Estimating Probabilistic Safe WCET Ranges of Real-Time Systems at Design Stages

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

Estimating worst-case execution times (WCET) is an important activity at early design stages of real-time systems. Based on WCET estimates, engineers make design and implementation decisions to ensure that task executions always complete before their specified deadlines. However, in practice, engineers often cannot provide precise point WCET estimates and prefer to provide plausible WCET ranges. Given a set of real-time tasks with such ranges, we provide an automated technique to determine for what WCET values the system is likely to meet its deadlines, and hence operate safely with a probabilistic guarantee. Our approach combines a search algorithm for generating worst-case scheduling scenarios with polynomial logistic regression for inferring probabilistic safe WCET ranges. We evaluated our approach by applying it to three industrial systems from different domains and several synthetic systems. Our approach efficiently and accurately estimates probabilistic safe WCET ranges within which deadlines are likely to be satisfied with a high degree of confidence.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; **Search-based software engineering**; • **Computer systems organization → Real-time systems**.

Additional Key Words and Phrases: Schedulability Analysis, Worst-Case Execution Time, Meta-Heuristic Search, Machine Learning, Search-Based Software Engineering

ACM Reference Format:
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 Trans. Softw. Eng. Methodol.* 1, 1 (June 2022), 33 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Safety-critical systems, e.g., those used in the aerospace, automotive and healthcare domains, require that their executions always complete before their specified deadlines in all execution scenarios, including the worst cases. The systems that must perform their operations in such a

Authors' addresses: Jaekwon Lee, jaekwon.lee@uni.lu, University of Luxembourg, 29 Avenue John F. Kennedy, Luxembourg, 1859, Luxembourg, 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; Shiva Nejati, snejati@uottawa.ca, University of Ottawa, 800 King Edward Avenue, Ottawa, ON K1N 6N5, Canada, 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, University of Ottawa, 800 King Edward Avenue, Ottawa, ON K1N 6N5, Canada; Yago Isasi Parache, isasi@luxspace.lu, LuxSpace, 9 Rue Pierre Werner, Betzdorf, 6832, Luxembourg.

1049-331X/2022/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

timely manner are known as real-time systems (RTS) [18]. To ensure that a real-time system meets its deadlines, we need an accurate estimation of the worst-case execution times (WCET) of software tasks that concurrently run in the system. For instance, the Anti-lock Braking System (ABS) of a vehicle has to activate within milliseconds after the driver brakes. However, an ABS taking more time for activation than the estimated WCET may result in a vehicle skid due to the wheels locking up.

Accurately estimating WCET values of a real-time system is particularly important at early design stages when real-time tasks are not yet fully implemented. Accurate WCET estimates greatly support engineers during development as they provide targets driving design and implementation choices. Based on the WCET estimates, engineers can make design and implementation decisions, e.g., using either a relational database or an in-memory data storage. In addition, when engineers develop software and hardware components in parallel, which is common in the aerospace, automotive, and healthcare domains, accurately estimated WCET values at early design stages help engineers find optimal configurations of hardware devices, e.g., CPUs, sensors, and actuators, by accounting for time constraints and performance requirements of the system.

Real-time tasks have various parameters such as task priorities, deadlines, inter-arrival times, and WCET values [22, 63]. Among the task parameters, WCET values are typically difficult to accurately estimate at early design stages. The other parameters, however, can be specified or estimated with a high degree of precision even at early stages. For example, task priorities are typically determined by the selected scheduling policy, e.g., rate monotonic [44], or based on the task criticality levels (i.e., more critical tasks are prioritised over the less critical ones). Task deadlines are typically decided by system requirements. Task inter-arrival times, i.e., the time interval between consecutive task executions, usually depend on system environmental events triggering task executions. In contrast, the WCET values of some tasks may depend on various factors such as implementation decisions, task durations, real-time operating systems, and hardware components. However, these factors may not be fully known at early stages of development, making it difficult to precisely estimate WCET values for real-time tasks [3, 15, 33]. As a result, engineers tend to provide ranges for WCET values instead of point estimates.

