Probabilistic Safe WCET Estimation for Weakly Hard Real-Time Systems at Design Stages

JAEKWON LEE, University of Luxembourg, Luxembourg and University of Ottawa, Canada
SEUNG YEOB SHIN, University of Luxembourg, Luxembourg
LIONEL C. BRIAND, University of Luxembourg, Luxembourg and University of Ottawa, Canada
SHIVA NEJATI, University of Ottawa, Canada

Weakly hard real-time systems can, to some degree, tolerate deadline misses, but their schedulability still needs to be analyzed to ensure their quality of service. Such analysis usually occurs at early design stages to provide implementation guidelines to engineers so that they can make better design decisions. Estimating worst-case execution times (WCET) is a key input to schedulability analysis. However, early on during system design, estimating WCET values is challenging and engineers usually determine them as plausible ranges based on their domain knowledge. Our approach aims at finding restricted, safe WCET sub-ranges given a set of ranges initially estimated by experts in the context of weakly hard real-time systems. To this end, we leverage (1) multi-objective search aiming at maximizing the violation of weakly hard constraints in order to find worst-case scheduling scenarios and (2) polynomial logistic regression to infer safe WCET ranges with a probabilistic interpretation. We evaluated our approach by applying it to an industrial system in the satellite domain and several realistic synthetic systems. The results indicate that our approach significantly outperforms a baseline relying on random search without learning, and estimates safe WCET ranges with a high degree of confidence in practical time (< 23h).

CCS Concepts: • **Computer systems organization → Real-time systems**; • **Software and its engineering → Real-time schedulability**; **Search-based software engineering**; **Empirical software validation**; • **Computing methodologies → Machine learning**.

Additional Key Words and Phrases: Worst-case execution time, Weakly hard real-time systems, Meta-heuristic search

ACM Reference Format:
Jaekwon Lee, Seung Yeob Shin, Lionel C. Briand, and Shiva Nejati. 2018. Probabilistic Safe WCET Estimation for Weakly Hard Real-Time Systems at Design Stages. *ACM Trans. Softw. Eng. Methodol.* 0, 0, Article 0 (2018), 35 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Real-time systems are required to perform operations under time constraints, specifying execution deadlines [23]. In real-world applications across many industry sectors, such as automotive and aerospace, real-time systems can often tolerate occasional deadline misses when their consequences

Authors' addresses: Jaekwon Lee, jaekwon.lee@uni.lu, University of Luxembourg, 29 Avenue John F. Kennedy, Luxembourg, 1859, Luxembourg and University of Ottawa, 800 King Edward Avenue, Ottawa, ON K1N 6N5, Canada; Seung Yeob Shin, seungyeob.shin@uni.lu, University of Luxembourg, 29 Avenue John F. Kennedy, Luxembourg, 1859, Luxembourg; Lionel C. Briand, lionel.briand@uni.lu, University of Luxembourg, 29 Avenue John F. Kennedy, Luxembourg, 1859, Luxembourg and University of Ottawa, 800 King Edward Avenue, Ottawa, ON K1N 6N5, Canada; Shiva Nejati, snejati@uottawa.ca, University of Ottawa, 800 King Edward Avenue, Ottawa, ON K1N 6N5, Canada.

1049-331X/2018/0-ART0 $15.00
https://doi.org/XXXXXXX.XXXXXXX

are negligible with respect to achieving the system objectives and are not noticeable by users. The systems that are robust to occasional deadline misses are known as weakly hard real-time systems [11]. Weakly hard deadline constraints specify the extent to which real-time tasks can tolerate deadline misses. For example, a control-loop task that computes throttle angles and sends throttle commands to an autonomous vehicle at a fixed rate (e.g., 20 Hz) can accept at most three deadline misses out of 20 task arrivals. When the task violates the weakly hard deadline constraint, e.g., four deadline misses within 20 task arrivals, it may result in the vehicle failing to arrive at the target destination on time or even colliding with objects.

While developing a weakly hard real-time system, estimating *safe* worst-case execution times (WCETs) of real-time tasks is an important activity to ensure that the system meets its deadline constraints. Engineers deem tasks' WCETs to be safe when, under such specified execution times, task executions satisfy their (weakly hard) deadlines constraints; i.e., the tasks are schedulable [13]. In particular, safe WCET estimates are practically useful at early design stages when tasks' implementations are not yet completed. Such estimates provide development objectives to guide engineers in making appropriate design and implementation decisions and thus prevent deadline misses. For example, depending on the safe WCET estimated for a data-processing task, engineers may choose either an in-memory storage, a file system, or an external database system to store and access data.

At early design stages, engineers find it challenging to estimate safe WCETs and thus to guarantee that tasks always meet their deadlines [37]. WCETs are determined based on a variety of factors such as task scheduling policies, task implementations, and hardware specifications. Regarding scheduling policies, advanced real-time operating systems (e.g., QNX Neutrino [16]) applied in industry employ sophisticated scheduling policies to accommodate various systems' requirements in different domains, such as automotive and aerospace. For example, an adaptive partitioning scheduler (APS) [15] developed by BlackBerry prevents unimportant tasks from monopolizing system resources (e.g., processing units) by using adaptive partitions. Such partitions separate tasks into virtual containers with their own resource-utilization budgets, which are adaptive depending on system performance. Due to the complexity of such scheduling policies, engineers face difficulties when applying existing WCET analysis techniques [24, 63] that are valid only when systems employ traditional scheduling policies, e.g., rate monotonic scheduling policy [52]. Furthermore, the problem of estimating safe WCETs becomes more challenging (i.e., computationally expensive) when real-time systems are constrained by weakly hard deadlines, which specify tolerable degrees for deadline misses. In addition, decisions regarding task implementations and hardware components are often not fully known at early design stages. Hence, engineers cannot determine exact WCET values ensuring that tasks are schedulable. Therefore, engineers usually resort to estimating WCET ranges that can ensure tasks are schedulable with a high probability [27, 50].

The problem of estimating WCET has been widely studied, relying mainly on measurements [24, 63, 73] and static analysis [32, 39, 58, 66]. Measurement-based approaches estimate WCETs by analyzing multiple executions on the target hardware or an accurate simulator using a set of worst-case inputs. In contrast, static analysis-based approaches estimate WCETs by investigating the longest path in source code and the cache hit ratio based on hardware specifications. There are approaches [5, 18, 37] aiming at estimating WCETs at early stages of implementation. For example, Altenbernd et al. [5] first create a timing model that predicts the execution times of machine instructions. Given source code to analyze, they then translate it to machine instructions captured in the timing model. The execution times of these instructions are used to approximate the source code's WCET. In contrast to our work that aims at estimating safe WCET ranges, these prior approaches aim at estimating the WCET of a real-time task, without accounting for the task's schedulability (i.e., deadline constraints). In addition, since these approaches rely on source code

available only at implementation stages, they are not applicable at early design stages. Recently, SAFE [50] has been proposed to estimate with a probabilistic interpretation, at early design stages, safe WCET ranges that satisfy deadline constraints for real-time systems. SAFE utilizes task models instead of source code to simulate task executions and estimate safe WCET ranges using machine learning and meta-heuristic search. However, SAFE does not account for the specificities of weakly hard real-time systems accepting occasional deadline misses and advanced, sophisticated scheduling policies used in industry. Instead, SAFE targets real-time systems that do not tolerate any occurrence of a deadline miss and relies on a simple task model. Hence, the problem addressed in our work is more complex than the problem tackled by SAFE. Our work complements SAFE and extends it to probabilistically estimate safe WCET ranges for weakly hard real-time systems involving advanced industry scheduling policies.

Contributions. In this article, we propose SWEAK, a Safe WCET analysis method for wEAKly hard real-time systems. SWEAK searches for effective test cases that likely cause violations of weakly hard deadline constraints using a multi-objective search algorithm [54]. SWEAK then estimates safe WCET ranges with a probabilistic interpretation by using logistic regression [45]. SWEAK evaluates the schedulability of a set of real-time tasks by using an industrial scheduler APS that supports complex scheduling policies, accounting for multi-core platforms and adaptive partitions [64]. In a multidimensional WCET space defined by different tasks in a system, SWEAK identifies a *safe WCET border* characterizing safe WCET ranges with a probability p of violating weakly hard deadline constraints. Such a border allows engineers to investigate, for each task, suitable WCET values by analyzing trade-offs within the safe ranges.

We evaluated SWEAK with an industrial system from the satellite domain and several realistic synthetic systems that were created following guidelines provided by our industry partner, Blackberry. Experimental results show that SWEAK can efficiently and accurately estimate safe WCET ranges for various weakly hard real-time systems. Regarding the execution time of SWEAK, it takes at most 22.1h across a large number of synthetic systems, indicating that SWEAK is acceptable in practice as an offline analysis tool. All the details of our evaluation results are available online [49].

Organization. This article is organized as follows: Section 2 motivates our work. Section 3 precisely defines the problem of estimating safe WCET ranges for weakly hard real-time systems. Section 4 describes SWEAK. Section 5 empirically evaluates SWEAK. Section 6 contrasts SWEAK against related work. Section 7 concludes this article.

2 MOTIVATION

Our work is motivated by the practical needs identified in collaboration with our partner company, BlackBerry. They have developed a real-time operating system (RTOS), named QNX Neutrino [16], which satisfies the functional safety standard ISO-26262 [44] with the highest automotive safety integrity level (ASIL-D). Due to the stringent assurance requirements, QNX Neutrino has been used in many safety-critical, real-time industries such as automotive and medical domains.

Adaptive partitioning scheduler (APS). QNX Neutrino employs a sophisticated scheduler named APS, which has been studied and applied in many systems [2, 17, 25, 26, 56], to support complex system requirements in managing real-time tasks. APS is based on a priority-driven preemptive scheduling policy, allocating tasks to processing cores based on the tasks' priorities for scheduling. The policy ensures that the highest priority task always has access to a processing core when required. APS also supports task partitions in which tasks are assigned. The time budgets of partitions, which impact task executions, are dynamically controlled depending on the system load. Such budget management not only scales up and down the budgets of partitions according to the tasks' demands, but also prevents tasks from monopolizing processors. In addition, APS supports various scheduling policies, e.g., FIFO and Round-Robin, and multi-core platforms. These

rich features of APS (and QNX Neutrino) have made it widely applicable in practice, but have also complicated schedulability analysis.

Analysis needs. BlackBerry provides customers who develop real-time systems with an APS simulator, allowing them to design and evaluate real-time tasks running on QNX Neutrino in a realistic and scalable manner. In particular, the APS simulator emulates tasks' behaviors without requiring their implementations or hardware devices. Hence, the simulator is applicable to analyze real-time tasks at early design stages. However, the simulator cannot estimate safe WCET ranges at early stages, which is important to develop and assess real-time systems.

Many organizations [3], including BlackBerry customers, develop weakly hard real-time systems that can tolerate occasional deadline misses. However, existing WCET analysis techniques [6, 12, 50] work on hard real-time systems with simpler scheduling policies than APS. In such contexts, using state-of-the-art techniques would therefore result in unnecessarily restricted WCET ranges compared to acceptable WCET ranges that would allow tasks to occasionally miss some deadlines. In practice, as early WCET estimates guide engineers to make appropriate implementation decisions and hardware resource choices, overly pessimistic WCET estimates may add unnecessary overhead for code optimization or over-provisioning of hardware resources. To this end, BlackBerry is interested in developing an APS simulation-based solution to estimate safe WCET ranges for weakly hard real-time systems.

3 PROBLEM DEFINITION

This section introduces the notation we use in this article and our task model. The latter builds on our previous work [50] and extends it with weakly hard deadline constraints, complex scheduling policies, and context switching times. We then describe the problem of identifying safe WCET ranges that satisfy the deadline constraints with a certain level of confidence.

Task model. We analyze a real-time system running n tasks in parallel on a multi-core platform. Each task τ_i ($1 \leq i \leq n$) is identified as either periodic or aperiodic. A periodic task, which arrives at regular intervals, is characterized by the following temporal parameters: offset O_i, period T_i, WCET C_i, and relative deadline D_i. These parameters determine the arrival times of a periodic task and its absolute deadlines. Specifically, the kth arrival of a periodic task τ_i, denoted by $a_{i,k}$, is $O_i + (k-1) \times T_i$. A periodic task arrived at $a_{i,k}$ is supposed to complete its execution, even in the worst case C_i, before the absolute deadline determined by $a_{i,k} + D_i$.

An aperiodic task has irregular arrival times as it is activated by external stimuli. In general, there is no limit on the arrival times of an aperiodic task. However, in real-time analysis, we typically specify a minimum inter-arrival time and maximum inter-arrival time to characterize irregular arrivals. For an aperiodic task τ_j, we define $[T_j^{min}, T_j^{max}]$, indicating the minimum and maximum time intervals between two consecutive arrivals of τ_j. Thus, an arrival time $a_{j,k}$ is determined by the $k-1$th arrival time of τ_j and its minimum and maximum arrival times as follows: $[T_j^{min}, T_j^{max}]$ for $k = 1$ and $[a_{j,k-1} + T_j^{min}, a_{j,k-1} + T_j^{max}]$ for $k > 1$. We note that, in real-time analysis, sporadic tasks can be separately defined as they have irregular arrivals and hard deadlines [53]. However, in our task model, we do not introduce new notations for sporadic tasks because the deadline and period concepts defined above are sufficient to characterize them.

Under a priority-driven preemptive scheduling policy [13], each task τ_i has its priority, denoted by P_i, which determines its order of execution. A task τ_i can preempt another task τ_j when the priority value of τ_j is less than the value of τ_i, i.e., $P_j < P_i$.

WCET ranges. As discussed earlier, engineers have a hard time estimating exact WCET values at early stages of development because there are many uncertain factors, for example related to hardware configuration, input data, and source code. In this study, we assume that engineers can

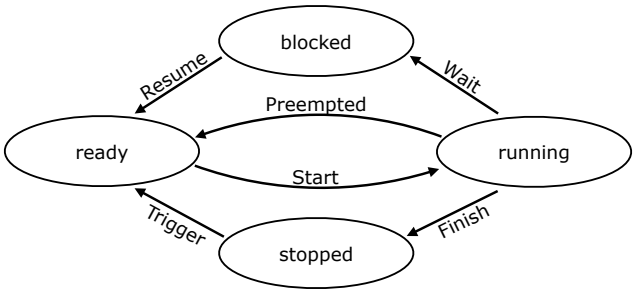


Fig. 1. A task state transition model.

provide WCET ranges instead of a single value. We denote by $[C_i^{min}, C_i^{max}]$ the minimum and maximum WCET values of a task τ_i.

Weakly hard deadline constraints. A deadline D_i is the time constraint of a task τ_i. Our task model requires the deadline to be greater than or equal to C_i [22], i.e., $D_i \geq C_i$. For a task τ_j with a WCET range, the deadline value must be greater than or equal to the maximum WCET value C_j^{max}. A task constrained by a hard deadline must meet its deadline in all executions of the task. However, when a task is subject to a weakly hard deadline constraint, which is also called a soft deadline, the task can occasionally miss its deadline. We adopt the formal definition of weakly hard deadline constraints introduced by Bernat et al. [11], i.e., (m, K)-constraint, which has been commonly used in weakly hard real-time studies [4, 35, 60]. The (m_i, K_i)-constraint of a task τ_i specifies that, within a time window K_i (i.e., the number of consecutive arrivals of τ_i), m_i consecutive task arrivals are allowed to miss the deadline D_i. For instance, $(m_i, K_i) = (2, 5)$ specifies that two consecutive deadline misses of a task τ_i are acceptable within five consecutive arrivals of τ_i. When τ_i is subject to a hard deadline constraint, $m_i = 0$. We note that, in weakly hard real-time systems, analyzing consecutively missed deadlines is important as the consequence of a deadline miss at a task arrival is propagated to the next arrivals. However, non-consecutive deadline misses may not be noticeable to users as a task execution may complete before its deadline even after a deadline miss occurred at the previous arrival.

