Therefore, by Thm. 1, these non-MC gang tasks must meet their deadlines under GEDF, *i.e.*, original MC-gang tasks must meet their actual deadlines in H-mode, if

$$\forall i, \frac{U^H}{1-x} \leq (M^L - \Delta_i^L)(1 - \frac{\frac{u_i^H}{1-x}}{m_i}) + \frac{u_i^H}{1-x} \tag{3}$$

Given the facts that $0 < x < 1$, $m_i > 0$, and $\Delta_i^H < M^H$,

$$\begin{aligned}
(3) \Leftrightarrow & \forall i, \frac{U^H}{1-x} \leq M^H - \Delta_i^H - (\frac{M^H - \Delta_i^H}{m_i} - 1) \cdot \frac{u_i^H}{1-x} \\
\Leftrightarrow & \forall i, \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i(1-x)} \leq M^H - \Delta_i^H \\
\Leftrightarrow & \forall i, 1 - x \geq \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i \cdot (M^H - \Delta_i^H)} \\
\Leftrightarrow & \forall i, x \leq 1 - \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i \cdot (M^H - \Delta_i^H)}.
\end{aligned}$$

Thus, the lemma follows. □

Theorem 2. *An MC gang task system is precise-MC schedulable under* GEDF-VD *on M^H processors with $M^\delta = (M^H - M^L)$ processors reserved in L-mode, if*

$$K^L + K^H \leq 1,$$

where

$$K^L = \max_i \left\{ \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i \cdot (M^L - \Delta_i^L)} \right\}$$

$$K^H = \max_i \left\{ \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i \cdot (M^H - \Delta_i^H)} \right\}.$$

Proof. $K^L + K^H \leq 1$ implies that $[K^L, 1 - K^H]$ is not an empty set. Also, it is evident that both $K^L > 0$ and $K^H >$

0, so $[K^L, 1-K^H] \subset (0,1)$. Therefore, a valid x can be (arbitrarily) selected from $[K^L, 1-K^H]$. Note that

$$x \geq K^L \Rightarrow \forall i, x \geq \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i \cdot (M^L - \Delta_i^L)} \quad \text{and}$$

$$x \leq 1-K^H \Rightarrow \forall i, x \leq 1 - \frac{m_i U^H + (M^H - \Delta_i^H - m_i)u_i^H}{m_i \cdot (M^H - \Delta_i^H)}.$$

By Lem. 1 and Lem. 2, the theorem follows. □

B. A Safe Lower Bound on M^L for Schedulability

We have already established a schedulability test for a given task system with given M^L and given M^H. Nonetheless, the value of M^L may not always be fixed in a prior but its proper setting may be a question for the system designer, where the system workload specification ($\mathcal{T}$) and the platform maximum capacity (M^H) are provided.

Under such a setting, K^H can still be obtained *a priori*, as $\mathcal{T}$ and M^H (and therefore, Δ_i^H) are given. Nonetheless, to evaluate the schedulability condition, *i.e.*, $K^L \leq 1 - K^H$, it is not only M^L that varies— each Δ_i^L could also be different under different M^L, and it is not a closed-form representation by M^L. One way to find an M^L that guarantees schedulability is to enumerate all possible values of M^L and to apply the schedulability test for each case. Alternatively, the following theorem directly provides a safe lower bound on M^L that guarantees the schedulability for certain systems.

Theorem 3. *For any MC gang task system where*

$$\forall i, (1-K^H)m_i > u_i, \tag{4}$$

it must be precise-MC schedulable under GEDF-VD*, if*

$$M^L \geq \max_i \left\{ \frac{m_i(U^L - u_i^L)}{(1-K^H)m_i - u_i^L} + m_i - 1 \right\}. \tag{5}$$

Proof. Our goal is to show that the lower bound on M^L specified in this theorem is sufficient to imply $K^L + K^H \leq 1$, which is the schedulability test.

From (5), it directly follows that

$$\begin{aligned} &\forall i, M^L \geq \frac{m_i(U^L - u_i^L)}{(1-K^H)m_i - u_i^L} + m_i - 1 \\ \Rightarrow &\forall i, M^L - (m_i - 1) \geq \frac{m_i(U^L - u_i^L)}{(1-K^H)m_i - u_i^L} \end{aligned} \tag{6}$$

While every Δ_i^L may vary as M^L varies, we note that $\forall i, \Delta_i^L \leq m_i - 1$ holds by definition in any case. Therefore, regardless the specific value of M^L, we always have

$$\begin{aligned} (6) &\Rightarrow \forall i, M^L - \Delta_i \geq \frac{m_i(U^L - u_i)}{(1-K^H)m_i - u_i^L} \\ &\overset{\text{by (4)}}{\Rightarrow} \forall i, (1-K^H)m_i \geq \frac{m_i(U^L - u_i^L)}{M^L - \Delta_i} + u_i^L \\ &\Rightarrow \forall i, (1-K^H) \geq \frac{m_i U^L + (M^L - \Delta_i^L - m_i)u_i^L}{m_i \cdot (M^L - \Delta_i^L)} \\ &\Rightarrow (1-K^H) \geq K^L, \end{aligned}$$

where the last step is by the definition of K^L and directly implies $K^L + K^H \leq 1$. The theorem follows. □

V. Related Work

Since it was introduced by Vestal [34], MC tasks and their scheduling have attracted a huge amount of interest in the real-time systems research community. (Please see [12] for a comprehensive survey on this topic.) Initially, most works were directed to scenarios where all low-critical tasks are completely dropped if any high-critical task behaves its worst case. More recently, this over-sacrificing was criticized, and gradual degradation of low-critical tasks was investigated. To provide degraded service, the imprecise MC model [11] was proposed, where the execution of low-critical tasks is reduced but not dropped even in the worst case. Several subsequent works [3, 11, 20, 22, 23, 27] explored various definitions of this execution reduction. To eliminate such reduction, the problem of precise MC scheduling was proposed and investigated on varying-speed uniprocessors [8, 35] and multiprocessors [33]. Such varying-speed processors are equipped with a capacity of *dynamic voltage and frequency scaling* (DVFS) for the purpose of energy efficiency [21]. However, DVFS is not effective in reducing static/leakage power consumption. Compared to DVFS, *dynamic power management* (DPM) and deep sleep modes can lead to significant energy conservation resulted from the power-down of a number of system components [5, 4]. [32] proposed precise MC scheduling of sequential tasks on multiprocessors that a part of processors in the system can be turned into sleep modes in typical scenarios while fully exploited under worst-case scenarios.

Besides sequential tasks, the problem of scheduling real-time parallel tasks has been investigated. Several works focus on parallel task models that are related to the gang task model, including periodic multi-thread task models [29, 31, 13], the synchronous task model [1], and DAG task models [10, 25, 19, 7, 18]. In these models, Parallel threads of a task can be independently considered and scheduled. By contrast, they must simultaneously occupy a set of processors to execute under the gang task model [17, 14, 24]. Goossens et al. considered the rigid gang task model in [16], and Berten et al. proposed the moldable gang scheduling in [6].

Upon MC scheduling, few works have been proposed for parallel task models. Liu et al. [28] proposed the MC scheduling of synchronous task model, while the MC scheduling for DAG task models is investigated in [2] and [26]. Rambo et al. [30] proposed a replica-aware co-scheduling approach for mixed-critical systems. For the MC gang task scheduling, [9] presented an approach combining Global Earliest Deadline First (GEDF) and Earliest Deadline First with Virtual Deadline (EDF-VD). Unlike these works, we consider precise MC scheduling of the gang task model that provides full service for low-critical gang tasks.

VI. Conclusion

In many conventional MC scheduling problems, LO-tasks may be dropped or degraded when a mode switch to H-mode is triggered. In this work, we investigated the precise MC scheduling, where LO-tasks preserve their execution estimates in H-mode. Upon a mode switch to H-mode, a certain number

of processors that are reserved in L-mode now become fully available and dedicated to the real-time tasks. Our focus in this paper was on rigid gang tasks, each of which may require multiple processors to commence any execution. We presented our scheduling algorithm EDF-VD for the problem of precise scheduling MC gang tasks and provided a schedulability test for EDF-VD as well as a proof for its correctness. In addition, we also analyzed a safe lower bound on the number of non-reserved processors in L-mode for guaranteeing the schedulability of a given system. This, from the other hand, implies how many processors can be safely reserved in L-mode without jeopardizing the schedulability.

ACKNOWLEDGMENT

This work is supported in part by NSF grants CCF-2028481, CNS-2104181, a start-up grant from the North Carolina State University, and start-up and REP grants from Texas State University.

REFERENCES

[1] Björn Andersson and Dionisio de Niz. Analyzing global-EDF for multiprocessor scheduling of parallel tasks. In *International Conference On Principles Of Distributed Systems*, pages 16–30. Springer, 2012.

[2] Sanjoy Baruah. The federated scheduling of systems of mixed-criticality sporadic dag tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 227–236. IEEE, 2016.

[3] Sanjoy Baruah, Alan Burns, and Zhishan Guo. Scheduling mixed-criticality systems to guarantee some service under all non-erroneous behaviors. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 131–138. IEEE, 2016.

[4] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

[5] Luca Benini, Alessandro Bogliolo, Giuseppe A Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.

[6] Vandy Berten, Pierre Courbin, and Joël Goossens. Gang fixed priority scheduling of periodic moldable real-time tasks. In *5th junior researcher workshop on real-time computing*, pages 9–12. Citeseer, 2011.

[7] Ashikahmed Bhuiyan, Zhishan Guo, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient real-time scheduling of DAG tasks. *ACM Transactions on Embedded Computing Systems*, 17(5):84, 2018.

[8] Ashikahmed Bhuiyan, Sai Sruti, Zhishan Guo, and Kecheng Yang. Precise scheduling of mixed-criticality tasks by varying processor speed. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, pages 123–132, 2019.