The problem of estimating WCET values is, in general, a hard problem. WCET values of real-time tasks impact every possible task schedules. WCET values depend on the content and implementation of tasks and not the schedule. But they impact how the tasks are scheduled. The space of all possible task schedules is very large. In our context, the problem becomes computationally more expensive when WCET values are uncertain and are specified as value ranges instead of single values. Specifically, provided with WCET value ranges, engineers need to have ways to determine for what WCET values, within the given ranges, the system is likely to miss or satisfy its deadline constraints. If engineers know that deadlines are likely met for all or most of the expected WCET ranges, they can consider a wider choice of design and implementation options. Otherwise, in situations where only tight WCET sub-ranges seem acceptable, developers may have to consider more expensive hardware, decreased functionality or performance, or more restricted design and implementation choices.

The WCET estimation problem of real-time systems has been widely studied in the past [3, 13, 15, 33–35]. To our knowledge, however, most of the existing WCET estimation methods often fail to provide WCET estimates at early design stages because they require inputs that can be defined only at a later point in time such as task implementations and hardware devices. Some model-based approaches [4, 20, 64] try to solve the WCET estimation problem exhaustively by applying a model checker to a real-time model, e.g., parametric timed automata, of the system under analysis. These exhaustive methods are applicable once real-time system models are available. However, such approaches tend to suffer from the state-space explosion problem [21] as the number of

software tasks and their different states increase. More recently, stress testing and simulation-based approaches [2, 17] have been proposed to stress RTS and generate test scenarios where their deadline constraints are violated. Such approaches cast the schedulability test problem as an optimisation problem to find worst-case task execution scenarios exhibiting deadline misses. However, none of the existing simulation-based approaches account for uncertainties in WCET values and therefore do not handle WCET value ranges. Our work complements the simulation-based stress testing approach and extends it to account for uncertainties in WCET values.

Contributions. In this article, we propose a Safe WCET Analysis method For real-time task schEdulability (SAFE) to estimate WCET ranges under which tasks are likely to be schedulable with a probabilistic guarantee. Our approach is based on a stress testing approach [17] using meta-heuristic search [47] in combination with polynomial logistic regression models. Specifically, we use a genetic algorithm [47] to search for sequences of task arrivals which likely lead to deadline misses. Then, logistic regression [40], a statistical classification technique, is applied to infer a *safe WCET border* in the multidimensional WCET space with a probabilistic guarantee. This border aims to partition the given WCET ranges into *safe* and *unsafe* sub-ranges for a selected deadline miss probability p, and thus enables engineers to investigate trade-offs among different tasks' WCET values. WCET ranges are deemed to be probabilistically safe if tasks, within such ranges, have a high probability to complete their executions before their specified deadlines. In this article, for the sake of simplicity, we refer to probabilistically safe WCET ranges as safe WCET ranges. We evaluated our approach by applying it to a complex, industrial satellite system developed by our industry partner, LuxSpace, as well as two industrial systems from different domains and several synthetic systems. Results show that our approach can efficiently and accurately compute safe WCET ranges. SAFE scales to complex industrial systems as an offline analysis method. Execution times of SAFE on our industrial systems are practically acceptable, i.e., at most 27h. To our knowledge, SAFE is the first attempt to estimate safe WCET ranges within which real-time tasks are likely to meet their deadlines for a given level of confidence, while enabling engineers to explore trade-offs among tasks' WCET values. Our full evaluation package is available online [43].

Organization. The remainder of this article is structured as follows: Section 2 motivates our work. Section 3 defines our specific schedulability analysis problem in practical terms. Section 4 describes SAFE. Section 5 evaluates SAFE. Sections 6 compares SAFE with related work. Section 7 concludes this article.

2 MOTIVATING CASE STUDY

We motivate our work with a mission-critical real-time satellite system, named Attitude Determination and Control System (ADCS), which LuxSpace, a leading system integrator for microsatellites and aerospace systems, has been developing over the years. ADCS determines the satellite's attitude and controls its movements [29]. ADCS controls a satellite in either autonomous or passive mode. In the autonomous mode, ADCS must orient a satellite in proper position on time to ensure that the satellite provides normal service correctly. In the passive mode, operators are able to not only control satellite positions but also maintain the satellite, e.g., upgrading software. Such a maintenance operation does not necessarily need to be completed within a fixed hard deadline; instead, it should be completed within a reasonable amount of time, i.e., soft deadlines. Hence, ADCS is composed of a set of tasks having real-time constraints with hard and soft deadlines.