Context switching times. In addition to the task model described above, our study accounts for context switching time, which is the time required for a task to change state during scheduling. Fig. 1 shows the task state transition model used in APS. According to this model, a task is *ready* when it is prepared to execute on a processing core. APS executes the task by assigning it to an idle processing core and sets its state to *running*. A *running* task can be *blocked* when it requires resources used by other tasks or *stopped* when it finishes its execution. APS can also preempt a *running* task when a higher priority task is *ready* or the partition budget that the task belongs to runs out. Each state transition requires time for exchanging data between memories and scheduling overheads. To account for such time, we define three types of context switching times. Start-up time, denoted by λ_s, is the time required to change the state of a task from *ready* to *running*. Exit time, denoted by λ_x, is the time required to change the state of a task from *running* to other states, *ready* or *blocked*, or to finalize its execution. Moreover, since APS deals with multi-core platforms, tasks may need to be assigned to different processing cores when their states are changed from *ready* to *running* depending on the availability of processing cores. Inter-processor interrupt (IPI) time, denoted by λ_p, is required to transfer a task execution from one core to another in a multi-core platform. These context switching times are affected by hardware performance as well as scheduling

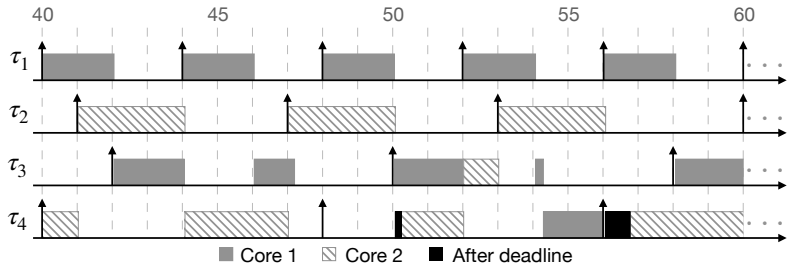


Fig. 2. A schedule scenario example that describes task executions of four tasks, τ_1, τ_2, τ_3, and τ_4, running on a two-core platform. This scenario includes context switching times.

overhead in a scheduler, which are uncertain. Hence, we specify them as ranges instead of single values. For example, the start-up time λ_s can be any value in the range $[0.012ms, 0.022ms]$. Note that representing these context switching times as ranges is consistent with the APS simulator provided by BlackBerry QNX.

Schedule scenario Based on a scheduling result, we define a schedule scenario, denoted by S, describing the executions of all tasks in a system in terms of their start and end times. Specifically, we formulate a schedule scenario as a list of tuples $(\tau_i, a_{i,k}, e_{i,k})$, where $a_{i,k}$ and $e_{i,k}$ are, respectively, the arrival time and the end (or completion) time of the kth arrival of the task τ_i.

For example, let Γ be a set of n tasks to be scheduled by a real-time scheduler. A scheduler then dynamically schedules the executions of tasks in Γ over the scheduling period $\mathbb{T} = [0, \mathbf{t}]$ according to a scheduling policy (e.g., APS scheduling policy [15]). Fig. 2 describes a schedule result running four tasks τ_1, τ_2, τ_3, and τ_4 on a two-core platform. The periodic task τ_1 is characterized by: $O_1 = 0$, $T_1 = 4$, and $C_1^{min} = C_1^{max} = 2$. The aperiodic task τ_2 is characterized by: $[T_2^{min}, T_2^{max}] = [6, 12]$ and $C_2^{min} = C_2^{max} = 3$. The aperiodic task τ_3 is characterized by: $[T_3^{min}, T_3^{max}] = [8, 14]$, and $[C_3^{min}, C_3^{max}] = [2, 3]$. The periodic task τ_4 is characterized by: $O_4 = 0$, $T_4 = 8$, and $[C_4^{min}, C_4^{max}] = [2, 4]$. All the task executions should be finished before the next task period or next minimum task arrival (i.e., $D_1 = T_1 = 4$, $D_2 = T_2^{min} = 6$, $D_3 = T_3^{min} = 8$, and $D_4 = T_4 = 8$). The tasks' priorities are $P_1 > P_2 > P_3 > P_4$, which means that τ_1 can preempt the processing cores at any time. All the context switching times in the example are 0.025, i.e., $\lambda_s = \lambda_x = \lambda_p = 0.025$.

As shown in Fig. 2, during the scheduling period $\mathbb{T} = [40, 60]$, the schedule scenario S is $\{(\tau_1, 40, 42.05), \cdots, (\tau_2, 41, 44.075), \cdots, (\tau_3, 42, 47.2), \cdots, (\tau_4, 40, 50.3), \cdots, (\tau_4, 56, 60.825)\}$. Due to randomness in task execution times, aperiodic task arrivals, and context switching times, a schedule scenario (i.e., scheduling result) can differ in each scheduler run.

Schedulability. We analyze the schedulability of a given schedule scenario S by checking the (m_i, K_i)-constraint of each task in S. If a schedule scenario shows a violation of any (m_i, K_i)-constraint, the schedule scenario is not schedulable. For example, the schedule scenario in Fig. 2 has two deadline misses for task τ_4 at the first and second arrivals. If the deadline constraint (m_4, K_4) of τ_4 is $(1, 4)$, the schedule scenario S is not schedulable as the deadline constraint of τ_4 only allows one deadline miss in four consecutive arrivals. However, the scenario S becomes schedulable when $(m_4, K_4) = (2, 4)$, accepting two consecutive deadline misses. Note that a set Γ of tasks is schedulable when every schedule scenario S of Γ is schedulable with respect to the tasks' (m_i, K_i)-constraints.

Problem. The effective design and assessment of real-time systems rely on the accurate evaluation of the task parameters. Among these parameters, WCET values are estimated as ranges, which is inevitable given the high uncertainty at early stages of development. Upper WCET bounds are the worst-case WCET values that are most likely to have deadline misses, since larger WCET

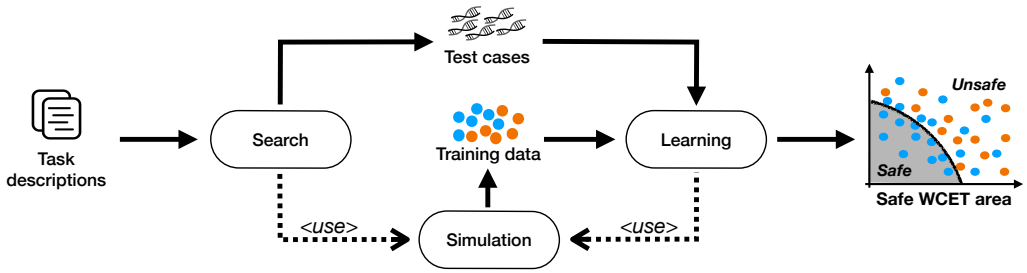


Fig. 3. An overview of our Safe WCET analysis method for wEAKly hard real-time system (SWEAK).

values increase the probability of deadline constraint violations. Lower WCET bounds are tasks' best-case WCET values but are harder to implement in practice.

Our work aims at determining the maximum upper bounds that allow tasks to be schedulable, under weakly hard deadline constraints, at a certain level of probability of violating deadline constraints. Practitioners can use these upper bounds as an objective when implementing the tasks. Specifically, for every task $\tau_i \in \Gamma$ to be analyzed, our approach computes a new upper bound value for the WCET range of τ_i (denoted by C_i^{max*}) by restricting it from C_i^{max} to C_i^{max*} such that, at a certain level of confidence, deadline constraint violations should not occur. For instance, as we aforementioned, the schedule scenario in Fig. 2 is not schedulable under (1,4)-constraint for the task τ_4. However, the tasks become schedulable when restricting the maximum WCET of τ_4 from $C_4^{max} = 4$ to $C_4^{max*} = 3$ or WCET of τ_3 from $C_3^{max} = 3$ to $C_3^{max*} = 2$.

4 APPROACH

Fig. 3 shows an overview of SWEAK, our Safe Worst-case execution time (WCET) analysis method for wEAKly hard real-time system. Given task descriptions, SWEAK first finds test cases, which consist of sequences of task arrivals and context switching times, using meta-heuristic search to maximize the magnitude and consecutiveness degree of deadline misses (Section 4.1). During search, SWEAK uses an industrial scheduling simulator, i.e., APSSimulator that simulates the APS policy, to evaluate the schedulability of test cases and produce a training dataset (Section 4.2). Using the training data, SWEAK then builds a logistic regression model to distinguish between the *safe* and *unsafe* areas in the WCET space with respect to satisfying and violating weakly hard deadline constraints (Section 4.3). The model estimates, with a probabilistic interpretation, safe WCET ranges under which tasks are likely to be schedulable. To improve the accuracy of the estimation model, SWEAK then augments the training dataset by running simulations with the test cases obtained from the search. In the next sections, we describe each step of SWEAK in detail.

We note that SWEAK is based on our past work (i.e., SAFE [50]), which estimates probabilistic safe WCET ranges for (hard) real-time systems using a single-objective search algorithm, single-queue multi-core scheduling policy simulation, and logistic regression. In contrast to SAFE, which targets real-time systems with relatively simple scheduling policies, SWEAK aims at estimating probabilistic safe WCET ranges for weakly hard real-time systems involving advanced industrial scheduling policies. Recall from Section 3 that the (m_i, K_i)-constraint (i.e., weakly hard constraint) of a task τ_i specifies that, within a time window K_i, m_i consecutive task arrivals are allowed to miss the deadline of τ_i. Such a weakly hard deadline constraint is more complex than a hard deadline constraint that does not allow any occurrence of a deadline miss. Analyzing weakly hard real-time systems, therefore, requires different techniques, compared to SAFE, to track and record the tasks

that have missed their deadlines and the frequency of such occurrences. Regarding the underlying techniques of the search step (Section 4.1) of SWEAK, it employs a multi-objective search algorithm to generate test cases that are likely to violate weakly hard deadline constraints and maximize the magnitude of deadline misses. During search, SWEAK further accounts for the context switching times, i.e., start-up, exit, and IPI times, which have an impact on scheduling results (described in Section 3). In contrast, SAFE does not consider these time aspects. For the simulation step (Section 4.2), SWEAK uses APSSimulator that simulates the APS policy. Regarding the learning step (Section 4.3), SWEAK also opts to use logistic regression, similar to the learning step of SAFE, to provide a probabilistic interpretation for safe WCET ranges. Hence, we adapt the learning step of SAFE to develop the learning step of SWEAK, which accounts for weakly hard deadline constraints and integrates with APSSimulator.

4.1 Searching for effective test cases

The search step of SWEAK aims to generate test cases that likely violate weakly hard deadline constraints and maximize the magnitude of deadline misses. We apply a multi-objective search algorithm for finding test cases guided by the following two objectives: (1) maximizing the magnitude of deadline misses and (2) maximizing the consecutiveness degree of deadline misses. These two objectives enable the search step to generate test cases that cause larger deadline misses (in terms of distances between tasks' completion times and their deadlines) and more consecutive deadline misses. To evaluate the test cases with respect to the objectives through simulations, SWEAK applies multiple sets of WCET values, that are randomly sampled within their specified ranges, since WCET values can lead to different schedule results with the same test case. We describe our search-based approach by defining the solution representation, the fitness functions, and the computational search algorithm, as recommended in the checklists for search-based software engineering research [61].

Representation. A feasible solution represents a test case for checking the schedulability of a set of tasks defined in the input task descriptions. Given a set Γ of tasks to be scheduled, a solution I consists of two parts: context switching times and sequences of task arrivals for all tasks in Γ. The context switching times are three scalar values, i.e., start-up λ_s, exit λ_x, and IPI λ_p times, each of which is selected within their valid ranges (see Section 3). The sequences of task arrivals are denoted by a set A of tuples $(\tau_i, a_{i,k})$, where $\tau_i \in \Gamma$ and $a_{i,k}$ is the kth arrival time of τ_i. The number of arrivals of τ_i is restricted by the scheduling time period $\mathbb{T} = [0, \mathbf{t}]$. For example, if a task τ_i is periodic and its offset $O_i = 0$, the number of τ_i arrivals is $\mathbf{t}/T_i$ where T_i denotes the period of τ_i (see Section 3). In the case of aperiodic tasks, the number of arrivals varies with changing inter-arrival times (see Section 3). Therefore, the size of I varies across different solutions along with the size of A.

Fitness. To evaluate the fitness of each solution, we define two objective functions, which quantify the magnitude of deadline misses and the consecutiveness degree of deadline misses. These objective functions compute fitness values using multiple simulations to account for uncertainty in WCETs. Specifically, given a solution I for a set Γ of tasks, SWEAK runs APSSimulator ns times with WCET values for the tasks in Γ that are randomly selected from their WCET ranges and thus obtains schedule scenarios $\mathbf{S} = \{S_1, S_2, \cdots, S_{ns}\}$ (see Section 4.2). Given the scenarios, SWEAK calculates the fitness values for a solution I using the fitness functions described below.

Fitness for the magnitude of deadline misses. We denote by $fd(I, \Gamma^\delta, ns)$ a fitness function that quantifies the magnitude of deadline misses regarding a solution I, a set $\Gamma^\delta \subseteq \Gamma$ of target tasks, and ns simulations. We note that SWEAK provides the capability of selecting target tasks Γ^δ as practitioners often need to focus on a subset of critical tasks. The function $fd(I, \Gamma^\delta, ns)$ calculates a

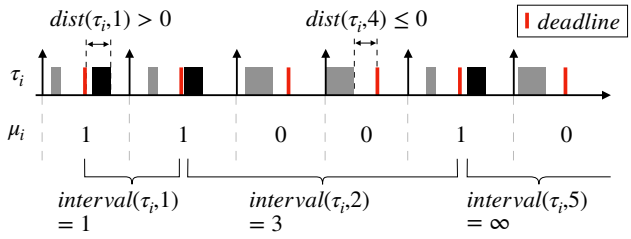


Fig. 4. An example of a μ-pattern for a task τ_i.

fitness value using a distance function $dist(\tau_i, k)$ defined as follows:

$$dist(\tau_i, k) = e_{i,k} - a_{i,k} + D_i$$

This function computes the distance between the end time and the deadline of the kth arrival of task τ_i in a schedule scenario (see notation in Section 3). If an arrival $a_{i,k}$ misses its absolute deadline $a_{i,k} + D_i$, the value of $dist(\tau_i, k)$ is larger than 0. As a larger distance value leads to a higher probability of deadline misses, SWEAK finds the maximum distance among all the arrivals for each scenario by using the fitness function $fd(I, \Gamma^\delta, ns)$ defined as follows:

$$fd(I, \Gamma^\delta, ns) = \sum_{h=1}^{ns} \max_{\tau_i \in \Gamma^\delta,\ k \in [1, lk(\tau_i)]} dist_h(\tau_i, k)/ns$$

where $lk(\tau_i)$ is the number of τ_i arrivals in I. We denote by $dist_h(\tau_i, k)$ the distance function for each schedule scenario $S_h \in \mathbf{S}$. SWEAK aims to maximize the fitness value computed by $fd(I, \Gamma^\delta, ns)$.

Fitness for the consecutiveness degree of deadline misses. We denote by $fc(I, \Gamma^\delta, ns)$ the fitness function that quantifies the consecutiveness degree of deadline misses regarding a solution I, a set $\Gamma^\delta \subseteq \Gamma$ of target tasks, and ns simulations. To compute the consecutiveness degree of deadline misses, SWEAK converts a schedule scenario into μ-patterns [11] by checking whether task arrivals in a schedule scenario meet their deadlines or not. Specifically, given a schedule scenario S, a μ-pattern μ_i for a task τ_i is a sequence of $(\mu_i(1), \ldots, \mu_i(k), \ldots, \mu_i(lk(\tau_i)))$ where k is the kth arrival of τ_i, $lk(\tau_i)$ is the number of τ_i arrivals in S, and $\mu_i(k)$ is defined as follows:

$$\mu_i(k) = \begin{cases} 1 & , dist(\tau_i, k) > 0 \\ 0 & , otherwise \end{cases}$$

Fig. 4 shows an example converting task arrivals of τ_i into a μ-pattern μ_i. The task τ_i has three deadline misses at the first, second, and fifth arrivals, resulting in μ_i to be equal to (1,1,0,0,1,0).