[9] Ashikahmed Bhuiyan, Kecheng Yang, Samsil Arefin, Abusayeed Saifullah, Nan Guan, and Zhishan Guo. Mixed-criticality multicore scheduling of real-time gang task systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 469–480. IEEE, 2019.

[10] Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Sebastian Stiller, and Andreas Wiese. Feasibility analysis in the sporadic DAG task model. In *2013 25th Euromicro conference on real-time systems*, pages 225–233. IEEE, 2013.

[11] Alan Burns and Sanjoy Baruah. Towards a more practical model for mixed criticality systems. In *1st WMC*, 2013.

[12] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Dept. of Computer Science, University of York, Tech. Rep*, pages 1–81, 2019.

[13] Pierre Courbin, Irina Lupu, and Joël Goossens. Scheduling of hard real-time multi-phase multi-thread (mpmt) periodic tasks. *Real-time systems*, 49(2):239–266, 2013.

[14] Zheng Dong and Cong Liu. Analysis techniques for supporting hard real-time sporadic gang task systems. *Real-Time Systems*, 55(3):641–666, 2019.

[15] Zheng Dong, Kecheng Yang, Nathan Fisher, and Cong Liu. Tardiness bounds for sporadic gang tasks under preemptive global edf scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 32(12):2867–2879, 2021.

[16] Joël Goossens and Vandy Berten. Gang ftp scheduling of periodic and parallel rigid real-time tasks. *arXiv preprint arXiv:1006.2617*, 2010.

[17] Joël Goossens and Pascal Richard. Optimal scheduling of periodic gang tasks. *Leibniz transactions on embedded systems*, 3(1):04–1, 2016.

[18] Zhishan Guo, Ashikahmed Bhuiyan, Di Liu, Aamir Khan, Abusayeed Saifullah, and Nan Guan. Energy-efficient real-time scheduling of DAGs on clustered multi-core platforms. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 156–168. IEEE, 2019.

[19] Zhishan Guo, Ashikahmed Bhuiyan, Abusayeed Saifullah, Nan Guan, and Haoyi Xiong. Energy-efficient multi-core scheduling for real-time DAG tasks. In *29th Euromicro conference on real-time systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[20] Zhishan Guo, Kecheng Yang, Sudharsan Vaidhun, Samsil Arefin, Sajal K Das, and Haoyi Xiong. Uniprocessor mixed-criticality scheduling with graceful degradation by completion rate. In *2018 IEEE Real-Time Systems Symposium (RTSS)*, pages 373–383. IEEE, 2018.

[21] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Energy efficient DVFS scheduling for mixed-criticality systems. In *Proceedings of the 14th International Conference on Embedded Software, ACM*, page 11. ACM, 2014.

[22] Pengcheng Huang, Pratyush Kumar, Georgia Giannopoulou, and Lothar Thiele. Run and be safe: Mixed-criticality scheduling with temporary processor speedup. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1329–1334. IEEE, 2015.

[23] Mathieu Jan, Lilia Zaourar, and Maurice Pitel. Maximizing the execution rate of low criticality tasks in mixed criticality system. *1st WMC*, 2013.

[24] Shinpei Kato and Yutaka Ishikawa. Gang EDF scheduling of parallel task systems. In *2009 30th IEEE Real-Time Systems Symposium*, pages 459–468. IEEE, 2009.

[25] Jing Li, Jian-Jia Chen, Kunal Agrawal, Chenyang Lu, Christopher D Gill, and Abusayeed Saifullah. Analysis of federated and global scheduling for parallel real-time tasks. In *ECRTS*, volume 14, pages 85–96, 2014.

[26] Jing Li, David Ferry, Shaurya Ahuja, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Mixed-criticality federated scheduling for parallel real-time tasks. *Real-time systems*, 53(5):760–811, 2017.

[27] Di Liu, Jelena Spasic, Nan Guan, Gang Chen, Songran Liu, Todor Stefanov, and Wang Yi. Edf-vd scheduling of mixed-criticality systems with degraded quality guarantees. In *2016 IEEE Real-Time Systems Symposium (RTSS)*, pages 35–46. IEEE, 2016.

[28] Guangdong Liu, Ying Lu, Shige Wang, and Zonghua Gu. Partitioned multiprocessor scheduling of mixed-criticality parallel jobs. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10. IEEE, 2014.

[29] Geoffrey Nelissen, Vandy Berten, Joël Goossens, and Dragomir Milojevic. Techniques optimizing the number of processors to schedule multi-threaded tasks. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 321–330. IEEE, 2012.

[30] Eberle A Rambo and Rolf Ernst. Replica-aware co-scheduling for mixed-criticality. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[31] Abusayeed Saifullah, Jing Li, Kunal Agrawal, Chenyang Lu, and Christopher Gill. Multi-core real-time scheduling for generalized parallel task models. *Real-Time Systems*, 49(4):404–435, 2013.

[32] Tianning She, Zhishan Guo, Qijun Gu, and Kecheng Yang. Reserving processors by precise scheduling of mixed-criticality tasks. In *2021 IEEE 27th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 103–108. IEEE, 2021.

[33] Tianning She, Sudharsan Vaidhun, Qijun Gu, Sajal K Das, Zhishan Guo, and Kecheng Yang. Precise scheduling of mixed-criticality tasks on varying-speed multiprocessors. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, 2021.

[34] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS)*, pages 239–243. IEEE, 2007.

[35] Kecheng Yang, Ashikahmed Bhuiyan, and Zhishan Guo. F2VD: Fluid rates to virtual deadlines for precise mixed-criticality scheduling on a varying-speed processor. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

A High-Resilience Imprecise Computing Architecture for Mixed-Criticality Systems

Zhe Jiang[†§], Xiaotian Dai[†*], Alan Burns[†], Neil Audsley[¶], Zonghua Gu[‖], Ian Gray[†]
[†]University of York, United Kingdom, [§]University of Cambridge, United Kingdom
[¶]City, University of London, United Kingdom, [‖]Umeå University, Sweden

I. Introduction

Mixed-Criticality Systems (MCS) are systems in which components can be developed under multiple assurance or criticality levels and integrated on a shared hardware platform. This is common in, for example, the automotive industry in which an Advanced Driver Assistance System (ADAS) may involve many modules developed under various criticality levels. The collision avoidance module will be high-criticality, whereas route-planning may be developed to a lower criticality. In this domain, criticality levels are defined as Automotive Safety and Integrity Levels (ASILs). Designing a modern complex embedded system in domains such as this is challenging because systems usually have to satisfy weight, power and cost (SWaP-C) requirements [2], producing a trend towards the integration of myriad software components and a huge driver towards mixed-criticality systems.

The most accepted approach is the widely studied dual-criticality MCS model (Vestal's model [3]). In this model, tasks are high-criticality and low-criticality (HI- and LO-tasks), and have estimated Worst-Case Execution Times (WCET)s with different confidence levels according to their criticality. The high-critical WCET (HI-WCET) is obtained using strict or formal methods, *e.g.*, static timing analysis, resulting in high confidence but often significant pessimism. On the other hand, low-criticality WCET (LO-WCET) may be obtained with measurement-based methods [3], which are more representative, but much less confident. To ensure timing correctness, it is necessary that: (i) if all tasks finish executing within their LO-WCETs, then they will all finish executing by their deadlines; (ii) if any task does not complete execution within its LO-WCET, then HI-tasks at least should complete execution by their deadlines [3].

Conventional dual-mode MCSs therefore define two *system modes*, LO-mode and HI-mode. In LO-mode (the optimistic mode) the system assumes that the execution time of every task (LO-task or HI-task) does not exceed its LO-WCET. If this assumption is violated, the system switches into HI-mode, in which it assumes the execution time of HI-tasks may exceed their LO-WCETs, but will not exceed their HI-WCETs [2].

Imprecise MCS. The key difficulty associated with conventional dual-mode MCS is the termination of the LO-tasks affecting system functionality [2], especially if the system still relies on the computation results of LO-tasks even in HI-mode. To extend the "life cycle" of the LO-tasks, a more practical model is possible: without dropping LO-tasks directly during mode switch, the system can continue to execute the LO-tasks as much as possible but with degraded computation precision, effectively accelerating LO-tasks' execution. Such a system model is called Imprecise MCS (IMCS) [4], [5]. In an IMCS, the imprecise computation tends to be used on Floating Point (FP) computations because: (i) Modern LO-tasks heavily rely on FP calculations, *e.g.*, image processing [6], signal processing [7], and Deep Neural Networks (DNNs) [6]; (ii) FP calculations consume significantly more clock cycles compared to other operations, which dominate the tasks' computation time [8].

This paper presents a new IMCS framework, *HIART*-MCS, which, mitigates the computation errors caused by imprecise computation for LO-tasks; achieves near-conventional MCS real-time performance for HI-tasks; and supports dynamically adjustable fine-grained levels of approximation of individual tasks at run-time without the need for software adaptations. Specifically, we present:

- A new FPU design, supporting approximation, accelerating *all* types of floating point computation.
- A novel system architecture that enables run-time configuration of software tasks' approximation degrees.
- A practical measurement-based method for exploring all potential configurations of the tasks in the system.
- A theoretical model, schedulability analysis, and optimisation approach, ensuring both the system's real-time performance and computation correctness.
- Comprehensive experiments examining the system under different configurations using various metrics.

II. *HIART*-FPU Overview

As task execution is affected by components at different system levels (language, compiler, OS, *etc.*), approximations can be potentially deployed at any system level. In *HIART*-MCS, we implement the approximation at the hardware level because of three reasons: (i) hardware-level approximation ensures *source compatibility* of software tasks, as the approximation is transparent to the software; (ii) hardware-level approximation occurs at run-time and so does not require input data to be known prior; (iii) hardware-level approximation can provide finer bit-width granularity, allowing run-time reconfiguration of the degree of approximation for each task.