Engineers at LuxSpace conduct real-time schedulability analysis across different development stages. At an early design stage, when task implementations and system hardware are not available, the engineers use a theoretical schedulability analysis technique [44] which determines that a set of tasks is schedulable if CPU utilisation of the task set is less than a threshold, e.g., 69%. As mentioned earlier, at an early design stage, engineers estimate task WCETs as ranges and often assign large

values to the upper bounds of such ranges. To be on the safe side, engineers tend indeed to be conservative in their analysis.

Engineers, however, are still faced with the following issues: (1) An analytical schedulability analysis technique, e.g., utilisation-based schedulability analysis [44], typically indicates whether or not tasks are schedulable. However, engineers need additional information to understand how tasks miss their deadlines. For instance, a set of tasks may not be schedulable for a few specific sequences of task arrivals. (2) Engineers estimate WCETs without any systematic support; instead, they often rely on their experience of developing tasks providing similar functions-to-develop. This practice typically results in imprecise estimates of WCET ranges, which may cause serious problems, e.g., significantly changing tasks at later development stages. To this end, LuxSpace is interested in SAFE as a way to address these issues in analysing schedulability.

3 PROBLEM DESCRIPTION

This section first formalises task, task relationship, scheduler, and schedulability concepts. We then describe the problem of identifying safe WCET ranges under which tasks likely meet their deadline constraints at a certain level of confidence; i.e., tasks are schedulable with a certain probability.

Task. A real-time system is composed of a set of n tasks that should complete their executions within specified deadlines after they are activated (or arrived). We denote by τ_i a real-time task indexed by i in the range from 1 to n. Every real-time task τ_i has the following properties: priority denoted by P_i, deadline denoted by D_i, and worst-case execution time (WCET) denoted by C_i. Task priority P_i determines if an execution of a task is preempted by another task. Typically, a task τ_i preempts the execution of a task τ_j if the priority of τ_i is higher than the priority of τ_j, i.e., $P_i > P_j$.

The deadline of a task τ_i relative to its arrival time is denoted by D_i. A task deadline can be either *hard* or *soft*. A hard deadline of a task τ_i specifies that τ_i *must* complete its execution within a deadline D_i after τ_i is activated. While violations of hard deadlines are not acceptable, depending on the operating context of a system, violating soft deadlines may be tolerated to some extent. Note that, for notational simplicity, we do not introduce new notations to distinguish between hard and soft deadlines. In this article, we refer to a hard deadline as a deadline. Section 4 further discusses how our approach manages hard and soft deadlines.

We denote by C_i^{min} and C_i^{max}, respectively, the minimum and the maximum WCET values of a task τ_i. As discussed in the introduction, at an early development stage, it is difficult to provide exact WCET values of real-time tasks. Hence, we assume that engineers specify WCETs using a range of values, instead of single values, by indicating estimated minimum and maximum values that they think each task's WCET can realistically take.

In this article, real-time tasks are either *periodic* or *aperiodic*. Periodic tasks, which are typically triggered by timed events, are invoked at regular intervals specified by their *period*. We denote by T_i the period of a periodic task τ_i, i.e., a fixed time interval between subsequent activations (or arrivals) of τ_i. Aperiodic tasks have irregular arrival times and are activated by external stimuli which occur irregularly, and hence, 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 denoted by T_i^{min} and a maximum inter-arrival time denoted by T_i^{max} indicating the minimum and maximum time intervals between two consecutive arrivals of an aperiodic task τ_i. In real-time analysis, sporadic tasks are often separately defined as having irregular arrival intervals and hard deadlines [45]. In our conceptual definitions, however, we do not introduce new notations for sporadic tasks because the deadline and period concepts defined above are sufficient to characterise sporadic tasks. Note that for a periodic task τ_i, we have $T_i^{min} = T_i^{max} = T_i$. Otherwise, for an aperiodic task τ_j, we have $T_j^{max} > T_j^{min}$.

Task relationship. The execution of a task τ_i depends not only on its own parameters described above, e.g., priority P_i and period T_i, but also on its relationships with other tasks. Relationships between tasks are typically determined by task interactions related to accessing shared resources [1], such as memory, file, and IO devices. Specifically, if two tasks τ_i and τ_j access a shared resource in a mutually exclusive way, τ_i may be blocked from executing for the period during which τ_j accesses the resource. We denote by $dp(\tau_i, \tau_j)$ the resource-dependency relation between tasks τ_i and τ_j that holds if τ_i and τ_j have mutually exclusive access to a shared resource such that they cannot be executed in parallel or preempt each other, but one can execute only after the other has completed its access to the resource. We note that resource-dependency relations are defined at the level of tasks, following prior works [5, 26, 46] describing the industrial case study systems used in our experiments (see Section 5.2). The $dp(\tau_i, \tau_j)$ relation is symmetric, i.e., $dp(\tau_i, \tau_j) = dp(\tau_j, \tau_i)$.