Based on a μ-pattern, we calculate the interval, denoted by $interval(\tau_i, k)$, between the kth and the k'th arrivals of a task τ_i, where $k' > k$, the kth and k'th arrivals miss their deadlines, and all arrivals between the kth and k'th arrivals meet their deadlines. For example, in Fig. 4, $interval(\tau_i, 2)$ is equal to 3 because, after the 2nd arrival, the next deadline miss occurs at the 5th arrival; hence, $interval(\tau_i, 2) = 5 - 2 = 3$. Note that the $interval(\tau_i, 5)$ in Fig. 4 is defined as ∞ because, after the 5th arrival of τ_i, the next deadline miss is unknown. When $\mu_i(k) = 0$, we define $interval(\tau_i, k) = 0$.

Given the function $interval(\tau_i, k)$ and a μ-pattern μ_i, we denote by $consec(\tau_i, k)$ the consecutiveness degree of deadline misses regarding the kth arrival of a task τ_i. The function $consec(\tau_i, k)$ is defined as follows:

$$consec(\tau_i, k) = \begin{cases} 10^{\frac{1}{interval(\tau_i, k)}} & , \mu_i(k) = 1 \\ 0 & , \mu_i(k) = 0 \end{cases}$$

Algorithm 1: An algorithm for searching test cases, aiming at maximizing (1) the magnitude of deadline misses and (2) the consecutiveness degree of deadline misses, based on NSGA-II.

```
1  Input Γ: a set of tasks
2  Input ns: number of samples
3  Input np: population size
4  Input pc: crossover probability
5  Input pm: mutation probability
6  Output P_α: population of test cases
7
8  P_α ← {}
9  P ← CreatePopulation(Γ, np)
10
11 repeat
12     // calculating fitness for P
13     for each I ∈ P do
14         W ← SampleWCET(ns)
15         S ← RunSimulation(I, W)
16         fd(I, Γ^δ, ns) = Σ_{h=1}^{ns} max_{τ_i∈Γ^δ, k∈[1,lk(τ_i)]} dist_h(τ_i, k)/ns
17         fc(I, Γ^δ, ns) = Σ_{h=1}^{ns} ( max_{τ_i∈Γ^δ} Σ_{k=1}^{lk(τ_i)} consec_h(τ_i, k) ) / ns
18     end for
19
20     // update archive
21     P_α ← P_α ∪ P
22     ComputeFrontRanks(P_α)
23     ComputeSparsities(P_α)
24     P_α ← SelectArchive(P_α, np)
25     BestFront ← ParetoFront(P_α)
26
27     //creating a new population
28     P ← Breed(P_α, np, pc, pm)
29 until we have run out of time or BestFront is the ideal Pareto front
30 return P_α
```

To reward small intervals and penalize large intervals between consecutive deadline misses, we invert $interval(\tau_i, k)$ and use an exponential function as shown in the $consec(\tau_i, k)$ definition. Hence, a consecutiveness degree $consec(\tau_i, k)$ exponentially decreases with the increasing value of $interval(\tau_i, k)$. For example, given the μ-pattern μ_i in Fig. 4, the value of $consec(\tau_i, k)$ decreases when the value of $interval(\tau_i, k)$ increases, i.e., $consec(\tau_i, 1) = \sqrt[1]{10} = 10$, $consec(\tau_i, 2) = \sqrt[3]{10} = 2.15$, and $consec(\tau_i, 5) = \sqrt[\infty]{10} = 1$, where $1/\infty = 0$.

To compute the fitness function $fc(I, \Gamma^\delta, ns)$, SWEAK runs APSSimulator ns times for I and obtains ns schedule scenarios $S_1, S_2, \ldots, S_{ns}$. For each schedule scenario S_h, we denote by $consec_h(\tau_i, k)$ the consecutiveness degree of deadline misses regarding the kth arrival of a task τ_i observed in each schedule scenario S_h. SWEAK aims to maximize the $fc(I, \Gamma^\delta, ns)$ fitness defined as follows:

$$fc(I, \Gamma^\delta, ns) = \sum_{h=1}^{ns} \Big(\max_{\tau_i \in \Gamma^\delta} \sum_{k=1}^{lk(\tau_i)} consec_h(\tau_i, k) \Big) / ns$$

Computational search. SWEAK employs the NSGA-II algorithm [54] as shown in Algorithm 1. It generates an initial population $\mathbf{P}$ (line 9) and iterates to evolve the population until finding the ideal Pareto front or exhausting the execution budget (lines 11–29). At each iteration, the algorithm

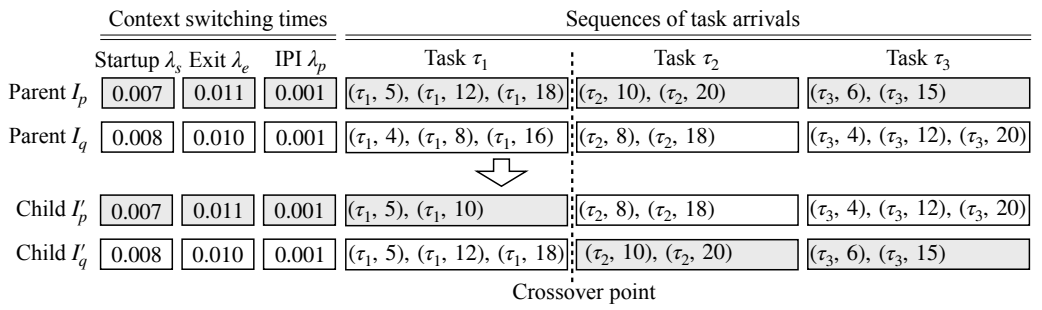


Fig. 5. An example of SWEAK's crossover operation. It swaps all context switching times and all task arrivals of task τ_1 between two parent solutions I_p and I_q to produce offspring I'_p and I'_q.

first evaluates individuals in **P** with the fitness functions defined above (lines 13–18) and adds them to the archive $\mathbf{P}_\alpha$ (line 21). The algorithm then calculates Pareto front rankings and sparsities of the solutions in the archive $\mathbf{P}_\alpha$ using the fitness values (lines 22-23). The calculation is used for determining the *np* individuals to be kept in the archive and the best Pareto front (lines 24–25). Based on the archive, the algorithm breeds a new population **P** to produce the next generation's population using the following genetic operators: (1) *Selection* chooses candidate solutions as parents using a tournament selection technique, with the tournament size equal to two, which is the most common setting [34]. (2) *Crossover* creates offspring from the selected parents using a modified version of the one-point crossover. (3) *Mutation* changes the offspring according to a mutation rate and strategy. After completing the evolution process, the algorithm returns the latest archive $\mathbf{P}_\alpha$ that contains the best found Pareto front. We describe our crossover and mutation approaches in detail.

Crossover. A crossover operator produces offspring from two parent solutions by inheriting their characteristics. Our crossover operator, named SWEAKCrossover, modifies the standard one-point crossover operator [54] that selects a random crossover point among all genes and swaps them between parent solutions based on the crossover point. However, in our context, as the size of two parents can differ, such random selection may produce invalid offspring. To prevent it, SWEAKCrossover selects a crossover point among the context switching times, i.e., λ_s, λ_x, and λ_p, or the first arrivals of the aperiodic tasks in Γ. As the size of Γ and context switching times are fixed for all solutions, SWEAKCrossover can crossover two solutions with different sizes.

Fig. 5 shows an example operation of SWEAKCrossover using a system with three aperiodic tasks, τ_1, τ_2, and τ_3. Let two parent solutions I_p and I_q be as follows: $I_p = (0.007, 0.011, 0.001, (\tau_1, 5), \ldots, (\tau_2, 10), \ldots, (\tau_3, 6), (\tau_3, 15))$ and $I_q = (0.008, 0.010, 0.001, (\tau_1, 4), \ldots, (\tau_2, 8), \ldots, (\tau_3, 4), \ldots, (\tau_3, 20))$, where (τ_i, t) states that task τ_i arrives at time t. Given the two parents I_p and I_q, SWEAKCrossover randomly selects a point— the first arrival of τ_2 in this example—and then it swaps the context switching times and all the arrivals of τ_1 between I_p and I_q. As shown in Fig. 5, SWEAKCrossover then generates the offspring I'_p and I'_q as follows: $I'_p = (0.007, 0.011, 0.001, (\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 4), \ldots, (\tau_3, 20))$ and $I'_q = (0.008, 0.010, 0.001, (\tau_1, 4), \ldots, (\tau_2, 10), \ldots, (\tau_3, 6), (\tau_3, 15))$. The shaded (resp. unshaded) cells in Fig. 5 indicate which context switching times and task arrivals in child I'_q (resp. I'_p) come from which parent.

Mutation. SWEAK uses a heuristic mutation algorithm called SWEAKMutation. For a solution I, SWEAKMutation mutates the context switching times or the kth task arrival time $a_{i,k}$ of an aperiodic task τ_i with a mutation probability. Regarding the context switching times, SWEAKMutation chooses a new time value from the range of each context switching time, i.e., startup λ_s, exit λ_x, and IPI λ_p

times. Regarding arrivals of an aperiodic task τ_i, SWEAKMutation chooses a new arrival time value $a_{i,k}$ based on the $[T_i^{min}, T_i^{max}]$ inter-arrival time range of τ_i. If a mutation of the kth arrival time of τ_i does not affect the validity of the k+1th arrival time, the mutation operation ends. Specifically, let $a^*_{i,k}$ be a mutated value of $a_{i,k}$. In case $a_{i,k+1} \in [a^*_{i,k} + T_i^{min}, a^*_{i,k} + T_i^{max}]$, SWEAKMutation returns the mutated I solution.

After mutating the kth arrival time $a_{i,k}$ of a task τ_i in a solution I, if the k+1th arrival becomes invalid, SWEAKMutation corrects the remaining arrivals of τ_i. We denote by $a^*_{i,k}$ the mutated kth arrival time of τ_i. For all the arrivals of τ_i after $a^*_{i,k}$, SWEAKMutation first updates their original arrival time values by adding the difference $a^*_{i,k} - a_{i,k}$. Let $\mathbb{T} = [0, \mathbf{t}]$ be the scheduling period. SWEAKMutation then removes some arrivals of τ_i if they are mutated to arrive after $\mathbf{t}$ or adds new arrivals of τ_i while ensuring that all tasks arrive within $\mathbb{T}$.

Given the offspring presented in Fig. 5, SWEAKMutation, for example, mutates a child solution $I'_q = (0.008, 0.010, 0.001, (\tau_1, 4), (\tau_1, 8), (\tau_1, 16), \ldots, (\tau_3, 15))$. Let $[T_1^{min}, T_1^{max}] = [2, 8]$ be the inter-arrival time range of task τ_1, let $\mathbb{T} = [0, 22)$ be the time period during which APSSimulator receives task arrivals, and let us assume SWEAKMutation selects the second arrival of task τ_1, i.e., $(\tau_1, 8)$ in Fig. 5, to mutate. Based on the inter-arrival time range of τ_1, SWEAKMutation randomly chooses a new arrival time, e.g., 6, for the second arrival of τ_1. The third arrival $(\tau_1, 16)$ of τ_1 then becomes invalid due to the mutated second arrival $(\tau_1, 6)$, i.e., τ_1 cannot arrive at time 16 because $16 \notin [6 + 2, 6 + 8]$, where $[T_1^{min}, T_1^{max}] = [2, 8]$. According to the correction procedure described above, the third arrival of τ_1 is modified to $(\tau_1, 14)$ as $14 = 16 + (6 - 8)$, where 16, 6, and 8 are, respectively, the original third arrival time of τ_1, the mutated second arrival time of τ_1, and the original second arrival time of τ_1. As APSSimulator can receive new arrivals of τ_1 after time 14, SWEAKMutation may add new arrivals of τ_1 based on its inter-arrival time range.

Note that for a system that consists of only periodic tasks, SWEAK will search for effective test cases by varying context-switching times without changing sequences of task arrivals since periodic tasks will follow the same arrival patterns (see Section 3).

4.2 Simulation

The objective of the simulation step is to produce schedule scenarios and a labeled dataset (training dataset). SWEAK uses a scheduling simulation technique to produce schedule scenarios since such a simulation technique can generate a large number of schedule scenarios at a lower cost compared to the cost required to run an actual system. Further, simulation enables analyzing the tasks based on task descriptions at early design stages when their actual code is not yet available. Hence, simulation techniques have been used in many prior studies [14, 47, 48, 50, 57]. Based on simulation results, we generate a labeled data set for the learning step (described in Section 4.3).

APSSimulator. A schedule simulator, named APSSimulator, simulates the behavior of APS according to the characteristics described in Section 2. We note that APSSimulator is our extension of the industrial APS simulator provided by BlackBerry. Our extension is mainly about applying the adapter pattern [33] to integrate the industrial APS simulator into SWEAK. Hence, below, we describe the interfaces (i.e., inputs and outputs) of APSSimulator, which need to be considered when adapting SWEAK to integrate with another scheduling simulator, which simulates, for example, different APS policies [2, 17, 56]. For more details about APS developed by BlackBerry, we refer readers to the APS user's guide [15]. As input, APSSimulator takes a feasible solution I, which contains sequences A of task arrivals for a set Γ of tasks, context switching times (startup λ_s, exit λ_x, IPI λ_p), and a set W of WCET values of the tasks. For a sequences A of task arrivals, APSSimulator calculates when each task arrival will be completed given the context switching times in a solution I and a set W of WCET values, as well as APS configurations, e.g., the time window for partitioning

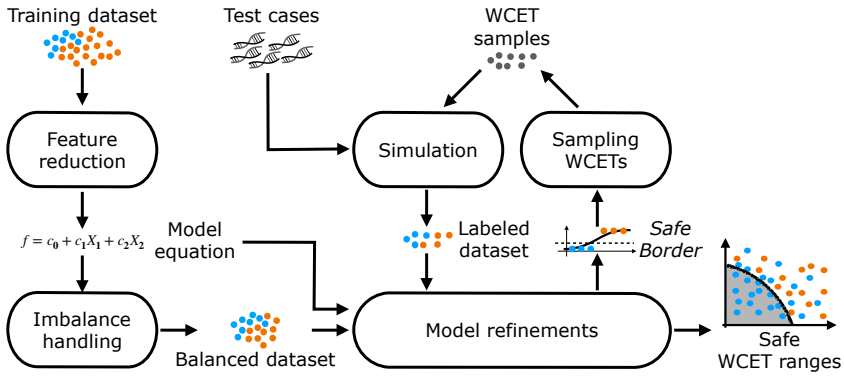


Fig. 6. An overview of the learning step.

and the timeslice for Round-Robin. We set the APS configuration values following the guidelines provided by BlackBerry. As output, a simulation result is encoded into a schedule scenario S (see Section 3).

Generating a labeled dataset. SWEAK requires a labeled dataset as it uses a supervised learning technique [62] to infer a model that predicts safe WCET ranges. Importantly, engineers want to have a certain level of confidence about our prediction results, i.e., safe WCET ranges. To this end, SWEAK applies logistic regression to enable a probabilistic interpretation of the prediction results. In our context, a prediction model, inferred from a given labeled dataset, captures the relationship between tasks' WCET values and the schedulability of the tasks. The detailed learning step is explained in Section 4.3.

A labeled dataset, denoted by $\vec{L}$, is a list of tuples (W, ℓ), where W is a set of WCET values, and ℓ is the label indicating the schedulability of a schedule scenario resulting from W. SWEAK generates a tuple (W, ℓ) for each APSSimulator run. For example, when SWEAK evaluates a feasible solution I during search (Section 4.1), it appends ns tuples (W, ℓ) to the labeled dataset $\vec{L}$. To evaluate a solution I, SWEAK runs APSSimulator ns times with the sampled sets of WCET values, i.e., $\{W_1, W_2, ...W_{ns}\}$. Each set W_h consists of a set of tuples (τ_i, C_i), where C_i is a randomly selected WCET value within the range $[C_i^{min}, C_i^{max}]$ of $\tau_i \in \Gamma$. Given a feasible solution I and a set W_h of WCET values, APSSimulator produces a schedule scenario S_h. SWEAK then labels ℓ as *safe* when the schedule scenario S_h satisfies all the deadline constraints of the target tasks in Γ^{δ}; otherwise, it labels ℓ as *unsafe*. Since schedule scenarios vary across test cases, the labeled dataset $\vec{L}$ can contain tuples that have different labels for the same set W of WCET values.