[*]A full version of this paper was published in the IEEE Transactions on Computers (TC) [1].

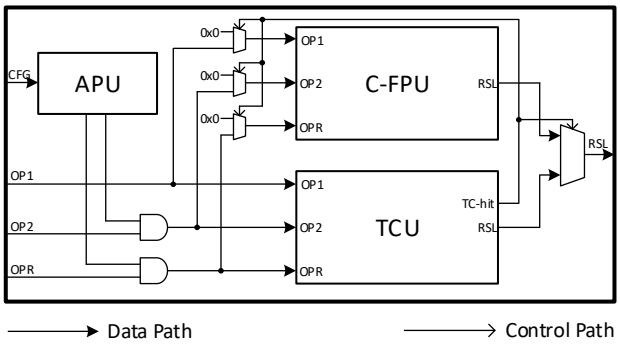


Fig. 1. Overview of *HIART*-FPU (CFG: configure interface).

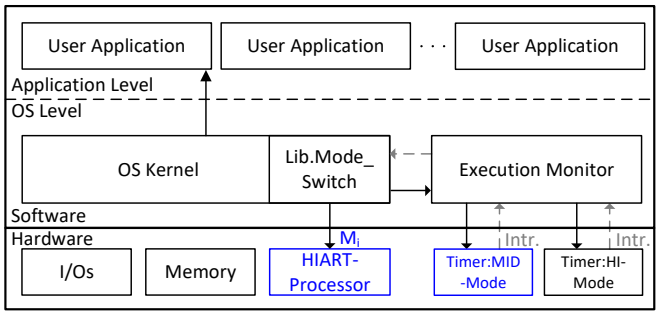


Fig. 2. System Architecture of *HIART*-MCS (The blue boxes show the new parts in *HIART*-MCS).

Top-level micro-architecture. The design has two features:

- **Run-time configurability**: The FPU can be configured at run-time to execute transparently with different approximation schemes. This is particularly important for IMCS.
- **FP-cache:** the FPU contains a dedicated FP-cache, recording the recent calculation results of the mantissa bits. During an FP-cache-hit, the FPU returns the mantissa bits' calculation in a single clock cycle, accelerating the FP computations. The benefits of this cache will be explained in the following sections.

We designed the C-FPU using three parts (see Fig 1): an Approximation Processing Unit (APU) that controls the approximation degrees of the operands; a Cached-FPU (C-FPU) and a Trivial Calculation Unit (TCU).

Computation Procedures. During an FP calculation, two operands (filtered by the APU) and one operator are first transmitted to the TCU. The TCU checks whether the calculation matches a Trivial Cases (TCs): if TC hits, the TCU directly returns the calculation results. Otherwise, the TCU generates a TC-miss signal to the C-FPU, requesting the C-FPU to proceed with the corresponding FP calculation.

III. System Architecture of HIART-MCS

Unlike the conventional dual-mode MCS framework, *HIART*-MCS uses three system modes:

- LO**-mode**: *HIART*-MCS initialises in LO-mode and stays in this mode if none of the HI-tasks exceeds LO-WCETs.
- MID**-mode**: when a HI-task (τ_i) overruns its LO-WCET (denoted as C_i^{LO}), at the moment of over-execution, the *HIART*-MCS immediately switches to MID-mode. In this mode, LO-tasks can still be executed with approximation, in which case their execution times (denoted as C_i^{AP}) should be less than C_i^{LO}.
- HI**-mode**: if the HI-task continues to overrun and exceeds a specific time point, *HIART*-MCS immediately switches to HI-mode while all LO-tasks are terminated. The time point triggering mode switch from MID-mode to HI-mode is termed MID-WCET (denoted as C_i^{MI}).

To enable these three system modes, we introduce architectural changes at both the hardware and the software levels. At the hardware level, we deploy our novel processor that supports FP imprecision (detailed previously in Sec. II) of the LO-task in the MID-mode. Also, we deploy two hardware timers to monitor the HI-tasks execution times for LO-WCETs and MID-WCETs.

The software-level structure is comprised of kernel and user spaces. In the kernel space, we present an implementation of both the *Lib.mode_switch* and the execution monitor. Specifically, we propose a new control function in the *Lib.mode_switch*, managing the approximation degree of LO-tasks in MID-mode. For the execution monitor, we operate the hardware timers during context switches corresponding to the new mode switch strategy. In user space, we need no changes to the original OS interfaces (see Fig. 2), thereby ensuring *source compatibility* and allowing tasks designed for a conventional MCS framework to be directly migrated.

Operation. At system initialisation, LO-tasks' approximation degree ($\mathcal{M}_i$) and HI-tasks' LO-WCETs and MID-WCETs are loaded. During context switches, the execution monitor suspends the timers of the currently executing task and then (re-)activates the timers for the next executing task to monitor the over-execution of LO-WCET and MID-WCET.

If a HI-task exceeds its LO/MID-WCET, the corresponding timer generates an interrupt to trigger a mode switch (to MID/HI-mode) by invoking *Lib.mode_switch*. Also, the system configures the approximation degree of LO-tasks at context switch if the system stays in MID-mode. The pseudo-code of the mode switch and context switch of *HIART*-MCS can be found in [8].

IV. Timing Analysis and Optimisation

To give an analytical bound for *HIART*-MCS, we introduce response time analysis (RTA) for a *HIART*-enabled platform in order to test the schedulability of given a taskset with constrained deadlines. With the introduction of the MID-mode, the model and analysis have to be tailored to reflect the new design. Specifically, the analysis for *HIART* uses a standard dual-criticality analysis, with an additional criticality level, and with the MID-mode execution times of LO-criticality tasks being *smaller* than that of LO-mode. Based on the schedulability analysis, the system's overall utility can then be optimised by applying either a fixed, or varying level of computational imprecision. This is discussed in detail in [1].

V. Experimental Evaluation

Platform setup. We built 4/8/16-core *HIART*-MCS variants on a Xilinx VC709 evaluation board. *HIART*|nc denotes the system without any FP-cache (which is the system framework presented in [8]), whereas *HIART*|N-way ($N \in \{2, 8\}$) contains an N-way FP-cache with 256 entries. The FP-cache address length is set to be 16, in which case only operands with fewer than 8 valid mantissa bits ($\mathcal{M}_i < 16$) will be buffered. This is because when $\mathcal{M}_i < 16$, the FP-cache hit rate decreases dramatically, thus the improvement from a cache-hit will be modest given the resources it would otherwise cost. We implemented the *HIART*-processors based on the SiFive Freedom E31, an open-source 32-bit RISC-V processor (5-stage pipeline). We implemented the proposed FPU (see Sec. II) in Verilog. The processors are connected to memory and I/O peripherals using a 5×5 mesh open-source NoC [9]. The hardware was synthesised and implemented by Xilinx Vivado (v2020.2). The software executing on the processors (OS kernels, drivers and user applications) was compiled using a RISC-V GNU toolchain. FreeRTOS (v.10.4) was the OS kernel for all processors, with the minimal modifications that were described in Sec. III.

Existing MCS frameworks usually implement run-time monitoring and mode switches either in the OS kernel (OSK) or a dedicated hypervisor (HYP). Therefore, we built two baseline systems (BS) on similar hardware platforms using conventional RISC-V processors. BS|OSK is a baseline MCS framework implementing execution monitoring and mode switches in OS kernels. BS|HYP is a baseline MCS framework using virtualisation technology, including real-time patches and I/O enhancement [10]. BS|HYP implements an execution monitor and mode switches in its hypervisor.All systems ran at 100 MHz because of the use of an FPGA as a prototyping.

A. Theoretical Evaluation

As one of the major design goals, the *HIART* should increase survivability of LO-tasks. To evaluate this, we used simulation based on synthetic tasksets using analytical evaluation. We defined an evaluation metric, survivability ($\mathcal{S}$), which is the percentage of LO-tasks that survive during an overload:

$$\mathcal{S} = \frac{\#\_of\_survived\_cases}{\#\_of\_all\_cases} \times 100\% \quad (1)$$

The task utilisation was generated with UUniFast, and half of the taskset was selected to be HI-tasks. In each trial, one of the HI-tasks was randomly selected to overrun and trigger a mode switch, with overrun execution time uniformly distributed from C_i^{LO} to C_i^{HI}. Task priority was assigned by deadline-monotonic priority ordering. We note that for the traditional dual-criticality model (without AP-mode), the survivability was 0% as all LO-tasks are dropped. Each utilisation consisted of 50 trials, and optimal $\mathcal{M}_i$ were selected.

Obs. 1. The proposed MCS model with the addition of the AP-mode significantly improved LO-tasks' survivability.

The observation can be seen in Fig. 3. The survivability of LO-tasks was significantly improved when utilisation was

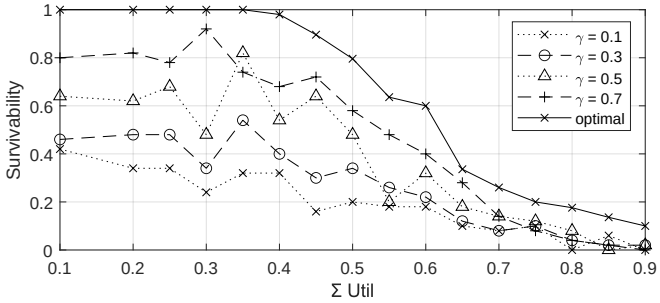


Fig. 3. Survivability of LO-tasks vs total utilisation

delrelatively low. For example, the survivability was around 100% when $\sum U_i \leq 0.35$ and was 60% when $\sum U_i = 0.6$. Survivability gradually decreased as utilisation increased, with survivability of 20% even when $\sum U_i \leq 0.8$. Eventually, survivability became close to 0, where nearly no system can maintain its schedulability. In general, a higher γ could prolong the AP-mode duration and thus improve survivability.