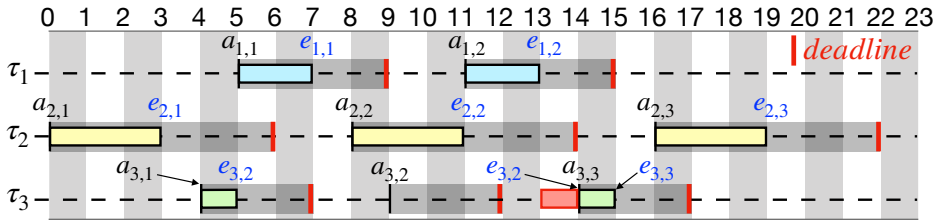
Scheduler. Let Γ be a set of tasks to be scheduled by a real-time scheduler. A scheduler then dynamically schedules executions of tasks in Γ according to the tasks' arrivals and the scheduler's scheduling policy over the scheduling period $\mathbb{T} = [0, \mathbf{t}]$. We denote by $a_{i,k}$ the kth arrival time of a task $\tau_i \in \Gamma$. The first arrival of a periodic task τ_i does not always occur immediately at the system start time 0. Such offset time from the system start time 0 to the first arrival time $a_{i,1}$ of τ_i is denoted by O_i. For a periodic task τ_i, the kth arrival of τ_i within $\mathbb{T}$ is $a_{i,k} \leq \mathbf{t}$ and is computed by $a_{i,k} = O_i + (k-1) \cdot T_i$. For an aperiodic task τ_j, $a_{j,k}$ is determined based on the $k-1$th arrival time of τ_j and its minimum and maximum arrival times. Specifically, for $k > 1$, $a_{j,k} \in [a_{j,k-1} + T_j^{min}, a_{j,k-1} + T_j^{max}]$ and, for $k = 1$, $a_{j,1} \in [T_j^{min}, T_j^{max}]$ where $a_{j,k} \leq \mathbf{t}$.

A scheduler reacts to a task arrival at $a_{i,k}$ to schedule the execution of τ_i. Depending on a scheduling policy (e.g., rate monotonic scheduling policy [44] and single-queue multi-core scheduling policy [7]), an arrived task τ_i may not start its execution at the same time as it arrives when a higher priority task is executing. Also, task executions may be interrupted due to preemption. We denote by $e_{i,k}$ the end execution time for the kth arrival of a task τ_i. Depending on actual worst-case execution time of a task τ_i, denoted by C_i, within its WCET range $[C_i^{min}, C_i^{max}]$, the $e_{i,k}$ end execution time of τ_i satisfies the following: $e_{i,k} \geq a_{i,k} + C_i$.

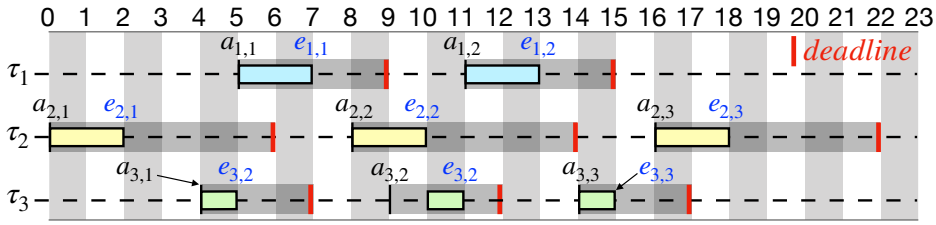
During the system operation, a scheduler generates a *schedule scenario* which describes a sequence of task arrivals and their end time values. We define a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ indicating that a task τ_i has arrived at $a_{i,k}$ and completed its execution at $e_{i,k}$. Due to the randomness of task execution times and aperiodic task arrivals, a scheduler may generate a different schedule scenario in different runs of a system.