4.3 Learning logistic regression model

The objective of the learning step is to estimate safe ranges of WCET values under which target tasks are likely to be schedulable. To achieve the objective, SWEAK builds a model to predict safe WCET ranges using logistic regression [45]. This technique provides a probabilistic interpretation and enables trade-off analysis when making implementation decisions about safe WCET ranges. Fig. 6 shows the overall process of the learning step. We note that SWEAK's learning step adapts the learning step of our previous work [50] to account for weakly hard deadline constraints and an industrial simulator, i.e., APSSimulator. We describe the learning step of SWEAK in the following order: feature reduction, imbalance handling, model refinements (including sampling and simulation), and selecting WCET ranges.

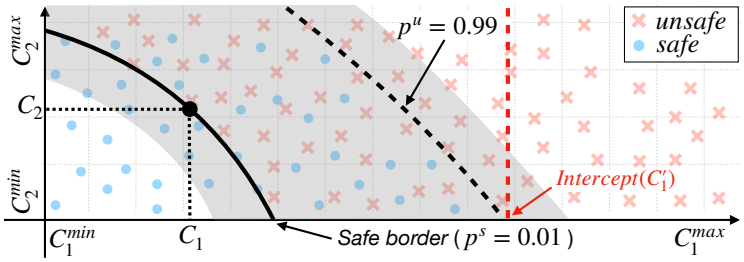


Fig. 7. A logistic regression model in the WCET space of two tasks τ_1 and τ_2.

Feature reduction. Given the training data $\vec{L}$ obtained from the search step, the feature reduction procedure generates an equation f for logistic regression. Logistic regression builds a prediction model by inferring coefficients of a given equation f. The equation f is formulated with the WCET variables of the tasks in Γ recorded in our dataset $\vec{L}$. Some WCET variables have significant effects in predicting whether the label is *safe* or *unsafe*, while other variables do not. Therefore, eliminating insignificant variables is needed to reduce computational complexity and increase model accuracy.

SWEAK applies a feature reduction technique based on random forest that has widely been used for dimensionality reduction [43, 59]. Given the labeled dataset $\vec{L}$, random forest builds a large number of decision trees to predict a label, i.e., either *safe* or *unsafe* in our case, using a randomly selected subset of WCET variables. The technique then derives the importance of each variable based on Gini impurity [19]. SWEAK selects a set V of important variables that are above a particular threshold. Note that we describe the parameter values for our feature reduction in Section 5.5. Given important variables in V, SWEAK formulates an equation f for logistic regression using a second-order polynomial response surface model (RSM) [46] as follows:

$$\log \frac{p}{1-p} = c_0 + \sum_{i=1}^{|V|} c_i v_i + \sum_{i=1}^{|V|} c_{ii} v_i^2 + \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} c_{ij} v_i v_j$$

where $v_i, v_j \in V$, p is the probability of violating deadline constraints, and c_0, c_i, c_{ii}, and c_{ij} are the coefficients that will be inferred by logistic regression. Hence, the probability p of violating deadline constraints is defined as follows:

$$p = \frac{1}{1 + e^{-(c_0 + \sum_{i=1}^{|V|} c_i v_i + \sum_{i=1}^{|V|} c_{ii} v_i^2 + \sum_{i=1}^{|V|-1} \sum_{j=i+1}^{|V|} c_{ij} v_i v_j)}}$$

In addition, SWEAK applies stepwise AIC (Akaike Information Criterion) [76] to the equation f to eliminate terms that do not significantly help predict the label. This enables logistic regression of SWEAK to predict only the coefficients of significant explanatory terms in f. Since SWEAK requires building logistic regression models multiple times within a time budget, and these models are computationally expensive, stepwise AIC allows SWEAK to execute more efficiently.

Imbalance handling. The performance of supervised machine learning highly depends on the training dataset $\vec{L}$. As $\vec{L}$ is generated by the search step that aims to find effective test cases (see Section 4.1) with respect to violating deadline constraints, it tends to be imbalanced, containing more sets of WCET values that result in violating deadline constraints. In general, imbalanced data likely lead to unsatisfactory results when relying on supervised machine learning. SWEAK handles the imbalance problem as described below.

SWEAK builds an initial model m from the training dataset $\vec{L}$ and the equation f. Logistic regression estimates a probability of violating deadline constraints for given tasks' WCET values. For example, Fig. 7 shows a model m in the WCET space of two tasks τ_1 and τ_2. The gray area in Fig. 7 represents the model area where the probability of violating deadline constraints is within the range [0.0001, 0.9999]. Given WCET ranges, a border can be defined by selecting a probability of violating deadline constraints dividing *safe* and *unsafe* areas. SWEAK automatically selects a probability p^u that maximizes the *unsafe* area, while ensuring that all the data instances in the *unsafe* area are classified as *unsafe*, i.e., no false negative (see the area above the border indicated by $p^u = 0.99$ in Fig. 7). SWEAK then calculates reduced WCET ranges $[C_i^{min}, C_i']$, where C_i' is the intercept between the WCET axis of τ_i and the WCET border determined by the probability p^u (see the red dashed line in Fig. 7). The more balanced dataset $\vec{L}^b$ is produced by pruning the data instances outside the reduced WCET ranges. Note that C_i' is equal to C_i^{max} when there is no intercept for a task τ_i.

Model refinements. Given the balanced dataset $\vec{L}^b$ and the equation f, SWEAK builds a logistic regression model m. SWEAK then finds a probability p^s that maximizes the *safe* area, while ensuring that all the data instances within the *safe* area are classified as *safe* with no false positive. Note that m and p^s determine a *safe border* that distinguishes safe and unsafe areas (see the solid line in Fig. 7). More precisely, a safe border is defined by the equation $1/(1 + e^{-f_m}) = p^s$, where f_m denotes the function f with the coefficients' values determined by m (see the RSM coefficients described earlier). The safe area defined by $1/(1 + e^{-f_m}) < p^s$ contains only safe WCETs, while for any probability $p^o > p^s$, the area defined by $1/(1 + e^{-f_m}) < p^o$ contains both safe and unsafe WCETs. To improve the safe border, SWEAK refines it using a distance-based sampling method that adds more WCET samples around the safe border (described in our previous work [50]). The sampled WCET values are evaluated and labeled by the simulation step using the test cases obtained from the search step. These simulation results are included into a new labeled dataset $\vec{L}^{new}$. SWEAK then rebuilds the safe border after merging $\vec{L}^b$ with $\vec{L}^{new}$ using logistic regression. This refinement is repeated until either reaching the specified number of refinements (i.e., assigned analysis budget) or reaching an acceptable level of precision of the safe border using a standard k-fold cross-validation [74]. In the cross-validation process, SWEAK first partitions the merged dataset into k equal-size folds. SWEAK builds and evaluates logistic regression models k times. Each time, SWEAK uses a different fold as the test dataset and the remaining $k - 1$ folds as the training dataset. From the k evaluations, SWEAK computes the accuracy of the safe border determined by a logistic regression model m and a probability p^s.

Selecting WCET ranges. Given a safe border, safe WCET ranges are then determined by selecting one point on that border. A safe border represents a set of points that represents the upper bounds of safe WCET ranges. Engineers thus can find safe WCET ranges by choosing one appropriate point on a safe border depending on their system requirements. For example, if a black dot on the safe border in Fig. 7 is the selected point, i.e., $[C_1, C_2]$, the safe WCET ranges become $[C_1^{min}, C_1]$ and $[C_2^{min}, C_2]$. However, engineers may not have proper contextual information to select a point at early design stages. Hence, SWEAK suggests a point, named *best-size point*, on a safe border that maximizes the volume of the WCET ranges. The best-size point with the largest volume provides engineers with more relaxed objectives for tasks' execution times when no domain-specific information can guide such decision. In addition, the inferred safe border provides engineers with the ability to select another point through trade-off analysis between tasks' WCET values.

5 EVALUATION

In this section, we evaluate SWEAK by answering three research questions below. We do so by applying SWEAK to an industrial study subject from the satellite domain and several synthetic study subjects. All experiment results can be found online [49].

5.1 Research questions

RQ1. (baseline comparison): *How does SWEAK fare against a baseline relying on random search?* In general, comparing a search-based approach against a baseline, i.e., random search, is important to determine if the search problem indeed requires sophisticated search algorithms [7, 40]. In addition, SWEAK is the first attempt to automatically estimate probabilistic safe WCET ranges for weakly hard real-time systems. Hence, we compare SWEAK and a baseline built on random search to see if SWEAK can infer significantly better WCET options than the baseline with respect to their confidence levels and best-size points (defined in Section 4.3). Note that due to the complexity introduced by uncertainties in task arrivals, context switching, adaptive partitioning, and multiple cores, finding optimal solutions in reasonable time is infeasible. Hence, the baseline approach built on random search serves as our best alternative solution.

RQ2. (probabilistic interpretation): *Can we rely on predicted probabilities from logistic regression?* During the design stage, engineers tend to be conservative when determining safe WCET ranges and probabilities of violating deadline constraints as these determine objectives for tasks' implementations. Recall from Section 4.3 that SWEAK employs logistic regression to infer such probabilities. We investigate the probabilities predicted by SWEAK by comparing them with those obtained from a large number of simulations with different WCET values within the estimated WCET ranges. Given the same WCET ranges, our conjecture is that SWEAK infers higher or similar probabilities of violating deadline constraints compared to simulation-based probabilities, as SWEAK relies on fine-tuning of the logistic regression step.

RQ3. (scalability): *Can SWEAK find safe WCET ranges for large-scale systems within a practical time budget?* It is challenging to estimate acceptable WCET ranges for large-scale systems because of complex task interactions caused by combinations of arrival sequences, priorities, context switching times, and WCET values. To assess the scalability of SWEAK in terms of execution time, we use a large number of synthetic systems that are generated with various characteristics.

5.2 Synthetic systems

A synthetic system is an artificially generated system accounting for the characteristics of real-time tasks that reflect actual systems in the real world. Synthetic systems are used in many real-time system studies [28–30, 36, 69, 77] to evaluate approaches while being able to fully control and vary systems' characteristics, thus assessing their impact on results. Algorithm 2 describes a procedure for generating a synthetic system by varying the key task parameters (lines 1-13). Note that the algorithm is developed based on our prior approach for evaluating SAFE and the guidelines provided by BlackBerry to account for weakly hard deadline constraints and adaptive partitions. Briefly, the algorithm synthesizes a set of periodic tasks (lines 18-24) and sets some tasks to have weakly hard deadline constraints (lines 25-26). The algorithm then modifies the system as follows: (1) converting some tasks to aperiodic tasks (lines 27-28), (2) transforming some tasks' WCET values into WCET ranges (lines 29-31), and (3) configuring partitions and assigning tasks to each partition (lines 32-33).

As shown in line 18 of Algorithm 2, the algorithm first creates a set $\mathbf{U}$ of task utilization values using the UUniFast-Discard algorithm [28], which is devised to generate an unbiased distribution of task utilization values. The UUniFast-Discard algorithm takes as input the number n of tasks to

Algorithm 2: An algorithm for generating synthetic systems including weakly hard real-time tasks and APS partitions.

```
1  Input n: number of real-time tasks
2  Input u^t: target utilization of the system
3  Input T^min: minimum task period
4  Input T^max: maximum task period
5  Input g: granularity of task periods
6  Input θ: maximum offset value
7  Input γ: ratio of aperiodic tasks
8  Input μ: range factor to determine inter-arrival times
9  Input ω: number of tasks having WCET ranges
10 Input λ: range factor to determine WCET ranges
11 Input ρ: number of partitions
12 Input (m, K): deadline constraint
13 Input nw: number of weakly hard real-time tasks
14 Output Γ: set of tasks
15
16 Γ ← {}, C ← {}
17 // synthesize a set of periodic tasks
18 U ← UUniFast_discard(n, u^t) // task utilizations
19 T ← generate_task_set(n, T^min, T^max, g) //task periods
20 // determine WCETs
21 for each i ∈ [1, n] do
22     C ← C ∪ {U_i·T_i}, where U_i ∈ U and T_i ∈ T
23 end for
24 Γ ← generate_task_periods(T, C, θ, g)
25 // select weakly hard real-time tasks
26 Γ ← set_deadline_constraints(Γ, nw, (m, K))
27 // convert some tasks to aperiodic tasks
28 Γ ← convert_to_aperiodic_tasks(Γ, γ, μ)
29 // convert some WCET values to WCET ranges
30 // default argument λ = undefined
31 Γ ← convert_to_WCET_ranges(Γ, ω, λ)
32 // assign partitions and partition budgets
33 Γ ← assign_partitions(Γ, ρ)
34 return Γ
```

be synthesized and a target utilization value u^t of the system. It then outputs n utilization values, $\{U_1, \ldots, U_n\}$, where $0 < U_i < 1$ for all U_i and $\sum_{i=1}^{n} U_i = u^t$. The maximum value of target utilization u^t relies on the number of processing cores, i.e., the maximum target utilization is equal to the number of processing cores. For example, if a system uses two processing cores, the maximum value of u^t is 2.

On line 19, the algorithm generates n task periods, $T_1 \ldots T_n$ according to a log-uniform distribution within a range $[T^{min}, T^{max}]$, i.e., $log\ T_i$ follows a uniform distribution. For example, when a period range $[T^{min}, T^{max}]$ is [10ms, 1000ms], the algorithm generates approximately an equal number of tasks in the period ranges [10ms, 100ms] and [100ms, 1000ms]. The parameter g is used to determine the granularity of period values to be multiples of g. Lines 21-23 of Algorithm 2 describe how the algorithm synthesizes tasks' WCET values $\mathbf{C}$. Specifically, for each task τ_i, the algorithm computes its WCET value C_i as $U_i \cdot T_i$.

Given the task periods **T** and the WCET values **C**, line 24 of Algorithm 2 synthesizes a set Γ of periodic tasks with their offsets, priorities, and deadlines. Recall from Section 3 that a periodic task τ_i is characterized by a period T_i, a WCET C_i, an offset O_i, a priority P_i, and a deadline D_i. A task offset O_i is randomly selected from an input range $[0, \theta]$ of offset values. The algorithm applies a rate-monotonic scheduling policy [52] to assign task priorities, in which tasks that have longer periods are given lower priorities. This policy assumes that task deadlines are equal to their periods.

Given the system Γ, line 26 assigns the (m, K)-constraint (defined in Section 3) to nw tasks to make them weakly hard real-time tasks. Note that real-time tasks in a system can have different deadline constraints, supported by SWEAK. However, to investigate the impact of different (m, K)-constraints in a controlled setting, we set the nw tasks to be subjected to the same deadline constraint (see Section 5.4). We select the nw tasks from the lowest priority tasks as they have higher chances of missing deadlines compared to higher priority tasks.

Line 28 selects some periodic tasks and converts them into aperiodic tasks according to the ratio γ of aperiodic tasks. The algorithm then uses a range factor μ to determine the minimum and maximum inter-arrival times of the aperiodic tasks. Specifically, for a task τ_i to be converted, the algorithm computes a range $[T_i^{min}, T_i^{max}]$ of inter-arrival times as $[T_i^{min}, T_i^{max}] = [T_i \times (1 - \mu), T_i \times (1 + \mu)]$, where $\mu \in (0, 1)$. For example, if $\mu = 0.45$ and $T_i = 50$ for a task τ_i to be converted, $[T_i^{min}, T_i^{max}] = [27.5, 72.5]$. For the converted aperiodic tasks, we set their offsets to 0, i.e., O_i=0 as the offsets are replaced by the tasks' inter-arrival times (see Section 3).