B. Real-time Performance and Computation Quality

As described in Sec. I, real-time performance and computation quality are the most important metrics for the IMCSs. In this section, we use real-world use cases to evaluate the real-time performance and computation quality of the systems.

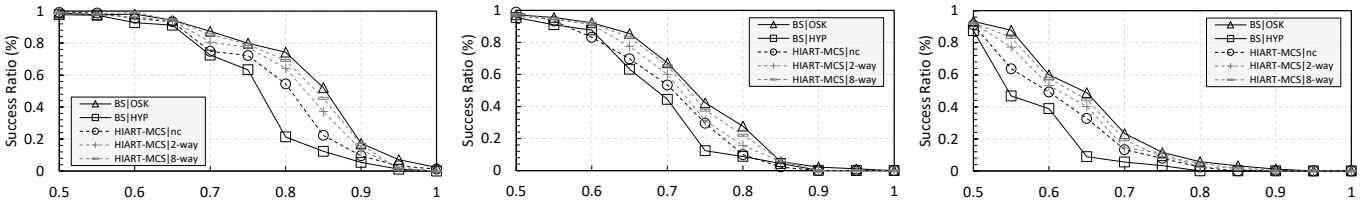
Hardware and software deployment. We configured the examined systems with 4/8/16 processors and deployed two sets of software tasks:

- 18 HI-tasks, selected from Renesas functional safety automotive use case database (*e.g.*, CRC and RSA32).
- 18 LO-tasks, including 6 DNN tasks, 6 image processing tasks, and 6 automotive function tasks selected from EEMBC benchmark.

In the LO-task set, the DNN and image processing tasks can be approximated at the MID-mode. The DNN tasks were classified into two categories, established upon LeNet-5 [11] and SqueezeNet (a variant of AlexNet) architectures. In each category, three tasks were respectively trained using MNIST, EMNIST and CIFAR-10 training datasets. The image processing tasks were implemented to perform Sobel filter, Canny filter, Scharr filter, Prewitt filter, Roberts filter and Sharpen filter on Oxford-IIIT Pet Dataset [12], respectively.

Experimental setup. In the experiments, the raw data for processing by the tasks was generated off-chip and sent to the evaluated systems via two Ethernet controllers (1 Gbps) at run-time. The experimental results were stored in the dedicated addresses of DRAMs. For each experimental setup, we executed the examined systems 200 times (500 seconds for each run) under varying target utilisation from 45% to 100%, with intervals of 5% increments.

Metrics. We examine the systems under each target utilisation using two metrics: (i) real-time performance; (ii) computation quality. For real-time performance, we used Success Ratio (SR) to report the percentage of the trials executed without any deadline miss of HI-tasks. For computation quality, we



(a) 4-core systems. (b) 8-core systems. (c) 16-core systems.

Fig. 4. HI-tasks: success ratio (x-axis: utilisation).

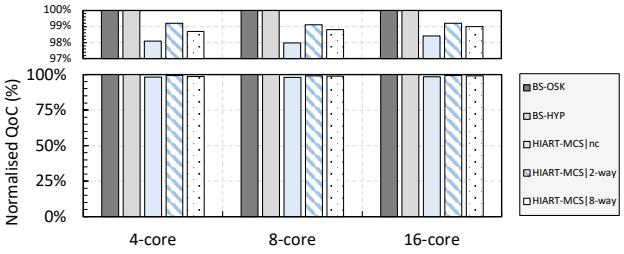


Fig. 5. System: average QoC.

examined the systems using system-level QoC to record the percentage of correct execution.

Obs. 2. IMCS decreased the HI-tasks' success ratio compared to the conventional MCS. This issue was effectively mitigated by *HIART*|2-way and *HIART*|8-way.

This observation is given by Fig. 4. When the systems were configured with the same settings (*i.e.*, core number and utilisation), *HIART*|nc suffered from a reduction of success ratio compared to the conventional MCS (BS|OSK), as the *HIART*|nc is designed upon BS|OSK but executes more tasks in a specific time period (*i.e.*, MID-mode). Unlike *HIART*|nc, *HIART*|2-way and *HIART*|8-way achieved similar success ratios as the BS|OSK. This is benefited by the deployment of the FP-cache (Sec. II), accelerating the tasks' executions at MID-mode.

Obs. 3. Although using imprecise computation in IMCS decreases system QoC, *HIART*|N-way could mitigate them.

As reported by Fig. 5, deploying imprecise computation in *HIART*|nc decreased the system-level QoC. Although such loss was caused by the nature of imprecise computing, introduction of FP cache accelerates the computation, and such brings the opportunity for finding more suitable parameters, including deferring switching points and with better approximation degrees of each LO-tasks.

VI. Conclusion

Imprecise Mixed Criticality (IMCS) has been recently studied as a more practical model than conventional dual-mode mixed criticality, because it could effectively improve the survivability of low-criticality tasks. The IMCS model raises three challenges to be solved. How to minimise the number of system errors caused by computational imprecision, how to minimise pessimism when compared with standard dual-mode mixed criticality, and how to adapt existing legacy systems and code to use the IMCS model. To solve these problems, we present a new processor design which implements transparent hardware-level computational approximation. The amount of approximation is run-time configurable and can be adjusted on a per-task basis to accelerate floating point computation. The cost of this approach is additional hardware area and power use. We propose a novel IMCS framework that takes advantage of this hardware support with an expanded system model, schedulability analysis, and optimisation approach that ensures both the system's real-time performance and minimises the impact of computational imprecision. As shown in our evaluations, the proposed system significantly extends the LO-tasks' survivability whilst ensuring that HI-tasks' real-time performance remains near to that of a conventional MCS.

For more details regarding the paper, please see [1].

References

[1] Z. Jiang, X. Dai, A. Burns, N. Audsley, Z. Gu, and I. Gray, "A high-resilience imprecise computing architecture for mixed-criticality systems," *IEEE Transactions on Computers*, 2022.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007.

[4] L. Huang, I.-H. Hou, S. S. Sapatnekar, and J. Hu, "Graceful degradation of low-criticality tasks in multiprocessor dual-criticality systems," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 159–169.

[5] X. Gu and A. Easwaran, "Dynamic budget management and budget reclamation for mixed-criticality systems," *Real-Time Systems*, 2019.

[6] S. Bateni and C. Liu, "ApNet: Approximation-aware real-time neural network," in *2018 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2018, pp. 67–79.

[7] M. Rosenthal, A. Weiss, and A. Mazloumian, "Realtime signal processing on embedded GPUs," in *Embedded Computing Conference (ECC2018), Winterthur, 5. Juni 2018*, 2018.

[8] Z. Jiang, X. Dai, and N. Audsley, "Hiart-mcs: High resilience and approximated computing architecture for imprecise mixed-criticality systems," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 290–303.

[9] G. Plumbridge, J. Whitham, and N. Audsley, "Blueshell: a platform for rapid prototyping of multiprocessor nocs and accelerators," *ACM SIGARCH Computer Architecture News*, 2014.

[10] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in xen," in *2011 Proceedings of the Ninth ACM International Conference on Embedded Software*. IEEE, 2011.

[11] Y. LeCun *et al.*, "LeNet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[12] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar, "Cats and dogs," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3498–3505.

WaP: Computing the Execution Probability of Jobs with Replication in Mixed-Criticality Schedules

Antonin Novak
Faculty of Electrical Engineering &
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Prague, Czech Republic
antonin.novak@cvut.cz

Zdenek Hanzalek
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Prague, Czech Republic

Premysl Sucha
Czech Institute of Informatics, Robotics, and Cybernetics
Czech Technical University in Prague
Prague, Czech Republic

***Abstract*—This extended abstract represents the journal paper published in [13]. In that paper, we study the computation of the execution probability of jobs with uncertain execution times in a static mixed-criticality schedule. In contrast to the majority of research in mixed-criticality systems that work with task models where the jobs are gradually revealed to the scheduler, we assume a time-triggered environment where the offline scheduler generates a static schedule [14], [15], [17]. The execution time of the mixed-criticality jobs is not known in advance and is revealed during the online execution. An online execution policy is designed to handle the prolongations of execution times and escalations of the system mode. The policy may eventually reject some of the low-criticality jobs under some execution scenarios, thus affecting the execution probability of the jobs.**

This paper deals with the complexity and the method for analysis of the execution probability of mixed-criticality jobs in a static schedule. To overcome the rigidity of static scheduling, we introduce job replication, i.e., scheduling multiple time slots for a single job, as a new mechanism for increasing the execution probability of jobs. We show that the general problem with job replication becomes as hard as the counting variant of 3-SAT problem. To compute the execution probability, we propose an algorithm utilizing the framework of Bayesian networks. The proposed methodology demonstrates an interesting connection between schedules with uncertain execution and probabilistic graphical models.

***Index Terms*—execution analysis, mixed-criticality, time-triggered, job replication, Bayesian networks, computational complexity**

I. Introduction

In contrast to the majority of research in mixed-criticality systems [3], [4], [20], we assume a time-triggered environment where the offline scheduler generates a static schedule [1], [14], [15], [17], [18]. Nevertheless, the basic idea is somewhat similar to classical Vestal's model [20] with the key differences described below. We assume that the execution time of the mixed-criticality jobs is not known and follows a probability distribution [13], [17]. The actual execution time is revealed during the online execution of the schedule. To compensate for the prolongations of execution times and the elevations of the system mode observed during the runtime (e.g., in Figure 1 at time $t = 5$, system mode is raised to $e_t = 2$), we employ an online policy to govern the execution of the schedule. The policy may eventually reject some of the low-criticality jobs under some execution scenarios, thus affecting their execution probability.

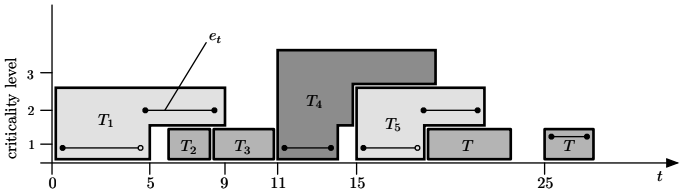
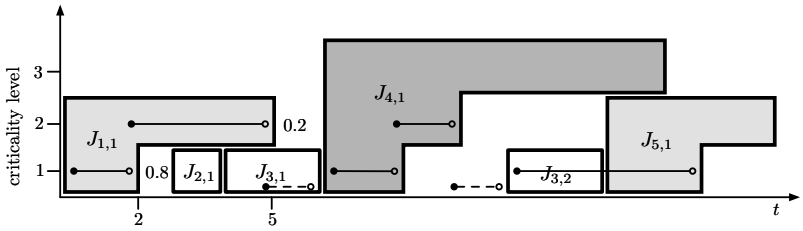


Fig. 1: Schedule with mixed-criticality jobs and an execution scenario e_t.

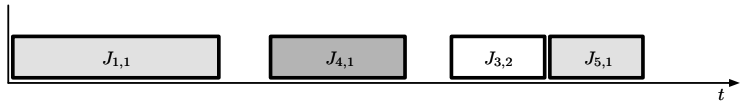
In our mixed-criticality model [8], [13]–[15], we also use the so-called *match-up* property, meaning that the system mode can be lowered and returned to nominal operation mode once it is escalated. This can be seen, e.g., in Figure 1, where after execution of job T_5, the execution scenario matches up with the nominal system mode, i.e., $e_t = 1$ at $t = 25$. This is, in fact, similar to the concepts used in Flexible Mixed-Criticality Systems [6], [11]. In this extended abstract, we will focus on a more intuitive explanation with an application example rather than giving formal definitions first. A more detailed description of the assumed mixed-criticality system model for non-preemptive scheduling in time-triggered environments can be found in the series of works [8], [13]–[15], [17]. For experimental evaluation of the model on a real hardware testbed, please see [9].

Although the static scheduling increases the predictability of

This work was supported by the EU and the Ministry of Industry and Trade of the Czech Republic under the Project OP PIK CZ.01.1.02/0.0/0.0/20_321/0024399. Furthermore, this work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS22/167/OHK3/3T/13.



(a) Mixed-criticality schedule with replication with five jobs where J_3 has two replicas.



(b) An execution scenario.

Fig. 2: Mixed-criticality schedule with three criticality levels with one of the possible execution scenarios.

the system (i.e., its behavior is given by a static schedule which can be analyzed offline), the execution of critical jobs may occasionally require to reject less critical jobs during online execution (e.g., $J_{2,1}$ and $J_{3,1}$ in Figure 2b). Thus, careful scheduling of jobs needs to be used to mitigate the degradation of the execution probability of non-critical jobs without affecting the requirements of critical jobs. To optimize the execution probability of jobs in a schedule, a scheduling algorithm needs to assess the quality (i.e., the objective function) of the current schedule in order to drive the search toward a good solution. Hence, the computation of the objective function of a schedule is the central component of any algorithm that produces high-quality schedules.

A. Contributions

In paper [13], we study job replication, which is a mechanism for increasing the execution probability of jobs in time-triggered mixed-criticality schedules. Specifically, the main contributions are:

- We introduce the concept of replication to mixed-criticality schedules as a mechanism for increasing the execution probability of jobs.
- We show that the general problem of computing execution probability for a job in a mixed-criticality schedule with replication as hard as counting the number of satisfiable assignments of a propositional formula in normal conjunction form, in contrast to the known polynomial-time algorithm for the case without replication.
- We solve the problem by a reduction to the probabilistic inference in a suitably defined Bayesian network.
- We show that the cases of reasonable interests (i.e., constant-bounded number of criticality levels and the maximum number of replicas per job) can be solved in polynomial time in the number of jobs, which enables practical usage of job replication.

II. Static Mixed-Criticality Systems with job replication

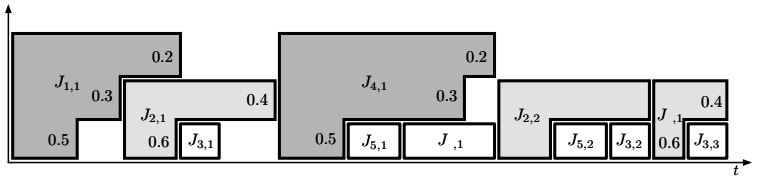
In this section, we describe an application example to illustrate the main concepts of static mixed-criticality systems with job replication. An extension of our mixed-criticality model by a frequency dimension has been implemented and tested in practice on a real-life testbed, e.g., for message scheduling in 5G NR (new radio) networks [9].

A. Application example

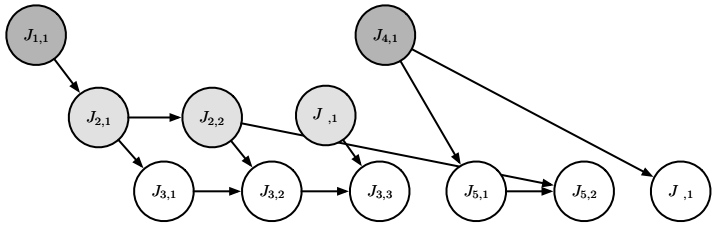
Consider a message scheduling problem on a shared communication bus in modern vehicles. Safety-related standards such as ASIL (Automotive Safety Integrity Levels) [2] introduce the existence of messages with several levels of criticality, such as:

- messages of high criticality (criticality 3) are used for safety-related functionalities (their failure may result in death or severe injury to people), such as steering;
- messages of medium criticality (criticality 2) are used for mission-related functionalities (their failure may prevent activity from being successfully completed), such as parking assist;
- messages of low criticality (criticality 1) are typically used for infotainment functionalities, such as automotive navigation system.

The messages are transmitted via the bus at the moments defined by the static time-triggered schedule [10] (e.g., the static segment of a FlexRay bus [7]), which improves determinism and predictability. The goal is to compute an objective function reflecting statistical properties of a given static schedule that accounts for disruption of the communication according to the message criticality. In real-life environments, the execution of jobs is affected by various sources of uncertainty, causing, e.g., transmission delays. In the above example, criticality expresses



(a) Example schedule $\boldsymbol{s}$.



(b) Corresponding Bayesian network $G(\boldsymbol{s})$.

Fig. 3: Representation of a schedule by a Bayesian network.

TABLE I: Execution probabilities of individual replicas in schedule $\boldsymbol{s}$.

	$J_{1,1}$	$J_{2,1}$	$J_{2,2}$	$J_{3,1}$	$J_{3,2}$	$J_{3,3}$	$J_{4,1}$	$J_{5,1}$	$J_{5,2}$	$J_{6,1}$	$J_{7,1}$
$\Pr\{J_{i,q} \succeq 1\}$	1.0	0.8	0.2	0.68	0.32	0.0	1.0	0.5	0.46	0.5	1.0

the commitment to the transmission when the original transmission is prolonged. Therefore, several transmission attempts are awarded to messages with a high criticality, whereas for low-criticality messages, it might be just a single one.

Figure 2a shows an example of a mixed-criticality static schedule with six job replicas. Each job has a given integer criticality, as it is seen on the vertical axis. For example, job $J_{4,1}$ has a criticality of three, job $J_{1,1}$ has a criticality of two, while $J_{2,1}$ has a criticality of one. Notice that $J_{3,1}$ and $J_{3,2}$ are two replicas of the same job; hence, they have the same parameters but different start times. Job $J_{1,1}$ has execution time 2 time units with probability 0.8, and 5 time units with probability 0.2. The considered values of execution times are derived from the (empirical) cumulative distribution function with respect to the selected probability thresholds [15], [20].

Mixed-criticality schedules contain several alternative execution scenarios, with the one being selected based on the realized execution times of jobs that occur during the runtime execution. It is dispatched in such a way that in any of these scenarios, all highly critical jobs are executed, rejecting jobs with lower criticality only when a highly critical one is prolonged. This leads to more efficient resource usage since the low-criticality jobs may use the resource when the critical ones are not prolonged. To compensate for unexpected prolongations of critical jobs observed at the runtime, some of the less critical ones might not be executed under the specific realization of execution times. This can be seen in Figure 2b, where jobs $J_{2,1}$ and $J_{3,1}$ are rejected if realized execution time of $J_{1,1}$ is equal to 5, happening with probability 0.2. However, the second replica $J_{3,2}$ was executed later on. Finally, we note that, e.g., $J_{1,1}$ is never rejected since it does not share its execution time with any other job with higher criticality. Therefore, the execution probability of $J_{1,1}$, denoted as P_1, is 1 while execution probability of $J_{2,1}$ is $P_2 = 0.8$.

In this paper, we deal with the problem of computing execution probabilities P_i of jobs J_i with replication for the given fixed schedule.

III. Results

A. Time complexity of the problem

We show that the general problem where either the maximum number of criticality levels $\mathcal{L}$ or the maximum number of replicas per job $\mathcal{R}$ is bounded by a polynomial in the number of jobs and the other is equal to some chosen constant remains $\#\mathcal{P}$-hard. We remind that $\#\mathcal{P}$ is a class of *counting problems*, i.e., a set of problems that count the number of accepting paths in a polynomial-time non-deterministic Turing machine [19]. An example of a problem contained in $\#\mathcal{P}$ is the following: What is the number of spanning trees in the given connected simple graph? A problem is said to be $\#\mathcal{P}$-hard, if for every problem in $\#\mathcal{P}$, there exists a polynomial-time counting reduction to it [5].