To synthesize tasks' WCET ranges, line 31 randomly selects ω tasks in Γ to convert their WCET point values into WCET ranges. When the range factor λ is defined, the algorithm computes the WCET ranges by applying λ to the selected tasks. More precisely, for a selected task $\tau_i \in \Gamma$, the algorithm computes a WCET range $[C_i^{min}, C_i^{max}]$ as $[C_i^{min}, C_i^{max}] = [C_i \times (1 - \lambda), C_i \times (1 + \lambda)]$, where $0 < \lambda < 1$. When the range factor λ is undefined, the algorithm selects a range factor λ_i for each task $\tau_i \in \Gamma$ from a log-uniform distribution in the range (0, 1). Each WCET range $[C_i^{min}, C_i^{max}]$ for τ_i is then determined according to $[C_i \times (1 - \lambda_i), C_i \times (1 + \lambda_i)]$. For example, if $\lambda_1 = 0.25$ and $C_1 = 10$ for a task τ_1, $[C_1^{min}, C_1^{max}] = [7.5, 12.5]$. This procedure results in a system having a small number of tasks with large WCET ranges and a large number of tasks with small WCET ranges. Note that we discard the invalid cases where the minimum WCET is equal to 0 or the maximum WCET is greater or equal to its deadline, i.e., $C_i^{min} = 0$ or $C_i^{max} >= D_i$.

Regarding APS partitioning, line 33 assigns tasks to partitions. The algorithm creates ρ partitions and assigns evenly distributed partition budgets. For example, when ρ=2, the budget distribution is [50%, 50%]. If ρ=3, the budget distribution is [34%, 33%, 33%]. The algorithm then randomly assigns tasks to partitions. Each partition must have at least one task, and a task can be assigned to only one partition.

5.3 Study subjects

To address RQ1 and RQ2, we use four case study subjects: ESAIL [50] (an industrial real-time system) and three synthetic systems. We note that, due to confidentiality, we were not able to obtain access to the actual weakly hard real-time systems developed by BlackBerry's customers, who use QNX Neutrino with APS. However, the synthetic systems used in our experiments are realistic and representative as they are based on BlackBerry's guidelines and employ the industrial APS policy. Regarding RQ3, we use a large number of synthetic systems generated by controlling parameters in Algorithm 2 (See Section 5.4). The full descriptions of the systems are available online [49]. We further describe the details of the systems below.

ESAIL is a commercial microsatellite developed by LuxSpace that tracks the movements of ships over the entire globe. The ESAIL management system is made up of 12 periodic tasks and 13 aperiodic tasks working on a single core platform with one partition. During the design stages, these tasks were analyzed their WCETs as ranges. Regarding deadline constraints, five aperiodic tasks are considered weakly hard real-time tasks while the other tasks are hard real-time tasks.

To generate three synthetic systems for RQ1 and RQ2, we first create a base system using Algorithm 2. The base system is generated with the following parameter values: (1) the number of tasks $n = 25$, the ratio of aperiodic tasks $\gamma = 0.5$, the range factor to determine inter-arrival times $\mu = 0.25$, and the maximum offset $\theta = 0$. These settings were decided based on the characteristics of ESAIL. (2) the minimum task period $T^{min} = 10$ms, the maximum task period $T^{max} = 1$s, and the granularity $g = 10$ms. These are commonly used in real-time systems [9]. (3) the target utilization $u^t = 0.9$ for a single-core platform. This was decided to ensure the tasks sometimes miss their deadlines [31]. (4) Regarding the number of APS partitions, we set $\rho = 1$. This was decided for the base system to be simple so that it can easily be converted to other synthetic systems. (5) Regarding deadline constraints, we set the 10 lowest priority tasks to be weakly hard real-time tasks (i.e., $nw = 10$). The (m, K)-constraint of these 10 tasks was set to $(0, 10)$, i.e., hard deadline constraint. It will subsequently be changed in our experiments (see Section 5.4) to account for weakly hard deadline constraints, i.e., $m > 0$. (6) For WCET ranges, we set the number ω of tasks with WCET ranges to 25 and the range factor λ for determining WCET ranges to *undefined* (see Algorithm 2). Recall from Section 5.2 that this configuration creates a system with a small number of tasks having large WCET ranges and a large number of tasks having small WCET ranges, aligning with the WCET characteristics of ESAIL.

Given the base system Γ, we synthesize the three systems described below by modifying Γ to account for the characteristics of APS following the guidelines from BlackBerry. These synthetic systems enable us to evaluate SWEAK in different operational settings of APS.

- PARTITION: This system has two APS partitions with 60% and 40% budgets. For efficient scheduling, a partition budget should be enough to execute all tasks in the partition. We thus assign tasks to each partition so that each partition budget is close to the total utilization of the tasks in the partition. In our evaluation, 19 tasks with high priorities in Γ are assigned to the first partition. The remaining six tasks are assigned to the second partition.
- POLICY: This system contains two pairs of tasks with the same priority. To make a pair, we randomly select two tasks from the given system Γ and assign the same priority and scheduling policy to the selected tasks. One pair applies FIFO. The other pair uses Round-Robin.
- MULTICORE: This system works on a two-core platform and assigns core affinities to some tasks. To make this system, we multiply WCET values (as well as the WCET ranges) by two for all tasks in Γ to make the total utilization $\approx$1.8. Recall that the maximum total utilization of a system is equal to the number of cores. We then assign core affinity to tasks using a random selection. For this system, we assign core 1 affinity to eight tasks, core 2 affinity to the other eight tasks, and no core affinity to the remaining nine tasks.

5.4 Experimental design

To answer the three RQs described in Section 5.1, we design three experiments EXP1, EXP2, and EXP3, respectively. We conduct EXP1 and EXP2 with the four case study subjects described in Section 5.3: ESAIL, PARTITION, POLICY, and MULTICORE. For EXP3, we experiment with 600 synthetic systems with different parameter settings. We describe each experiment in detail below.

EXP1. To answer RQ1, we implement a baseline approach (named Baseline) that uses random search without the learning step in SWEAK. Baseline's random search is a variant of the search

step in SWEAK that does not use genetic operators, i.e., selection, crossover, and mutation, to breed offspring (see Section 4.1). Instead, Baseline generates offspring randomly for the next generation and evaluates them with the same multi-objective fitness functions as SWEAK. During search, a labeled dataset $\overrightarrow{L}$ is produced by Baseline, which contains tuples (W, ℓ) where W is a set of tasks' WCET values and ℓ is the label that classifies the simulation result with W as safe or unsafe. Once the labeled dataset $\overrightarrow{L}$ is obtained, Baseline retrieves all tuples from $\overrightarrow{L}$ to select a specific tuple (W_s, ℓ_s) that is safe (i.e., $\ell_x = safe$) and maximizes the volume of the hyperbox defined by W_s. Note that W_s should satisfy the condition that any tuple (W_x, ℓ_x) contained in the hyperbox defined by W_s be safe, i.e, $\ell_x = safe$.

EXP1 compares the results obtained from SWEAK against Baseline. Recall from Section 4.3 that SWEAK suggests safe WCET ranges on the safe border by selecting a best-size point that maximizes its volume of the hyperbox. Since both SWEAK and Baseline return best-size points, the comparison can be done by measuring the best-size volumes. To analyze the relationship between deadline constraints and best sizes, we apply both approaches to the subjects with different deadline constraints (m_i, K_i), where m_i is the number of tolerable deadline misses and K_i is the time window to check the deadline constraint (see Section 3). To do this, we vary m_i from 0 to 4, with a fixed K_i (10) by assuming that all tasks in a subject are subjected to the same deadline constraint; hence, EXP1 uses 4×5 synthetic systems (i.e., ESAIL, PARTITION, POLICY, and MULTICORE with five different deadline constraints). Note that we do not vary K_i because it does not affect the results.

EXP2. To answer RQ2, EXP2 calculates the empirical probability of violating deadline constraints for the safe WCET ranges obtained from SWEAK. To this end, we first randomly sample multiple test cases, defining task arrivals and context switching times, and execution times within the safe WCET ranges obtained from each approach. We then simulate many combinations of the test cases and execution times using APSSimulator and check for the presence of violating deadline constraints in each simulation result. The empirical probability is calculated as the number of simulations that violate deadline constraints over the number of all simulation runs. We simulate 40000 times to compute the empirical probability of safe WCET ranges obtained by SWEAK and Baseline. In addition, we conduct EXP2 by varying the number m_i from 0 to 4 tolerable deadline misses to investigate the impact of deadline constraints.

EXP3. To answer RQ3, we conduct EXP3 to assess the execution time of SWEAK using 600 synthetic systems with different system characteristics using Algorithm 2. EXP3 varies each system parameter value while fixing the other parameters' values so that we can analyze correlations between the execution time of SWEAK and the varying parameter. We generate the synthetic systems by changing the following six parameters: (a) number of tasks, $n \in \{5, 10, \cdots, 50\}$, (b) ratio of aperiodic tasks, $\gamma \in \{0.05, 0.10, \cdots, 0.50\}$, (c) number of WCET ranges, $\omega \in \{1, 2, \cdots, 10\}$, (d) number of processing cores, $\epsilon \in \{1, 2, \cdots, 10\}$, (e) number of APS partitions, $\rho \in \{1, 2, \cdots, 10\}$, and (f) simulation time, $\mathbf{t} \in \{5s, 10s, 15s, \cdots, 50s\}$.

The number of all tasks n, the ratio of aperiodic tasks γ, the number of WCET ranges ω, and the number of processing cores ϵ are selected because they are the main factors when designing real-time systems. The number of APS partitions ρ is selected as it is adjustable by APS. We also include the simulation time $\mathbf{t}$ as the execution time of SWEAK obviously depends on simulation time. We note that the total utilization of a generated synthetic system changes according to the number of processing cores. For example, when the target utilization $u^t = 0.9$ and the number of cores $\epsilon = 2$, the total utilization of the system becomes 1.8 (see Section 5.2).

When varying a parameter's value, to enable controlled experiments, we fix the other parameters' values as follows: (1) We set the number of all tasks $n = 25$, the ratio of aperiodic tasks $\gamma = 0.50$, and the maximum offset $\theta = 0$. We set these values according to our industrial subject, ESAIL.

(2) Regarding the task periods, we set the range $[T^{min}, T^{max}]$ of minimum and maximum periods to [10ms, 1s] with granularity g = 10ms. These values are commonly used in many real-time subjects [9]. (3) We set the range factor to determine inter-arrival times of aperiodic tasks μ = 0.25, the number of WCET ranges ω = 2, the range factor to determine WCET ranges λ = 0.25, and the target utilization per processing core u^t = 0.9. The parameters' values are determined based on our preliminary experiments. They ensure that the executions of the synthetic systems examined in EXP3 can sometimes violate their deadline constraints, i.e., they contain both safe and unsafe data instances (see Section 4.3). (4) We set the number of processing cores ϵ and the number of APS partitions ρ equal to 1. These values are selected to build simple baseline systems. (5) For the simulation time of APSSimulator (see Section 4.2), we assign the minimum simulation time of 5s to guarantee that any aperiodic task arrives at least once and all possible arrivals of periodic tasks can be analyzed during that period. Additionally, with regard to the deadline constraint, we set (m, K) to (2, 10) for the ten lowest priority tasks in each system.

Due to the randomness of SWEAK, we conduct our experiments multiple times, i.e., 50 times for EXP1 and EXP2 and 10 times for each parameter configuration of EXP3. To compare the results, we perform a statistical comparison using the non-parametric Mann-Whitney U-test [55] and Vargha and Delaney's $\hat{A}_{12}$ [67]. The Mann-Whitney U-test is used to compare statistical differences between two independent sample groups. We set the level of significance α = 0.05. Vargha and Delaney's $\hat{A}_{12}$ is a measure of effect size to assess the practical significance of differences between two search algorithms. If $\hat{A}_{12}$ is equal to 0.5, the two algorithms are equivalent. If $\hat{A}_{12}$ is close to 1, the first algorithm is largely superior to the second algorithm.

5.5 Implementation and parameter tuning

We set the following parameter values for running SWEAK and Baseline. For the NSGA-II search in SWEAK, we set the population size to 10, the crossover rate to 0.7, and the mutation rate to 0.2 based on existing guidelines [42]. We set the number of iterations to 1000 since we observed that the fitness values reached a plateau after that in our preliminary experiment. To calculate the fitness values, we ran APSSimulator 20 times for each solution (I in Section 4.1). Based on preliminary experiments, we found that this number was sufficient to compute the fitness values of a solution within a reasonable time period (i.e., less than 1m).

For random search in Baseline, we set the number of iterations to 1500 to ensure that Baseline produces a dataset $\vec{L}$ of the same size as SWEAK. We used the same values as SWEAK for the population size and the number of APSSimulator runs.

To simulate study subjects, we set the simulation time to 60s for the ESAIL subject and 5s for other synthetic subjects. Such values are determined by the following rules: (1) If a system is composed of only periodic tasks, the simulation time is the least common multiple (LCM) of the period values for all tasks [70]. (2) If a system contains aperiodic tasks, the simulation time is determined as the larger value of the following two values: the LCM of the period values of the periodic tasks and the maximum value among the maximum inter-arrival times of the aperiodic tasks. This simulation time allows us to simulate all possible patterns of arrivals of periodic tasks including at least one arrival of aperiodic tasks.

For each run of APSSimulator, we set the time window for partitioning to 100ms, which is the default value of APS. The timeslice for Round-Robin is set to 4ms, and the processor tick interval is 1ms. According to the guidelines from BlackBerry, the timeslice is usually set to 4 times a processor tick interval.

SWEAK has some parameters in the learning step for tuning feature reduction and model refinement. Regarding the former, we employed the random forest algorithm that includes the

following parameters: (1) The tree depth was set to $\sqrt{|F|}$, where $|F|$ is the number of features, following the guidelines [41]. For example, since the ESAIL subject contains 25 features (i.e., WCET ranges), we assigned $\sqrt{|25|}$ to the tree depth of the subject (see Section 4.3). (2) The number of trees was set to 100 as we found that learning more than 100 trees did not provide further benefits for reducing the number of features in our preliminary experiments. Regarding model refinement, we set the number of WCET samples to 100 and the number of model updates to 100. We observed that the precision of the model reaches an acceptable level with these parameters in our preliminary experiments.

We note that all the parameters can be further tuned to improve the performance of SWEAK. However, we were able to clearly and convincingly support our conclusions with the current parameter settings mentioned above; hence, we do not report further experiments on tuning the parameters' values.

All experiments have been performed on nodes in the high-performance computing cluster [68] at the University of Luxembourg. Each run of experiments was executed on a node by assigning 10 cores running at 2.6GHz and 16GB of memory.

5.6 Results

RQ1. Fig. 8 shows the results of EXP1, which compare the volumes of the hyperboxes defined by the safe WCET ranges computed by SWEAK and Baseline. The comparisons are carried out with five different numbers of tolerable deadline misses, 0 to 4, in deadline constraints of the following four study subjects: ESAIL (Fig. 8a), PARTITION (Fig. 8b), POLICY (Fig. 8c), and MULTICORE (Fig. 8d). Each boxplot in the figures shows a distribution (25%-50%-75% quartiles) obtained from 50 runs of SWEAK and Baseline. The figures also report p-values and $\hat{A}_{12}$ values from comparing 50 runs of SWEAK and Baseline. Note that the unit of the volumes is ms^{25}, where the number of tasks with WCET ranges $\omega = 25$. Since the minimum time unit in our experiments is 0.1ms, the minimum volume of the study subjects is $1 \times 10^{-25}\text{ms}^{25}$.

As shown in Fig. 8, SWEAK produces larger volumes of hyperboxes compared to Baseline across all the subjects. Note that a larger hyperbox volume provides greater flexibility in selecting appropriate WCET values, as such a hyperbox has wider WCET ranges. Regarding deadline constraints, for both SWEAK and Baseline, the larger m, the larger the volume of the hyperbox for the following subjects: PARTITION, POLICY, and MULTIFORE. In particular, when $m = 1$, the hyperbox volume becomes much larger than that obtained when $m = 0$. The increase in volume is smaller when m further increases to 2, 3, and 4. This trend implies that when a deadline miss occurs, the subjects are likely to have more consecutive deadline misses. Unlike the three subjects discussed above, the hyperbox volumes of ESAIL are similar when the system is subjected to weakly hard deadline constraints ($m \geq 1$), which are significantly larger than the hyperbox volumes when the hard deadline constraint ($m = 0$) is applied. This trend is caused by the characteristics of ESAIL, which works on a single core platform with one partition (see Section 5.3). Hence, the lowest priority task in ESAIL can easily starve when a deadline violation occurs due to high priority tasks having long execution times, which continuously occupy the processing core of ESAIL. Regarding the other study subjects, the results show that the different operational settings of APS, i.e., adaptive partitions (Fig. 8b), multiple policies (Fig. 8c), and multiple cores (Fig. 8d), could alleviate the problem of starvation. Across all the subjects and deadline constraints, the experiment results obtained from SWEAK are significantly superior to those obtained from Baseline (i.e., p-value < 0.05 and large effect sizes of $\hat{A}_{12} \approx 1.0$) with regard to the best-sizes of safe WCET ranges. The average execution times of SWEAK and Baseline are 9.37h and 7.55h, respectively, when both methods produce datasets of the same size.