First, we show that deciding whether a job has a non-zero probability of being executed is as hard as determining whether a CNF (*conjunctive normal form*) formula is satisfiable.

Proposition 1: There exists a finite number of maximum replicas per job $\mathcal{R}$ such that deciding whether $P_i > 0$ for some job J_i is $\mathcal{NP}$-complete.

The reduction suggests that the problem remains hard even for a constant number of the maximum job replicas, i.e., $\mathcal{R} = 4$. Moreover, we will show that a non-constant number of criticality levels is not the only source of hardness. Indeed, the problem remains hard, assuming a constant number of

criticality levels when the maximum number of replicas is not fixed to a constant.

Proposition 2: There exists a finite number of criticality levels $\mathcal{L}$ such that determining whether $P_i > 0$ for some job J_i is $\mathcal{NP}$-complete.

To compute the exact execution probability of jobs in a given schedule, we propose an algorithm based on the theoretical framework of the Bayesian network.

B. Algorithm for computation of the execution probability

We show how the statistical properties of mixed-criticality schedules with replication can be described with Bayesian networks. A Bayesian network $G = (V, A)$ is a probabilistic directed acyclic graphical model representing the joint distribution over the set V of random variables using conditional dependencies defined by edges A. The network contains one vertex for each job replica in the schedule. The directed edges connect the vertices if the corresponding job replica can affect the execution of the other (i.e., a conflicting higher-criticality job or preceding replica). Since the execution time of jobs is uncertain, we can view the job replicas as random variables. Then, the execution policy defines a joint probability distribution $\Pr\{J_{1,1}, \ldots, J_{n,n_i}\}$ over the given schedule that assigns a probability to each execution scenario.

To represent this distribution, we use Bayesian networks (BN) [16], which can be seen as an efficient way of representing joint distributions. An example of such a static mixed-criticality schedule and its representation by a Bayesian network can be seen in Figure 3. To compute the execution probabilities of jobs, one can use algorithms, such as variable elimination or junction trees, for the inference in Bayesian networks. The outcome of this procedure is the execution probability of each job, which can be seen in Table I. For example, the result of the analysis in Table I reveals that replica $J_{3,3}$ cannot be executed, thus, can be removed in the design phase. Please see [13] for more details.

IV. Conclusion

In this paper, we have introduced the job replication mechanism for the static time-triggered mixed-criticality model to help to overcome the rigidity of static scheduling. Job replication, i.e., scheduling a single job with multiple occurrences, increases the execution probability of jobs but introduces additional computation complexity for the analysis of the resulting static schedules. We have shown that the complexity is affected by two natural parameters—the number of criticality levels and the maximum number of replicas per job. To practically solve the problem of the computation of the execution probability, we have proposed a reduction to the theoretical framework of Bayesian networks for which many efficient algorithms exist.

For future research, we suggest looking into the integration of event-triggered and time-triggered paradigms using techniques such as schedule graph abstraction [12]. One of the possible applications might be to use static scheduling for the most critical functionality, as it often consists of core and essential components that do not change very often, but its correctness needs to be formally verifiable. However, static approaches lack the flexibility and efficiency of event-triggered environments, which could be used for applications with smaller criticality. Therefore, combining the two might bring the best of both paradigms.

References

[1] Lalatendu Behera and Purandar Bhaduri. Time-triggered scheduling for multiprocessor mixed-criticality systems. In *Distributed Computing and Internet Technology*, pages 135–151, Cham, 2018. Springer International Publishing.

[2] Ron Bell. Introduction to IEC 61508. In *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*, pages 3–12. Australian Computer Society, Inc., 2006.

[3] Alan Burns and Robert I. Davis. A survey of research into mixed criticality systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, November 2017.

[4] Alan Burns, Robert I Davis, Sanjoy Baruah, and Iain Bate. Robust mixed-criticality systems. *IEEE Transactions on Computers*, 67(10):1478–1491, 2018.

[5] Nadia Creignou and Miki Hermann. *On P completeness of some counting problems*. PhD thesis, INRIA, 1993.

[6] Xinyang Dong, Gang Chen, Mingsong Lv, Weiguang Pang, and Wang Yi. Flexible mixed-criticality scheduling with dynamic slack management. *Journal of Circuits, Systems and Computers*, 30(10):2150306, 2021.

[7] Jan Dvořák and Zdeněk Hanzálek. Using two independent channels with gateway for flexray static segment scheduling. *IEEE Transactions on Industrial Informatics*, 12(5):1887–1895, 2016.

[8] Zdenek Hanzalek, Tomas Tunys, and Premysl Sucha. An analysis of the non-preemptive mixed-criticality match-up scheduling problem. *Journal of Scheduling*, 19(5):601–607, Oct 2016.

[9] Xi Jin, Yu Tian, Chi Xu, Changqing Xia, Dong Li, and Peng Zeng. Mixed-criticality industrial data scheduling on 5G NR. *IEEE Internet of Things Journal*, 2021.

[10] H. Kopetz. *Event-triggered versus time-triggered real-time systems*, pages 86–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991.

[11] Jaewoo Lee and Jinkyu Lee. MC−FLEX: Flexible mixed-criticality real-time scheduling by task-level mode switch. *IEEE Transactions on Computers*, 71(8):1889–1902, 2022.

[12] Mitra Nasri and Bjorn B Brandenburg. An exact and sustainable analysis of non-preemptive scheduling. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, pages 12–23. IEEE, 2017.

[13] Antonín Novák and Zdenek Hanzalek. Computing the execution probability of jobs with replication in mixed-criticality schedules. *Annals of Operations Research*, 2022.

[14] Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. Efficient algorithm for jitter minimization in time-triggered periodic mixed-criticality message scheduling problem. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, page 23–31, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Antonín Novák, Premysl Sucha, and Zdenek Hanzalek. Scheduling with uncertain processing times in mixed-criticality systems. *European Journal of Operational Research*, 279(3):687–703, 2019.

[16] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[17] Yasmina Seddik and Zdenek Hanzalek. Match-up scheduling of mixed-criticality jobs: Maximizing the probability of jobs execution. *European Journal of Operational Research*, 262(1):46 – 59, 2017.

[18] Jens Theis, Gerhard Fohler, and Sanjoy Baruah. Schedule table generation for time-triggered mixed criticality systems. *Proceedings of the 1st International Workshop on Mixed Criticality Systems, RTSS*, pages 79–84, 2013.

[19] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189 – 201, 1979.

[20] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 239–243. IEEE, 2007.

Bridging the Pragmatic Gaps for Mixed-Criticality Systems in the Automotive Industry

Zhe Jiang*‡, Shuai Zhao†, Ran Wei§, Richard Parterson‡, Nan Guan††, Yan Zhuang§, Neil Audsley¶,
*University of Cambridge, United Kingdom, ‡ARM Ltd., United Kingdom,
†University of York, United Kingdom, ¶City, University of London, United Kingdom,
§Dalian University of Technology, China, ††City, University of Hong Kong, China

I. Introduction

There is an increasing trend for safety-critical systems to be integrated onto a shared platform to achieve functions of different criticalities [1] [2]. Such systems are often referred to as *Mixed-Criticality Systems (MCS)s* [3]. With the ever-increasing demand of system functionalities and the shift of the semiconductor industry to more powerful (multi-core) platforms, mixing functionalities of different criticality levels in a common hosting platform is appealing - system costs introduced by size, weight and power consumption could potentially be significantly reduced in the mixed-criticality setting [4]. For example, leading automated vehicle companies are devoting themselves to integrating Electronic Control Unit (ECU) clusters and In-Vehicle Information System (IVIS) in a shared platform. In such system, the ECU clusters must be developed with the highest criticality as they usually control the mechanical components (e.g. engine control), whereas the IVIS can be developed with relatively lower criticality as it only provides the interactive functionalities (e.g. navigation).

In academia, extensive research efforts have been made towards MCS [3]. However, in industries, limited development guidance for MCS have been provided in industrial standards (e.g. DO-178C for avionics, ISO 26262 for automotive and EN 50128 for railway). Consequently, no standard industrial system architecture for MCS has been established [5]. This is due to the fact that a pragmatic gap exists between theoretical MCS models and industrial practice – researchers have not taken industrial practice/requirements into sufficient consideration, causing conceptual mismatches to emerge between theoretical models and industrial architectures. This is observed by a number of studies, in [6], multiple definitions on *criticality* between the theoretical MCS models and industrial standards are discussed; in [7], the applicability of *graceful degradation* from theoretical MCS models to the industrial context is discussed. To the best of our knowledge, no systematic development methodology for MCS in industrial scenarios has been proposed.

Contributions. In [8], we proposed a new system architecture to take the first practical step in an attempt to bridge the gaps between theoretical MCS models and industrial practices. The original contributions made in [8] include i) a *systematic approach* to develop MCS in an industrial scenario, which is proposed upon Adaptive Mixed-Criticality (AMC) MCS model [3] with considerations of industrial practice and requirements; ii) a *system architecture* (*P-MCS*) and three *design methodologies* for the proposed approach; iii) a *theoretical model* and *schedulability analysis* for *P-MCS*, which guarantees system predictability; and iv) comprehensive *experiments* to examine *P-MCS* against conventional MCS frameworks in different perspectives.

This paper makes the following additional contributions: i) a comprehensive *analysis* of pragmatic gaps between MCS theory in academia and industrial practice; ii) a *systematic approach* to develop MCS in an industrial scenario, which is proposed upon Adaptive Mixed-Criticality (AMC) MCS model [3] with considerations of industrial practice and requirements; iii) a *theoretical model* and *schedulability analysis* for *P-MCS*, which guarantees system predictability; and iv) comprehensive *experiments* to examine *P-MCS* against conventional MCS frameworks in different perspectives.