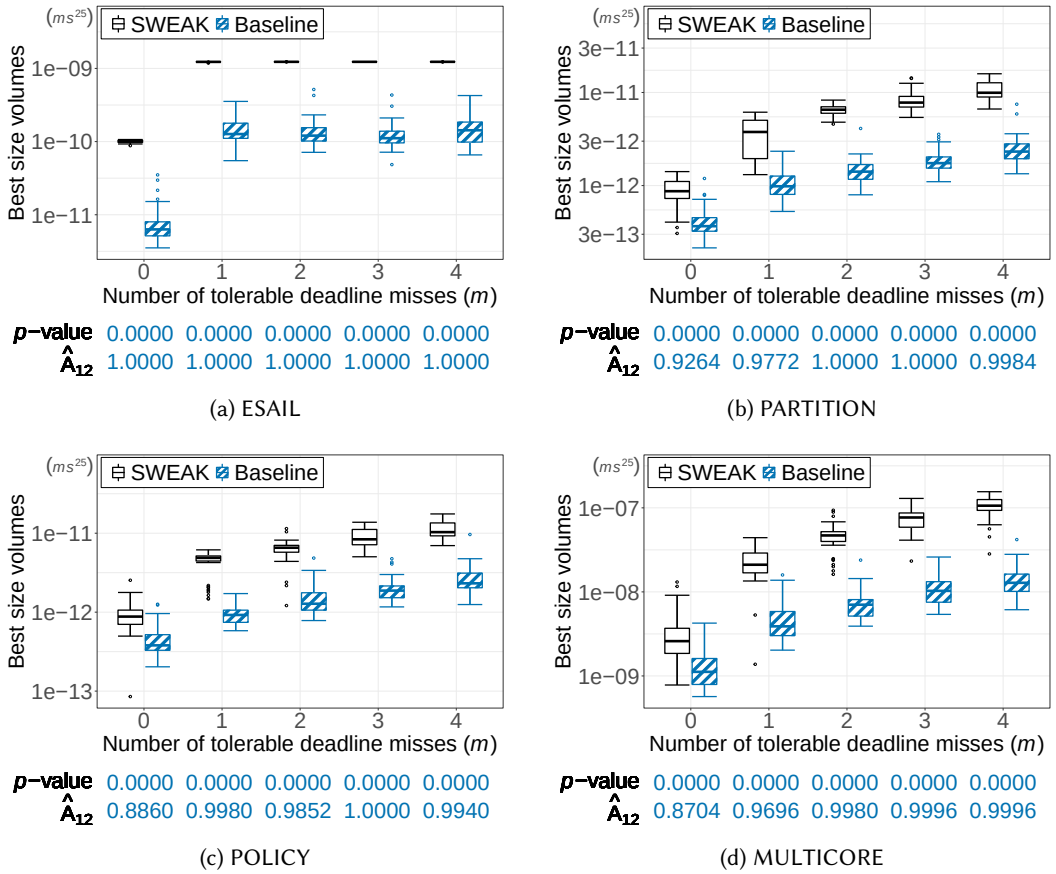


Fig. 8. Distributions of the hyperboxes' volumes that are defined by the safe WCET ranges obtained from SWEAK and Baseline for (a) ESAIL, (b) PARTITION, (c) POLICY, and (d) MULTICORE. The boxplots (25%-50%-75%) show the hyperboxes' volumes obtained from 50 runs of SWEAK and Baseline.

In addition, EXP1 evaluates the best-size WCET ranges obtained from 50 runs of SWEAK and Baseline using 40000 simulation runs by varying test cases (i.e., task arrivals and context switching times) and WCET values within the best-size WCET ranges. Table 1 shows the (maximum, median, minimum, and average) number of simulation runs (out of 40000 runs) in which any violation of deadline constraints occurred in the following subjects: ESAIL (Table 1a), PARTITION (Table 1b), POLICY (Table 1c), and MULTICORE (Table 1d). Once again, we vary the number of tolerable deadline misses m in the experiments from 0 to 4. The p-values and $\hat{A}_{12}$ values report the differences between the results obtained from 50 runs of both approaches.

As shown in Table 1, SWEAK is significantly better (p-values are less than 0.05) with respect to violating deadline constraints than Baseline across all values of m for the PARTITION, POLICY, and MULTICORE subjects. The $\hat{A}_{12}$ values are also much lower than 0.5. Specifically, SWEAK shows smaller variation than Baseline (i.e., the differences between maximum and minimum values) in the number of simulation runs that violate deadline constraints. Regarding the ESAIL subject, both SWEAK and Baseline show similar results when m = 1, 2, 3, and 4 due to the starvation of the lowest priority task in ESAIL as discussed above. However, Baseline has a larger variation in the

Table 1. Summary of the numbers of simulation runs that violate any deadline constraints in (a) ESAIL (b) PARTITION, (c) POLICY, and (d) MULTICORE. EXP1 ran 40000 simulation runs for each subject with different test cases and WCET values within the best-size WCET ranges obtained from SWEAK and Baseline. Each table shows the max, median, min, and average simulation runs under different deadline constraints, i.e., the number of tolerable deadline misses m = 0, 1, 2, 3, or 4.

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	19.00	0.00	0.00	0.00	0.00
	Median	2.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	3.80	0.00	0.00	0.00	0.00
Baseline	Max	574.00	0.00	0.00	0.00	0.00
	Median	0.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	19.24	0.00	0.00	0.00	0.00
	p-value	0.1269	1.0000	1.0000	1.0000	1.0000
	$\hat{A}_{12}$	0.5830	0.5000	0.5000	0.5000	0.5000

(a) ESAIL

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	17.00	13.00	2.00	2.00	3.00
	Median	0.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	2.60	0.90	0.06	0.08	0.10
Baseline	Max	409.00	333.00	128.00	180.00	345.00
	Median	76.00	10.50	7.00	2.00	1.50
	Min	0.00	0.00	0.00	0.00	0.00
	Average	111.36	42.00	15.68	13.68	15.94
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.0196	0.1604	0.1280	0.1840	0.2378

(b) PARTITION

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	29.00	20.00	1.00	1.00	0.00
	Median	1.50	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	3.48	1.14	0.04	0.08	0.00
Baseline	Max	705.00	199.00	137.00	46.00	216.00
	Median	56.00	7.00	2.50	1.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	125.70	18.46	15.44	7.18	15.16
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.1024	0.2130	0.1988	0.2316	0.3000

(c) POLICY

		Number of tolerable deadline misses (m)				
		0	1	2	3	4
SWEAK	Max	146.00	73.00	7.00	21.00	5.00
	Median	27.00	0.00	0.00	0.00	0.00
	Min	0.00	0.00	0.00	0.00	0.00
	Average	35.60	2.68	0.34	1.08	0.20
Baseline	Max	1495.00	848.00	339.00	444.00	220.00
	Median	192.50	13.50	12.00	6.50	2.50
	Min	0.00	0.00	0.00	0.00	0.00
	Average	266.60	58.04	52.46	25.98	27.66
	p-value	0.0000	0.0000	0.0000	0.0000	0.0000
	$\hat{A}_{12}$	0.1320	0.0958	0.1556	0.2026	0.1972

(d) MULTICORE

number of simulation runs that violate deadline constraints when ESAIL is subjected to the hard deadline constraint (i.e., m = 0). The results indicate that the best-size WCET ranges computed by SWEAK are more reliable in terms of violating deadline constraints than those computed by Baseline.

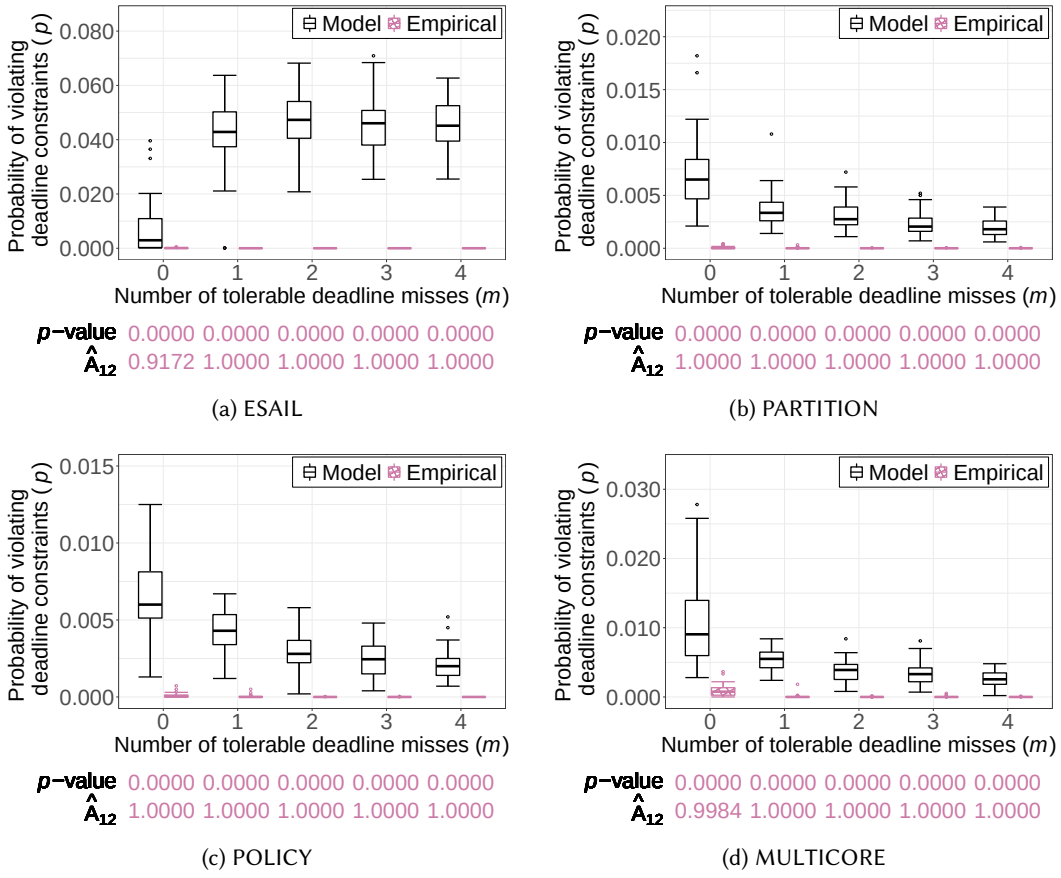


Fig. 9. Distributions of empirical probabilities and model probabilities across different numbers of tolerable deadline misses m in (a) ESAIL, (b) PARTITION, (c) POLICY, and (d) MULTICORE. The boxplots (25%-50%-75%) show distributions of probabilities for the best-size WCET ranges obtained from 50 runs of SWEAK.

The answer to **RQ1** *is that* SWEAK significantly outperforms the baseline approach with respect to maximizing the hyperbox volume of the best-size WCET ranges under hard and weakly hard deadline constraints. The best-size WCET ranges obtained by SWEAK have a significantly smaller chance of violating deadline constraints than the baseline approach. SWEAK takes on average 9.37h to compute the safe WCET ranges, while the baseline takes on average 7.53h, which indicates that SWEAK is acceptable for use in practice as an offline analysis tool.

RQ2. Fig. 9 depicts the results of EXP2 for all subjects: ESAIL (Fig. 9a), PARTITION (Fig. 9b), POLICY (Fig. 9c), and MULTICORE (Fig. 9d). Each sub-figure compares the model probability (computed by SWEAK's logistic regression) and empirical probability (computed by simulations) of violating deadline constraints by varying the number of tolerable deadline misses m. Each boxplot in the figures shows the distributions (25%-50%-75% quartiles) of model probabilities and empirical probabilities for the best-size WCET ranges obtained from 50 runs of SWEAK. As shown in Fig. 9, the empirical probabilities across all values of m and all the subjects are significantly smaller than the model probabilities. Statistical comparisons show that all the p-values are 0 and all the $\hat{A}_{12}$

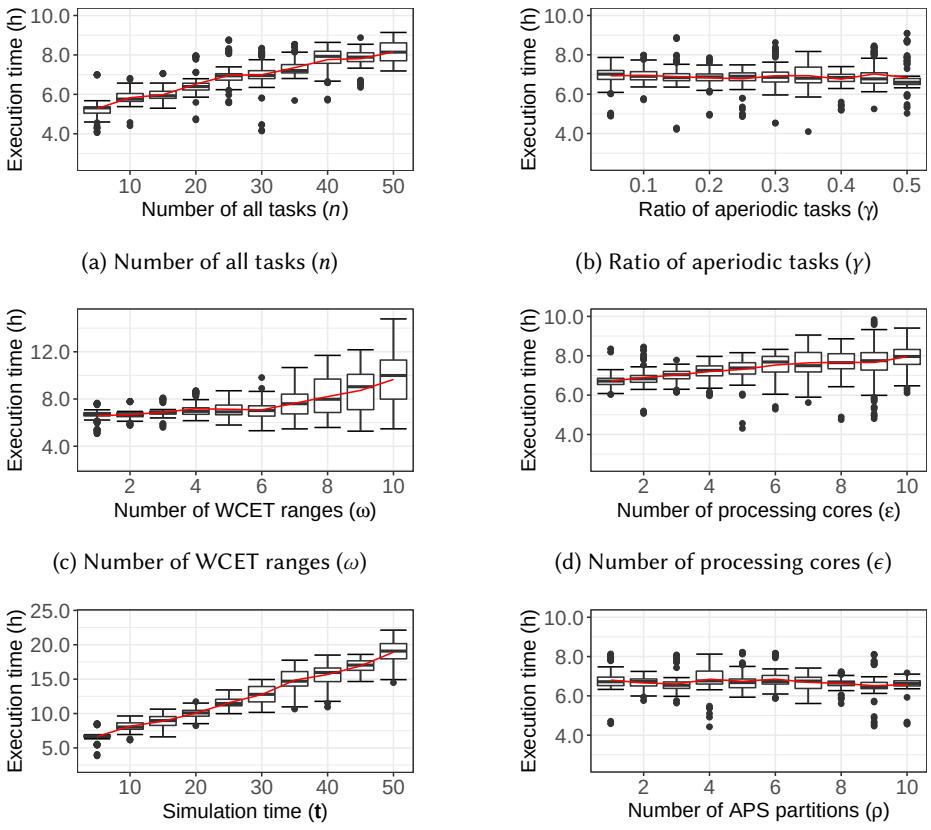
values are approximately 1. Recall from Section 4 that SWEAK infers a logistic regression model with a probability of violating deadline constraints based on the labeled dataset that is generated by evaluating the worst-case task arrivals and context switching times, i.e., test cases. SWEAK thus infers the model that fits the worst-case test cases. However, SWEAK shows higher probabilities than the empirical probabilities, which are computed by running simulations with random test cases and random WCET values within the best-point WCET ranges obtained by SWEAK. Hence, a logistic regression model produced by SWEAK allows engineers to probabilistically interpret the safe WCET ranges in a more conservative manner than evaluating the WCET ranges by simulations. Such results indicate that we can expect the actual probability of violating deadline constraints to be lower than the model probability determined by SWEAK. Note that such conservative interpretations of WCET ranges are desirable in practice.