II. The Academic Model

The majority of the academic research effort on MCS relies on the AMC model proposed by Vestal [3]. The AMC model assumes that the system has several execution modes ($L \in \{(Mode)\ A, B, C, D, ...\}$), and contains a finite set of *sporadic tasks*. Each task τ_i is defined by its period (T_i), relative deadline (D_i), a priority P_i, a criticality level ($l_i \in \{A, B, C, D, ...\}$), and a set of Worst-Case Execution Time (WCET) estimations ($\{C_{i,A}, C_{i,B}, ..., C_{i,l_i}\}$), in which these estimations reflect the WCET of τ_i in the criticality level, up to l_i. The model assumes that $C_{i,A} \leq C_{i,B} \leq ...C_{i,l_i}$. Specifically, the measured WCET at the lowest system mode (i.e. $L = A$) is set to $C_{i,A}$, whereas at each higher system mode, the subsequent estimations ($C_{i,B}, ..., C_{i,l_i}$) are obtained by either more pessimistic WCET analysis techniques, or by considering safety margins imposed by certification authorities. The system initialises from system mode A, and all tasks are scheduled to execute. During execution, if any task τ_i exceeds its execution budget ($C_{i,A}$), the system will switch to the next mode (i.e. $L = B$). In the meantime, tasks with a criticality level lower than B ($l_i < B$) are suspended.

In the first AMC work [3], only a single-core MCS with two system modes (i.e. Low- and High-criticality modes) is considered. The AMC model is further extended by various works, e.g. extensions on multiple system modes and criticality levels [9], re-activation of the dropped tasks [10], etc. (see [2] for a comprehensive survey). Our proposed *P-MCS* architecture is also built atop the AMC model, with assumptions that the system does not switch modes backwards, and tasks do not reactivate after being terminated/suspended.

*A full version of this paper was published in the IEEE IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) [1].

[1]By criticality, we mean the required safety assurance level of system components, such as the Automotive Safety Integrity Level (ASIL) from the international automotive safety standard ISO 26262.

III. The Industrial Gaps

Our work is motivated by two observations obtained from analysing the pragmatic gap between the AMC model and practice in the automotive industry.

A. The Breaking of the ASIL Allocation System

The core concept of the AMC model is *system mode switch*, which guarantees the execution of tasks with higher criticality, by suspending tasks with lower criticality [2], [3]. As summarised by Burns and Davis [2], the majority of the research links the *system mode switch* to '*graceful degradation*' from industrial standards, i.e. '*a technique aimed at maintaining the more important system functions available, despite failures, by dropping the less important system functions*' (ISO 26262-4:2018, Clause 7). Therefore, determining 'more important functions' and suspending 'less important functions' is the key of implement *system mode switch* correctly in an industrial MCS architecture.

However, ISO 26262 provides a means to determine the criticality of the requirement of a function, which considers *S* (Severity), *E* (Exposure) and *C* (Controllability). According to the philosophy of *system mode switch* of the AMC model, tasks assigned with higher *ASIL* requirements must always be allowed to execute (when tasks assigned with lower *ASIL* cannot finish execution within the time given to them), even if it means that tasks assigned with lower *ASIL* requirements will be suspended. This means, for example, when the AMC model executes at system mode C ($L = C$), which enables *ASIL-C* and *ASIL-D* tasks to execute, it will suspend all *ASIL-A* and *ASIL-B* tasks, regardless of their associated *S*, *E* and *C* classes. This is problematic, consider an *ASIL-B* task τ_i ($l_i = B$) with $\{$*S3*, *E3* and *C2*$\}$ and an *ASIL-C* task τ_j ($l_j = C$) with $\{$*S2*, *E4* and *C3*$\}$. When AMC determines that task τ_j needs to execute and perform mode switch from mode B ($L = B$) to mode C ($L = C$), task τ_i is suspended. However, what has not been considered is that the *Exposure* of task τ_i is raised from *E3* to *E4* because its intended function will always fail (because it is suspended by the mode switch). In addition, failure of task τ_i causes more harm than τ_j for it has an *S3* Severity class. As previously discussed, ASILs are statically allocated, this means that the safety requirement of τ_i cannot be raised at run-time, even though the suspension of τ_i essentially raises its safety requirement from *ASIL-B* ($l_i = B$) to *ASIL-C* ($l_i = C$).

Another problem caused by the *system mode switch* is the suspension of tasks with *ASIL* allocated to their requirements as a result of ***ASIL Decomposition***. For example, provided a safety requirement $R1$ with *ASIL-D* is decomposed into $R1.1$ (*ASIL-C*) and $R2.2$ (*ASIL-A*), where τ_i (with $l_i = C$) and τ_j (with $l_j = A$) are tasks fulfilling these two safety requirements. When the AMC model switches from Mode A ($L = A$) to Mode B ($L = B$), task τ_j is suspended as $l_j < L$. Although τ_j is independent from τ_i (ISO 26262 enforces component independence when performing ASIL Decomposition), the suspension of τ_j poses threats to the fulfilment of safety requirement $R1$. Another more severe problem is when $R1$ is decomposed into $R1.1$ (*ASIL-B*) and $R2.2$ (*ASIL-B*), that both tasks fulfilling $R1.1$ and $R1.2$ will be suspended when the AMC switches from Mode B ($L = B$) to Mode C ($L = C$). The above observations give rise to the first pragmatic gap of AMC:

Pragmatic Gap I. *During the system mode switch, safety analysis must be performed (either run-time or off-line) on the executing tasks to determine ones that need to be preserved, in order to avoid catastrophic consequences caused due to the termination of these applications.*

B. Isolation v.s. Freedom from Interference

In safety-related standards (including ISO 26262), isolation/separation between different critical functions is presented as the essential requirement for MCS. As regulated in ISO 26262, '*If freedom from interference between elements implementing safety requirements cannot be argued in the preliminary architecture, then the architectural elements shall be developed in accordance with the highest ASIL for those safety requirements.*'. This implies that without certain isolation, all elements have to be treated with the highest criticality. Although *ASIL Decomposition* can be applied, in practice, *ASIL Decomposition* is rather abused without considering isolation/separation [11].

Isolation must be sufficiently considered in these dimensions: temporal isolation, spatial isolation and fault isolation. In an industrial MCS architecture, different criticality levels introduce diverse requirements in the design and verification, which lead to different fault-tolerance capabilities among different critical applications (i.e. faults have more possibility occurring in a low-criticality application, compared to a high-criticality application [5], [7]). Spatial and fault isolation avoid fault propagation between applications with different criticality levels. Whilst temporal isolation effectively avoids the propagation of malfunctions, e.g. caused by consuming too high processor execution time (performance isolation) [2], [7]. This leads to the second Pragmatic Gap:

Pragmatic Gap II. *In the industrial MCS architecture, temporal isolation, spatial isolation and fault isolation must be guaranteed between the different critical elements.*

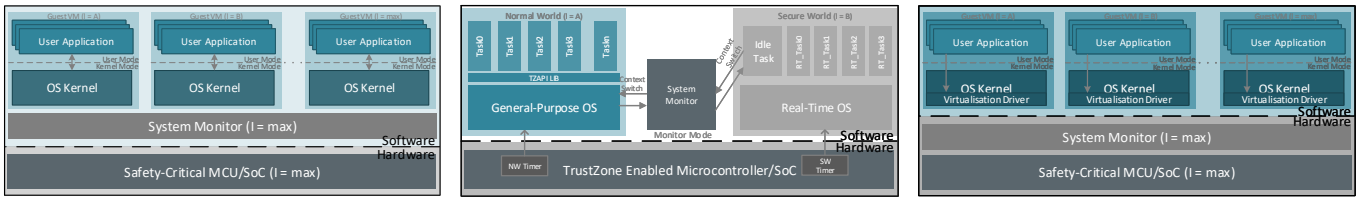
Although application-level isolation between different critical components has already been introduced in existing MCS frameworks, the required isolation must consider the entire system architecture, including Operating System (OS), system monitor, device drivers, and hardware platform. Detailed discussion regarding partitioning and isolation is provided in [1].

IV. Practical Mixed-Criticality System Architecture (P-MCS)

With sufficient consideration of the AMC model and industrial requirements, we propose a generic industrial MCS architecture, termed *P-MCS* (Practical-Mixed-Criticality System) to bridge the pragmatic gaps identified. The *P-MCS* inherits most of its features from the AMC model (e.g. system mode switch) with the additional consideration of industrial requirements. The proposed architecture is suitable for generic and sightly forward-looking MCS industrial scenarios, e.g. systems with more than four criticality levels.

A. Run-time Safety Analysis

To determine the correct 'important functions' in *system mode switch* (Pragmatic Gap I), a run-time two-level safety analysis is proposed to examine each task at the current system mode, and to determine the *preserved* task set and the *terminated* task set for the system mode to be switched into.



(a) Software Virtualisation (b) TrustZone-based Implementation (c) Hardware Acceleration

Fig. 1. Implementation Examples of *P-MCS* (Two Criticality Levels Supported)

A *preserved* task in mode K indicates the task is allowed to execute in mode K but with a lower criticality level. A *terminated* task in mode K is a task that will be terminated during the mode switch to K.

Level 1: Failure Modes and Effect Analysis (FMEA[2]). At the first level, the impacts of terminating each task are analysed. If the termination of the task causes an unacceptable consequence in the next system mode, the task is preserved. Otherwise, the task is added into the termination list for the time being and is passed to the level 2 analysis.

Level 2: Dependency Analysis. At this level, the dependency of the important tasks (output from level 1) is analysed. Any task that can cause corruption of an important task due to its termination, has to be kept in the next system mode.