Regarding the trend for model probabilities over the number m of tolerable deadline misses, ESAIL (Fig 9a) shows similar probability distributions when $m > 0$. The probabilities in these distributions are higher than that obtained when $m = 0$. This trend contrasts with that of the other three subjects (Figs. 9b, 9c, and 9d), where model probabilities decrease with an increasing number of tolerable deadline misses. In ESAIL, this particular trend is caused by its characteristics (Section 5.3), which make it prone to starvation, as described earlier (see the RQ1 results). Hence, when high-priority tasks with long execution times continuously occupy the processing core of ESAIL, low-priority tasks with short execution times will likely encounter consecutive deadline misses due to starvation. However, regarding the other three subjects that operate under the different settings of APS, i.e., adaptive partitions (Fig. 9b), multiple policies (Fig. 9c), and multiple cores (Fig. 9d), we did not observe starvation problems that are likely to lead to consecutive deadline misses.

The answer to **RQ2** *is that* SWEAK estimates higher probabilities of violating deadline constraints for the safe WCET ranges than empirical probabilities computed by simulation-based evaluations for the ranges. SWEAK, therefore, enables conservative probabilistic interpretations of safe WCET ranges.

RQ3. Fig. 10 shows the results of EXP3 that present the distributions (boxplots) of execution times obtained from 10×10 runs of SWEAK, i.e., 10 runs for each synthetic system with the same experimental setting (see Section 5.4). The red solid lines represent the changes in mean values of the execution times while varying the following control parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) number of WCET ranges ω, (d) number of processing cores ϵ, (e) simulation time $\mathbf{t}$, and (f) number of APS partitions ρ. The experiments in EXP3 took 22.1 hours at most, which is acceptable as an offline analysis technique. As shown in Fig. 10a, Fig. 10d, and Fig. 10e, there are positive linear relationships between SWEAK's execution times and the parameters, n, ϵ, and $\mathbf{t}$. Thus, we expect SWEAK to scale well as the number of tasks, the number of processing cores, and the simulation time increase. Further, for parameters such as γ (Fig. 10b) and ρ (Fig. 10f), there is no correlation with execution time.

In contrast, parameter ω (Fig. 10c, number of WCET ranges) is quadratically related with the execution time of SWEAK. Recall from Section 4.3 that, to build the equation used in logistic regression, we leverage the second-order polynomial RSM that contains linear, quadratic, and two-way interaction terms. The total number of terms is determined by the number of WCET ranges ω, which is $1 + \omega + \omega + \binom{\omega}{2}$. The logistic regression algorithm infers the coefficients for each term of RSM and is used during the execution of SWEAK. As the execution time of logistic regression is linearly related to the number of coefficients [45], the execution time of SWEAK is quadratically correlated with the number of WCET ranges. Moreover, as the number of WCET ranges ω increases, the variance of execution time becomes larger, as visible in Fig. 10c. We found



(a) Number of all tasks (n)

(b) Ratio of aperiodic tasks (γ)

(c) Number of WCET ranges (ω)

(d) Number of processing cores (ϵ)

(e) Simulation time (**t**)

(f) Number of APS partitions (ρ)

Fig. 10. Execution times of SWEAK when varying the values of the following parameters: (a) number of all tasks n, (b) ratio of aperiodic tasks γ, (c) number of WCET ranges ω, (d) number of processing cores ϵ, (e) simulation time **t**, and (f) number of APS partitions ρ. Each boxplot (25%-50%-75%) shows the distributions of 100 execution time values measured from 10 runs of SWEAK for 10 synthetic systems with the same configuration. The red line in each figure represents the trend of mean values of the execution times over the parameter values.

that this phenomenon occurs due to feature reduction and stepwise regression in the learning step (see Section 4.3). These techniques output an equation consisting of terms that are considered highly related to the violation of deadline constraints. Depending on the system characteristics, the synthetic systems generated by setting $\omega = 10$ have more diverse equations than those generated with $\omega = 1$. This output affects the execution time for sampling WCET values (distance-based) and building logistic regression models.

The answer to **RQ3** *is that* SWEAK's execution time is linearly related to the number of tasks, the number of processing cores, and the simulation time. However, execution time has a quadratic correlation with the number of tasks whose WCETs are defined as ranges. Overall, SWEAK is applicable in practice as an offline analysis technique since it took at most 22.1h in our experiments, which is generally acceptable.

Usefulness of SWEAK from the perspective of practitioners. To understand the practical usefulness of SWEAK, we discussed it with two practitioners at Blackberry with whom we are closely collaborating. The feedback we received is as follows: (1) practitioners perceived that the test cases, i.e., worst-case task arrivals and context switching times, produced by SWEAK help them conduct further analysis of their systems' schedulability, (2) practitioners agreed that SWEAK enables trade-off analysis by using a logistic regression model, and (3) practitioners perceived that SWEAK can be applied to their customers' systems since it uses an industry-strength simulator (APS simulator) and supports weakly hard real-time tasks.

As a company that develops a real-time operating system, Blackberry has a long-term goal of providing its customers with a schedulability analysis tool for systems with heterogeneous characteristics such as multiple policies, partitions, and weakly hard deadline constraints. SWEAK is a practical candidate solution as it is not only an industrial simulation-based approach but it also addresses both tasks with hard and weakly hard deadline constraints. Although we have not conducted user studies, given the positive feedback from Blackberry, we believe SWEAK can be practically applicable and is worthy of further research.

5.7 Threats to validity

Internal validity. An internal validity threat is the randomness of SWEAK. To mitigate this threat, we compared SWEAK against a baseline method under identical parameter settings, and ran both approaches 50 times for each experiment setting. We then performed statistical comparisons using the Mann-Whitney U-test and Vargha and Delaney's $\hat{A}_{12}$.

Another threat to internal validity is related to the configuration of the experiments. As SWEAK uses a multi-objective search algorithm, there are many parameters that need to be optimized to find best solutions. In our experiments, we configured the parameters' values based on guidelines [42] and preliminary experiments (see Section 5.5). Even though, to improve the performance of SWEAK, these values could be further tuned for different study subjects, our results clearly show that SWEAK is a promising solution.

Regarding NSGA-II, which SWEAK employs for searching worst-case test cases, Byers et al. [21] found that when the number of solution elements can freely evolve (i.e., variable-length genome), the crowding distance operator of NSGA-II prioritizes solutions with fewer elements. Hence, the search of NSGA-II is biased towards regions containing such solutions. However, in SWEAK, the size of a test case cannot freely evolve during search. In fact, the size of a test case is bounded within a range determined by the minimum and maximum task arrivals during the given simulation time. Furthermore, Byers et al. [21] observed this finding from the application of NSGA-II to address the remote data mirroring problem, which differs significantly from the problem addressed by SWEAK. Hence, further study is needed to investigate the extent to which bounded-variable-length genomes impact the performance of NSGA-II in general application contexts. In addition, considering other multi-objective search algorithms [54] can be beneficial to enhance the performance of SWEAK. Even though these research directions are interesting (but outside the scope of this article), our results nevertheless indicate that SWEAK significantly outperforms the baseline approach and estimates safe WCET ranges with a high degree of confidence in practical time.

External validity. The main threat to external validity is that our results may not be generalizable to other contexts. SWEAK explicitly targets WCET ranges that are estimated at early design stages when actual executions of the systems are not feasible. Hence, SWEAK was evaluated with an industrial system (i.e., ESAIL) from the satellite domain, using its task design descriptions. In addition, we applied SWEAK to a large number of synthetic systems generated by following BlackBerry's guidelines to ensure they were realistic and representative. We precisely described

the method used to generate synthetic systems (see Section 5.2). We have also made these systems available online [49]. Even though our results, based on simulations, show that SWEAK is applicable at early design stages for probabilistically estimating safe WCET ranges for weakly hard real-time systems, further studies remain necessary to evaluate whether or not the estimated safe WCET ranges are useful for engineers during later development stages, when tasks' implementations are available. Indeed, SWEAK is also applicable at later stages of development for testing the schedulability of systems and for providing more precise WCET ranges. Furthermore, additional study systems in other domains should be further investigated to evaluate the general usefulness of SWEAK.

6 RELATED WORKS

This section discusses previous studies on WCET estimation in real-time systems, schedulability analysis of real-time systems with weakly hard real-time tasks, and industry-strength task schedulers (e.g., APS). In addition, we discuss existing work that relies on search techniques for analyzing real-time systems. To our knowledge, no previous work investigated the probabilistic estimation of WCET ranges accounting for weakly hard real-time tasks and industry-strength task schedulers.

WCET estimation in real-time systems. Measurement-based methods to estimate WCETs are widely studied [1, 10, 20, 24, 38, 73] and commonly used in practice [3]. The basic idea of such methods is to run several task executions on the targeted hardware using various input data. To obtain tasks' input data, Wenzel et al. [73] proposed a method that analyzes the execution paths of source code. Burns and Edgar [20] proposed a probabilistic WCET estimation approach that applies statistical analysis to find worst-case input data. However, as they need executable source code and target hardware, these approaches are only applicable for systems at later stages of development.

To not rely on target hardware when estimating WCET values, static analysis approaches [32, 39, 58, 66] have been proposed. These approaches estimate WCET values based on an abstract model of the target hardware and software structure analysis. For example, some prior studies in these research strands rely on models of cache behaviors [39, 58, 66] or timing models of hardware instructions [5, 18, 37]. However, these approaches still require source code; hence, they are not applicable at early design stages.

In contrast to the previous studies that aim at estimating the WCETs of tasks regardless of their schedulability, SWEAK targets early stages of development, taking estimated, conservative WCET ranges as input. SWEAK then finds restricted safe WCET sub-ranges corresponding to a probability of violating deadline constraints, relying on a multi-objective search algorithm, simulation, feature reduction, a dedicated sampling strategy, and logistic regression. SWEAK enables trade-off analysis of tasks' WCET values and allows practitioners to select an appropriate violation probability depending on the context.

Schedulability analysis with weakly hard real-time tasks. To ensure quality of service (QoS) in real-world systems that can tolerate occasional deadline misses, Bernat et al. [11] introduced the concept of weakly hard real-time systems and precisely defined the concept of weakly hard deadline constraints adopted in SWEAK. The follow-up studies [60, 75] on weakly hard real-time systems introduced analytical methods to analyze the schedulability of weakly hard real-time systems.

Xu et al. [75] proposed a deadline miss model for computing the number of potential deadline misses within consecutive task arrivals when a task is under unexpected overload caused by task arrivals triggered through external events. In particular, they introduced task arrival models that capture arrival patterns of both periodic and sporadic tasks, enabling their analysis method to account for possible overload situations. Their analysis method relies on integer linear programming (ILP) and works under the assumption that tasks are running on a single-core platform with the fixed-priority preemptive or non-preemptive scheduling policy.

Pazzaglia et al. [60] presented a schedulability analysis method for weakly hard real-time systems by accounting for free offsets and release jitters using a mixed integer linear programming (MILP) formulation. Their analysis method checks whether or not a task satisfies its (m, K)-constraint to ensure that the task has no more than m deadline misses out of any K consecutive arrivals. This MILP-based method works under the assumption that a real-time system is composed of periodic tasks scheduled by fixed-priority with preemption on a single-core platform.

In contrast to these analytical schedulability analysis methods that takes fixed WCETs as input, SWEAK addresses the problem of probabilistically estimating safe WCET ranges that satisfy weakly-hard deadline constraints. SWEAK relies on logistic regression, search, and an industrial scheduling simulator, enabling it to analyze more complex systems involving uncertainties in task arrivals, context switching time, multiple processing cores, and advanced scheduling policies such as APS. The complex relationships among these system characteristics are difficult to capture in a (mixed) ILP formulation.

Adaptive partitioning scheduling (APS). Due to the increasing complexity of real-time systems, several APS approaches have been proposed [2, 17, 56]. These approaches combine global scheduling, which allows for the free migration of tasks, and static partitioned scheduling policies. Bletsas and Andersson [17] proposed a semi-adaptive partitioning scheduling approach that first allocates heavy-load tasks with utilization higher than 50% to processors. The scheduling approach then assigns the remaining tasks to the available time slots on each processor. Massa et al. [56] designed an online adaptive partitioning scheduling approach for a selected group of tasks based on offline analysis of their requirements. The groups are scheduled by global-like or partitioned scheduling methods depending on system load. Abeni and Cucinotta [2] introduced an approach that harmonizes global scheduling and static partitioned scheduling policies. The approach operates in partitioned mode and, if a system workload is over a specified threshold, it switches to global mode and then rapidly returns to the partitioned mode. Unlike these studies, BlackBerry provides an RTOS (i.e., QNX Neutrino) equipped with APS that supports the dynamic partition budget management for industrial use.

Even though the usage of APS for QNX Neutrino has been increasing, only a few schedulability analysis studies have recently been conducted for APS [25, 26]. Dasari et al. [26] found that APS has drawbacks in partition configurations. Hence, they provided some guidelines to help engineers properly configure partitions. Dasari et al. [25] investigated the APS behavior in practice and proposed a technique that verifies the end-to-end delay (i.e., schedulability) of event chains, which are the sequences of tasks activated by events, on a real-time system using APS. The technique employs a response time analysis of the chains at later development stages of systems containing periodic tasks with fixed WCET values. SWEAK complements these APS-related research strands as an analysis tool for estimating safe WCET ranges at early design stages. It helps engineers design their systems with safe WCET ranges as objectives with a certain level of confidence.

Search-based analysis in real-time systems. In real-time systems, search-based techniques are used in many prior approaches that aim at testing the systems [8, 51, 65, 71, 72]. For example, Wegener et al. [72] introduced a testing approach that relies on a genetic algorithm aiming at checking whether or not the system violates its timing constraints. Specifically, this prior work checks the timing constraints with regard to maximum and minimum execution times measured in processor cycles. Lin et al. [51] proposed a search-based approach to check whether a real-time system satisfies both its deadline and security constraints. Regarding security constraints, their work aims at optimally using security services for confidentiality, integrity, and authentication, while ensuring the schedulability of real-time tasks. Shin et al. [65] presented a test case prioritization approach based on a multi-objective search algorithm. This prior work, specifically, focused on studying the test case prioritization problem in the context of acceptance testing for cyber-physical

real-time systems. In this context, their approach accounts for not only the criticality levels of test cases but also the risk of hardware damage posed by executing the test cases.

Beyond testing real-time systems, recently, Lee et al. [48] proposed OPAM, an optimal task priority assignment method for real-time systems. OPAM uses a multi-objective, competitive coevolutionary search algorithm to find near-optimal priority assignments that maximize the magnitude of safety margins and the extent to which engineering constraints are satisfied. Lee et al. [50] introduced SAFE, a tool that provides probabilistic estimation of safe WCET ranges for real-time systems. As described in Section 4, however, SAFE is not applicable to weakly hard real-time systems, as SAFE accounts for hard deadline constraints that do not allow any occurrence of a deadline miss and relies on a simple task model. SWEAK extends SAFE to account for weakly hard deadline constraints, context switching times, and the APS policy.

In contrast to these prior works, SWEAK is the first attempt to address the problem of probabilistically estimating safe WCET ranges for weakly hard real-time systems. Furthermore, SWEAK accounts for multiple objectives to generate test cases that likely violate weakly hard deadline constraints and maximize the magnitude of deadline misses. SWEAK relies on logistic regression, adapted from SAFE, to infer safe WCET ranges in the form of safe WCET border with a probability of violating weakly hard deadline constraints. This enables engineers to investigate suitable WCET values by analyzing trade-offs within the safe ranges.

7 CONCLUSION

This article introduced SWEAK, which probabilistically estimates the safe WCET ranges of tasks for weakly hard real-time systems at early design stages. SWEAK employs a multi-objective search algorithm to find test cases (i.e., worst-case task arrivals and context switching times), aiming to maximize the magnitude of deadline misses and the degree of consecutive deadline misses. Based on the search results, SWEAK infers safe WCET ranges, for a probability of violating deadline constraints, using logistic regression. We evaluated SWEAK on a mission-critical real-time satellite system and several synthetic systems created by following the guidelines provided by BlackBerry. The results indicate that SWEAK provides high flexibility in selecting WCET ranges for practitioners. We further evaluated SWEAK through 600 synthetic systems with different characteristics, varying their degree of complexity. The results show that SWEAK scales to complex systems. Furthermore, SWEAK completed all experiments within at most 22.1h. Hence, SWEAK is acceptable in practice as an offline analysis technique for estimating safe WCET ranges given a probability of violating deadline constraints.