B. Support for Isolation

Considering isolation (Pragmatic Gap II), applications with different criticality levels are allocated in the independent executing environment (including spatial, temporal and fault isolation). The mapping from isolated application groups to operating systems is performed off-line by allocation algorithms (e.g. Worst-Fit Decreasing). The corresponding scheduling between the isolated environment is supported by the system monitor (more privileged). Because the hardware platform and low-level drivers can be simultaneously accessed by different critical applications without '*freedom from interference*', both hardware platforms[3] and shared low-level drivers are designed and executed at the highest criticality level. Following the same rationale, the system monitor is also executed at the highest criticality level. Differently, with independent high-level drivers or operating systems being provided to the different critical applications (e.g. kernel separation), such high-level drivers or OSs only need to inherit the highest criticality from accessing applications. Notably, some low-level drivers are not accessed by all components [4]. These low-level drivers only need to inherit the highest criticality of the accessed applications.

C. Implementing P-MCS

We introduce three possible ways to implement *P-MCS*.

1) Software Virtualisation ($P|swv$): Virtualisation technology enables spatial, temporal, and fault isolation between guest VMs [12]. In order to achieve isolated executing environment for applications with different criticality, $P|swv$ assigns different criticality levels to guest VMs and allocates the applications correspondingly. During *system mode switch*, if all tasks in a guest VM are suspended, the guest VM can be terminated directly. $P|swv$ is agnostic to the applied virtualisation. In our implementation, we adopt Xen [13]. The system architecture of $P|swv$ is shown in Fig. 1(a).

System Monitor. In $P|swv$, system monitor is integrated into the ready-built Virtual Machine Monitor (VMM), e.g. Xen hypervisor [13], which is mainly responsible for: virtualisation (e.g. interposition), the run-time safety analysis, system mode switch (including execution monitoring), and real-time scheduling of the VMs.

2) Trusted-Execution Environment ($P|tz$): ARM TrustZone is a hardware-assisted separation technology introduced in Cortex-A processors since 2004 [14]. This technology is centred around the concept of separating the system execution into two independent executing environments (i.e. *secure world* and *normal world*). Such worlds are granted uneven privileges, e.g. normal software is prevented from directly accessing secure world resources. For isolation, critical and less-critical applications are allocated to the secure and the normal world, respectively. The system architecture is presented in Fig. 1(b). The key limitation of this implementation is that only two criticality levels can be supported on each processor.

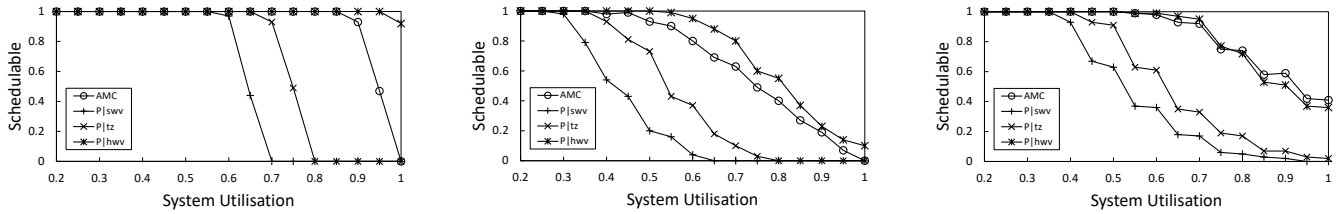
System Monitor. In $P|tz$, the system monitor is implemented in the additional privileged mode (i.e. monitor mode), and is mainly responsible for: inter-domain context switches, run-time safety analysis, and system mode switch (including execution monitoring).

3) Hardware-assisted Virtualisation ($P|hwv$): In *P-MCS*, the additional features introduced in the system monitor and the isolation imposed between different critical domains lead to extra overhead compared to the conventional theoretical model. This overhead significantly undermines the system performance and predictability. $P|hwv$ proposes the same design concept as $P|swv$, but implements the system monitor based on an open-source hardware-designed VMM [15]. The hardware-designed VMM effectively offloads the previously introduced overhead and low-layer drivers from the software to hardware, which simplifies the access paths between the VMs and the underlying hardware. The methodology effectively improves system performance and schedulability, compared to previous implementations. The system architecture of $P|hwv$ is presented in Fig. 1(c). The key limitation of the implementation is the additional hardware overhead and modification of OS kernels (to support hardware-designed hypervisor).

System Monitor. In $P|hwv$, the system monitor is integrated into the hardware-designed VMM, which is mainly responsible for: the majority of virtualisation, run-time safety analysis,

[2]The details of implementing FMEA can be found in any textbook of safety.

[3]Like the software, criticality levels must also be assigned to the hardware components (out-of-scope of this paper, see [4]).



(a) low mode (b) mode switch (c) high mode

Fig. 2. System schedulability under each execution mode

system mode switch (including execution monitoring), and real-time scheduling of the VMs.

V. Response Time Analysis

With the system architecture and run-time behaviours described in Sec. IV, the response time analysis is developed in this section to provide the timing bound for the *P-MCS*. Different from the response time analysis for the traditional dual-mode (high and low) AMC architecture [16], the analysis proposed along with *P-MCS* is generic (applicable to systems with two or more modes), with additions to reflect the unique features of the proposed architecture (e.g. run-time safety analysis and task preservation). We acknowledge that a tighter response time bounding during a mode change is presented in [16]. For details of the analysis, please see [1].

VI. Evaluation

To compare the resulting schedulability of *P-MCS* with the traditional AMC model, the typical uni-processor dual-mode MCS system is used for evaluation. The utilisation of each task is generated by the UUniFast-Discard algorithm, with a total system utilisation bound given by $0.05 \times |\Gamma|$. Task utilisation is computed for the low-mode. Periods are generated in a log-uniform distribution between $[1ms, 1000ms]$, with implicat deadlines. Priorities are given by the deadline-monotonic policy. Among all generated tasks, half of the tasks are randomly chosen to assign with the low criticality level, with others set to high-criticality tasks. The system has 4 severity levels with $S_{low} = 1$ and $S_{high} = 3$. For each task, its severity is generated randomly in a uniform distribution between $[1, 4]$. In addition, task dependency under *P-MCS* is considered and is generated randomly, in which each high-criticality task has 10% possibility to have a dependent low-criticality task. 10,000 systems are performed for each test configuration.

Results. Fig. 2 presents the percentage of schedulable systems of the AMC model (tested by the analysis in [16]) and *P-MCS* (tested by the analysis proposed in Sec. V) under each system execution state with shared resources. The system utilisation is incremented by 5%. With the system running in the low mode (Fig. 2(a)), the AMC model outperforms $P|swv$ due to the additional facilitiesin the *P-MCS*. For $P|tz$, the schedulability loss is reduced by implementing the system monitor on hardware. Notably, $P|hmv$ at the low mode outperforms the AMC model even with the additional costs, via the hardware acceleration. Similar observations are also obtained during the mode switch (Fig. 2(b)). However, the schedulability difference between $P|hwv$ and AMC under the mode switch becomes less significant due to the execution of safety analysis and the potential increased interference from the system monitor. At the high mode (Fig. 2(c)), the schedulability of AMC and $P|hmv$ becomes similar (with AMC slightly better) for preserving additional low-criticality tasks, where more tasks will be preserved under *P-MCS* with the increase of utilisation.

VII. Conclusion

In this paper, we formalise and analyse the mismatches between the MCS theoretical models and industrial standards via the system architecture perspective. Then, we present a generic industrial architecture (i.e. *P-MCS*) upon the conventional AMC, with additional satisfaction on the industrial safety requirements – i.e. run-time safety analysis and isolation between different critical elements. Experimental results show that the hardware-based implementation of P-MCS effectively alleviates the additional cost for satisfying the industrial safety requirements and outperforms the traditional AMC model.

For more details regarding the paper, please see [1].

References

[1] Z. Jiang, S. Zhao, R. Wei, D. Yang, R. Paterson, N. Guan, Y. Zhuang, and N. C. Audsley, "Bridging the pragmatic gaps for mixed-criticality systems in the automotive industry," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1116–1129, 2021.

[2] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[3] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *RTSS*, 2007.

[4] "26262 road vehicles-functional safety," *International Standard ISO*, 2018.

[5] A. Esper, G. Nelissen, V. Nélis, and E. Tovar, "An industrial view on the common academic understanding of mixed-criticality systems," *Real-Time Systems*, 2018.

[6] P. Graydon and I. Bate, "Safety assurance driven problem formulation for mixed-criticality scheduling," *Proc. WMC, Real-Time Systems Symposium*, pp. 19–24, 2013.

[7] R. Ernst and M. Di Natale, "Mixed criticality systems—a history of misconceptions?" *IEEE Design & Test*, 2016.

[8] Z. Jiang, S. Zhao, P. Dong, Y. Dawei, R. Wei, N. Guan, and N. Audsley, "Re-thinking mixed-criticality architecture for automotive industry," in *38th International Conference on Computer Design*, 2020.

[9] N. Kim, S. Tang, N. Otterness, J. H. Anderson, F. D. Smith, and D. E. Porter, "Supporting I/O and IPC via fine-grained OS isolation for mixed-criticality RT tasks," in *RTNS*, 2018.

[10] S. Baruah and A. Burns, "Implementing mixed criticality systems in ada," in *International Conference on Reliable Software*, 2011.

[11] D. D. Ward and S. E. Crozier, "The uses and abuses of asil decomposition in iso 26262," in *7th IET International Conference on System Safety, incorporating the Cyber Security Conference*, 2012.

[12] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[13] X. Website, "https://xenproject.org/," 2020.

[14] S. Pinto and N. Santos, "Demystifying trustzone: A comprehensive survey," *ACM Computing Surveys (CSUR)*, 2019.

[15] Z. Jiang, N. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *RTAS*, 2018.

[16] S. Baruah, A. Burns, and R. Davis, "Response-time analysis for mixed criticality systems," in *Real-Time Systems Symposium*, 2011.