For future directions of this research, we plan to develop a real-time task modeling language that facilitates schedulability analysis and represents task constraints and system behaviors. In addition, the usefulness of SWEAK should be further validated with other case studies from different domains and through user studies. In particular, it is important to assess the usefulness of SWEAK from the perspective of practitioners. We plan to investigate how SWEAK assists engineers by providing safe WCET estimates, which, in turn, guide engineers in making appropriate decisions during the development process.

ACKNOWLEDGMENTS

This work was supported by Mitacs through the Mitacs Accelerate program (IT23234), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 694277), and NSERC of Canada under the Discovery and CRC programs. The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [68]– see hpc.uni.lu. We thank Chris Hobbs and Alexandre Koppel from

BlackBerry for their help in identifying the problem based on practitioners' needs and conducting realistic case studies.

REFERENCES

[1] Jaume Abella, Maria Padilla, Joan Del Castillo, and Francisco J. Cazorla. 2017. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM Transactions Design Automation of Electronic Systems* 22, 4, Article 72 (Jun 2017), 29 pages.

[2] Luca Abeni and Tommaso Cucinotta. 2020. Adaptive partitioning of real-time tasks on multiple processors. In *Proceedings of the ACM Symposium on Applied Computing (SAC'17)*. ACM, New York, NY, USA, 572–579.

[3] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert I. Davis. 2020. An Empirical Survey-based Study into Industry Practice in Real-time Systems. In *Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS'20)*, Vol. 2020-Decem. 3–11.

[4] T.A. AlEnawy and H. Aydin. 2005. Energy-constrained scheduling for weakly-hard real-time systems. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*. 376–385.

[5] Peter Altenbernd, Jan Gustafsson, Björn Lisper, and Friedhelm Stappert. 2016. Early Execution Time-Estimation Through Automatically Generated Timing Models. *Real-Time Systems* 52, 6 (2016), 731–760.

[6] Sebastian Altmeyer and Gernot Gebhard. 2008. WCET Analysis for Preemptive Scheduling. In *Proceeding of the 8th International Workshop Worst-Case Execution Time Analysis (WCET'08)*. Schloss Dagstuhl, Dagstuhl, Germany, 105–112.

[7] Andrea Arcuri and Lionel C. Briand. 2014. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability* 24, 3 (2014), 219–250.

[8] Andrea Arcuri, Muhammad Zohaib Iqbal, and Lionel C. Briand. 2010. Black-box system testing of real-time embedded systems using random and search-based testing. In *Proceedings of the IFIP International Conference on Testing Software and Systems (ICTSS'10)*, Vol. 6435. 95–110.

[9] S.K. Baruah, A. Burns, and R.I. Davis. 2011. Response-Time Analysis for Mixed Criticality Systems. In *Proceedings of the IEEE 32nd Real-Time Systems Symposium (RTSS'11)* (Vienna, Austria). IEEE, 34–43.

[10] Kostiantyn Berezovskyi, Luca Santinelli, Konstantinos Bletsas, and Eduardo Tovar. 2014. WCET Measurement-Based and Extreme Value Theory Characterisation of CUDA Kernels. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems (RTNS'14)* (Versaille, France). ACM, New York, NY, USA, 279–288.

[11] Guillem Bernat, Alan Burns, and Albert Llamosí. 2001. Weakly Hard Real-Time Systems. *IEEE Trans. Comput.* 50, 4 (2001), 308–321.

[12] Guillem Bernat, Antoine Colin, and Stefan M. Petters. 2002. WCET Analysis of Probabilistic Hard Real-Time System. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)* (Austin, TX, USA). IEEE, 279–288.

[13] Enrico Bini and Giorgio C. Buttazzo. 2004. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Comput.* 53, 11 (2004), 1462–1473.

[14] Enrico Bini and Giorgio C Buttazzo. 2005. Measuring the Performance of Schedulability Tests. *Real-Time Systems* 30 (2005), 129–154.

[15] BlackBerry QNX. 2022. Adaptive Partitioning Scheduler (APS). Retrieved October 6, 2022 from https://www.qnx.com/developers/docs/7.1/#com.qnx.doc.neutrino.sys_arch/topic/adaptive.html

[16] BlackBerry QNX. 2022. QNX Neutrino 7.1. Retrieved October 6, 2022 from https://blackberry.qnx.com/en/products/foundation-software/qnx-rtos

[17] Konstantinos Bletsas and Björn Andersson. 2009. Notional Processors: An Approach for Multiprocessor Scheduling. In *Proceedings of the 15th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'09)* (San Francisco, CA, USA). IEEE, 3–12.

[18] Armelle Bonenfant, Denis Claraz, Marianne De Michiel, and Pascal Sotin. 2017. Early WCET Prediction Using Machine Learning. In *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET'17)*, Vol. 57. Schloss Dagstuhl, Dagstuhl, Germany, Article 5, 9 pages.

[19] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[20] A. Burns and S. Edgar. 2000. Predicting computation time for advanced processor architectures. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems (ECRTS'00)*. 89–96.

[21] Chad M. Byers, Betty H. C. Cheng, and Kalyanmoy Deb. 2015. Unwanted Feature Interactions Between the Problem and Search Operators in Evolutionary Multi-objective Optimization. In *Proceedings of the 8th International Conference on Evolutionary Multi-Criterion Optimization*. 19–33.

[22] Jian Jia Chen, Georg Von Der Brüggen, and Niklas Ueter. 2018. Push forward: Global fixed-priority scheduling of arbitrary-deadline sporadic task systems. *Leibniz International Proceedings in Informatics, LIPIcs* 106 (2018), 1–24.

[23] Albert M. K. Cheng. 2003. *Real-Time Systems: Scheduling, Analysis, and Verification*. Wiley. 552 pages.

[24] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. 2012. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS'12)*. 91–101.

[25] Dakshina Dasari, Matthias Becker, Daniel Casini, and Tobias Blas. 2022. End-to-End Analysis of Event Chains under the QNX Adaptive Partitioning Scheduler. In *Proceedings of the IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS'22)*. IEEE, 214–227.

[26] Dakshina Dasari, Arne Hamann, Holger Broede, Michael Pressler, and Dirk Ziegenbein. 2021. Brief Industry Paper: Dissecting the QNX Adaptive Partitioning Scheduler. In *Proceedings of the IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS'21)*. IEEE, 477–480.

[27] Robert Davis and Liliana Cucu-Grosjean. 2019. A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems. *LITES: Leibniz Transactions on Embedded Systems* (2019), 1–60.

[28] Robert I. Davis and Alan Burns. 2011. Improved Priority Assignment for Global Fixed Priority Pre-Emptive Scheduling in Multiprocessor Real-Time Systems. *Real-Time Systems* 47, 1 (Jan 2011), 1–40.

[29] Robert I. Davis, Attila Zabos, and Alan Burns. 2008. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Trans. Comput.* 57, 9 (2008), 1261–1276.

[30] Marco Dürr, Georg Von Der Brüggen, Kuan Hsun Chen, and Jian-Jia Chen. 2019. End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems. *ACM Transactions on Embedded Computing Systems* 18, 5s (Oct 2019), 1–24.

[31] Paul Emberson, Roger Stafford, and Robert I. Davis. 2010. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS'10)*. 6–11.

[32] Christian Ferdinand and Reinhard Wilhelm. 1998. On predicting data cache behavior for real-time systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1474 (1998), 16–30.

[33] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

[34] Michel Gendreau and Jean-Yves Potvin. 2010. *Handbook of Metaheuristics* (2nd ed.). Springer.

[35] Oliver Gettings, Sophie Quinton, and Robert I. Davis. 2015. Mixed Criticality Systems with Weakly-Hard Constraints. In *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS'15)*. ACM, 237–246.

[36] Werner Grass and Thi Huyen Chau Nguyen. 2018. Improved response-time bounds in fixed priority scheduling with arbitrary deadlines. *Real-Time Systems* 54, 1 (2018), 1–30.

[37] Jan Gustafsson, Peter Altenbernd, Andreas Ermedahl, and Björn Lisper. 2009. Approximate Worst-Case Execution Time Analysis for Early Stage Embedded Systems Development. In *Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS'09)*. Springer, Berlin, Heidelberg, 308–319.

[38] Jeffery P. Hansen, Scott A. Hissam, and Gabriel A. Moreno. 2009. Statistical-Based WCET Estimation and Validation. In *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis (WCET'09)*. Schloss Dagstuhl, Dagstuhl, Germany, 1–11.

[39] Damien Hardy and Isabelle Puaut. 2011. WCET analysis of instruction cache hierarchies. *Journal of Systems Architecture* 57, 7 (Aug 2011), 677–694.

[40] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Computing Survey* 45, 1, Article 11 (2012), 61 pages.

[41] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer. 745 pages.

[42] Randy L. Haupt and Sue Ellen Haupt. 1998. *Practical Genetic Algorithms*. John Wiley & Sons, Inc. 288 pages.

[43] Kawakubo Hideko and Yoshida Hiroaki. 2012. Rapid Feature Selection Based on Random Forests for High-Dimensional Data. *Expert Systems with Applications* 40, 1 (2012), 6241–6252.

[44] International Organization for Standardization. 2018. ISO 26262: Road vehicles-functional safety. (2018).

[45] David W. Hosmer Jr., Stanley Lemeshow, and Rodney X. Sturdivant. 2013. *Applied Logistic Regression* (3rd ed.). John Wiley & Sons, Inc. 528 pages.

[46] André I Khuri and Siuli Mukhopadhyay. 2010. Response surface methodology. *Wiley Interdisciplinary Reviews: Computational Statistics* 2, 2 (2010), 128–149.

[47] Rocco Le Moigne, Olivier Pasquier, and J-P Calvez. 2004. A Generic RTOS Model for Real-Time Systems Simulation with SystemC. In *Proceedings of the 2004 Design, Automation and Test in Europe Conference and Exhibition*. 82–87.

[48] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, and Lionel C. Briand. 2022. Optimal Priority Assignment for Real-Time Systems: A Coevolution-Based Approach. *Empirical Software Engineering* 27, 6 (2022), 142:1–49.

[49] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, and Lionel C. Briand. 2023. [Case study data] Probabilistic Safe WCET Estimation for Weakly Hard Real-Time Systems at Design Stages. https://github.com/SNTSVV/SWEAK.

[50] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, Lionel C. Briand, and Yago Isasi Parache. 2022. Estimating Probabilistic Safe WCET Ranges of Real-Time Systems at Design Stages. *ACM Transactions on Software Engineering and Methodology* (Jun 2022). Just Accepted.

[51] Man Lin, Li Xu, Laurence T. Yang, Xiao Qin, Nenggan Zheng, Zhaohui Wu, and Meikang Qiu. 2009. Static security optimization for real-time systems. *IEEE Transactions on Industrial Informatics* 5, 1 (2009), 22–37.

[52] Chang Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[53] Jane W. S. Liu. 2000. *Real-Time Systems* (1st ed.). Prentice Hall PTR.

[54] Sean Luke. 2013. *Essentials of Metaheuristics* (2nd ed.). Lulu. http://cs.gmu.edu/~sean/book/metaheuristics/

[55] Henry B. Mann and Donald R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

[56] Ernesto Massa, George Lima, Paul Regnier, Greg Levin, and Scott Brandt. 2016. Quasi-partitioned scheduling: optimality and adaptation in multiprocessor real-time systems. *Real-Time Systems* 52, 5 (2016), 566–597.

[57] Mohammad Moallemi and Gabriel Wainer. 2013. Modeling and Simulation-Driven Development of Embedded Real-Time Systems. *Simulation Modelling Practice and Theory* 38 (2013), 115–131.

[58] Frank Mueller. 2000. Timing analysis for instruction caches. *Real-Time Systems* 18, 2 (2000), 217–247.

[59] Thanh-Tung Nguyen, Joshua Zhexue Huang, and Thuy Thi Nguyen. 2015. Unbiased Feature Selection in Learning Random Forests for High-Dimensional Data. *The Scientific World Journal* 2015 (2015), 1–18.

[60] Paolo Pazzaglia, Youcheng Sun, and Marco Di Natale. 2021. Generalized Weakly Hard Schedulability Analysis for Real-Time Periodic Tasks. *ACM Transactions on Embedded Computing Systems* 20, 1, Article 3 (2021), 26 pages.

[61] Paul Ralph, Nauman bin Ali, Sebastian Baltes, Domenico Bianculli, Jessica Diaz, Yvonne Dittrich, Neil Ernst, Michael Felderer, Robert Feldt, Antonio Filieri, Breno Bernard Nicolau de França, Carlo Alberto Furia, Greg Gay, Nicolas Gold, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara Kitchenham, Valentina Lenarduzzi, Jorge Martínez, Jorge Melegati, Daniel Mendez, Tim Menzies, Jefferson Molleri, Dietmar Pfahl, Romain Robbes, Daniel Russo, Nyyti Saarimäki, Federica Sarro, Davide Taibi, Janet Siegmund, Diomidis Spinellis, Miroslaw Staron, Klaas Stol, Margaret-Anne Storey, Davide Taibi, Damian Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, Xiaofeng Wang, and Sira Vegas. 2020. Empirical Standards for Software Engineering Research. arXiv:2010.03525 [cs.SE]

[62] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence - A Modern Approach* (3rd ed.). Pearson Education. 1132 pages.

[63] Luca Santinelli, Fabrice Guet, and Jerome Morio. 2017. Revising Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the 2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*. 199–208.

[64] Jeff Schaffer and Steve Reid. 2011. The joy of scheduling. *QNX Software Systems* (2011).

[65] Seung Yeob Shin, Shiva Nejati, Mehrdad Sabetzadeh, Lionel C. Briand, and Frank Zimmer. 2018. Test Case Prioritization for Acceptance Testing of Cyber Physical Systems: A Multi-Objective Search-Based Approach. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'18)*. 49–60.

[66] Henrik Theiling, Christian Ferdinand, and Reinhard Wilhelm. 2000. Fast and Precise WCET Prediction by Separated Cache and Path Analyses. *Real-Time Systems* 18, 2 (2000), 157–179.

[67] András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics* 25, 2 (2000), 101–132.

[68] Sébastien Varrette, Pascal Bouvry, Hyacinthe Cartiaux, and Fotis Georgatos. 2014. Management of an Academic HPC Cluster: The UL Experience. In *Proceedings of the 2014 International Conference on High Performance Computing & Simulation (HPCS'14)* (Bologna, Italy). IEEE, 959–967.

[69] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. 2018. Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems. In *Proceedings of the 30th Euromicro Conference on Real-Time Systems (ECRTS'18)*, Vol. 106. Schloss Dagstuhl, Dagstuhl, Germany, Article 6, 22 pages.

[70] K. C. Wang. 2017. *Embedded and Real-Time Operating Systems* (1st ed.). Springer. 481 pages.

[71] Joachim Wegener and Matthias Grochtmann. 1998. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems* 15, 3 (1998), 275–298.

[72] Joachim Wegener, Harmen Sthamer, Bryan F. Jones, and David E. Eyres. 1997. Testing real-time systems using genetic algorithms. *Software Quality Journal* 6, 2 (1997), 127–135.

[73] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. 2005. Measurement-based worst-case execution time analysis. In *Proceedings of the Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*. 7–10.

[74] Ian H. Witten, Eibe Frank, and Mark A. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc. 665 pages.

[75] Wenbo Xu, Zain Alabedin Haj Hammadeh, Alexander Kröller, Rolf Ernst, and Sophie Quinton. 2015. Improved Deadline Miss Models for Real-Time Systems Using Typical Worst-Case Analysis. In *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS'15)* (Lund, Sweden). IEEE, 247–256.

[76] Toshie Yamashita, Keizo Yamashita, and Ryotaro Kamimura. 2007. A Stepwise AIC Method for Variable Selection in Linear Regression. *Communications in Statistics - Theory and Methods* 36, 13 (2007), 2395–2403.

[77] Fengxiang Zhang and Alan Burns. 2009. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Trans. Comput.* 58, 9 (2009), 1250–1258.