Figure 1 shows two schedule scenarios produced by a scheduler over the $[0, 23]$ time period of a system run. Both Figure 1a and Figure 1b describe executions of three tasks, τ_1, τ_2, and τ_3 arrived at the same time stamps (see $a_{i,k}$ in the figures). In both scenarios, the aperiodic task τ_1 is characterised by: $T_1^{min} = 5$, $T_1^{max} = 10$, $D_1 = 4$, and $C_1^{min} = C_1^{max} = 2$. The periodic task τ_2 is characterised by: $T_2 = 8$ and $D_2 = 6$. The aperiodic task τ_3 is characterised by: $T_3^{min} = 3$, $T_3^{max} = 20$, $D_3 = 3$, and $C_3^{min} = C_3^{max} = 1$. The priorities of the three tasks satisfy the following: $P_1 > P_2 > P_3$. In both scenarios, task executions can be preempted depending on their priorities. We note that a WCET range of the τ_2 task is set to $C_2^{min} = 1$ and $C_2^{max} = 3$ in Figure 1a, and $C_2^{min} = 1$ and $C_2^{max} = 2$ in Figure 1b. Then, Figure 1a can be described by the $S_a = \{(\tau_1, 5, 7), \ldots, (\tau_2, 0, 3), \ldots, (\tau_3, 9, 14), (\tau_3, 14, 15)\}$ schedule scenario; and Figure 1b by $S_b = \{(\tau_1, 5, 7), \ldots, (\tau_2, 0, 2), \ldots, (\tau_3, 9, 11), (\tau_3, 14, 15)\}$.

Schedulability. Given a schedule scenario S, a task τ_i is *schedulable* if τ_i completes its execution before its deadline, i.e., for all $e_{i,k}$ observed in S, $e_{i,k} \leq a_{i,k} + D_i$. Let Γ be a set of tasks to be scheduled by a scheduler. A set Γ of tasks is then schedulable if for every schedule S of Γ, we have no task $\tau_i \in \Gamma$ that misses its deadline.



(a) Deadline miss



(b) No deadline miss

Fig. 1. Example schedule scenarios of three tasks, τ_1, τ_2, and τ_3, running on a single core system. (a) τ_3 is not schedulable, i.e., $e_{3,2} > a_{3,2} + D_3$. (b) All the three tasks are schedulable. When τ_2 executes over 3 (WCET) time units, it causes a deadline miss of τ_3. When the WCET is reduced to 2, the three tasks are schedulable even for the same sequence of task arrivals.

As shown in Figure 1a, a deadline miss occurs after the second arrival of τ_3, i.e., $e_{3,2} > a_{3,2} + D_3$. During $[a_{3,2}, a_{3,2} + D_3]$ period, the τ_3 task cannot execute because the other tasks τ_1 and τ_2 with higher priorities are executing. Thus, τ_3 is not schedulable in the schedule scenario of Figure 1a. This scheduling problem can be solved by restricting tasks' WCET ranges as discussed below.

Problem. Uncertainty in task WCET values at an early development stage is a critical issue preventing the effective design and assessment of mission-critical real-time systems. Upper bounds of WCETs correspond to worst-case WCET values and have a direct impact on deadline misses as larger WCET values increase their probability. Lower bounds of WCETs are estimates of tasks' best-case WCET values, below which task implementations are likely not feasible. Our approach aims to determine the maximum upper bounds for WCET under which tasks are likely to be schedulable, at a given level of risk, and thus provides an objective to engineers implementing the tasks. Specifically, for every task $\tau_i \in \Gamma$ to be analysed, our approach computes a new upper bound value for the WCET range of τ_i (denoted by C_i^{max*}) such that $C_i^{max*} \leq C_i^{max}$ and by restricting the WCET range of τ_i to C_i^{max*} we should, at a certain level of confidence, no longer have deadline misses. That is, tasks Γ become schedulable, with a certain probability, after restricting the maximum WCET value of τ_i to C_i^{max*}. For instance, as shown in Figure 1b, restricting the maximum WCET of τ_2 from $C_2^{max} = 3$ to $C_2^{max*} = 2$ enables all the three tasks to be schedulable.

We note that, in our context, both arrival time ranges for aperiodic tasks and WCET ranges for all tasks are represented as continuous intervals. Since our approach works based on sampling values from these continuous ranges, our approach cannot be exhaustive and cannot provide a guarantee that the tasks can always be schedulable after restricting their WCET ranges. Our approach instead relies on sampling values within the WCET and arrival time ranges, simulating the scheduler behaviour using the sampled values and observing whether, or not, a deadline miss occurs. In lieu

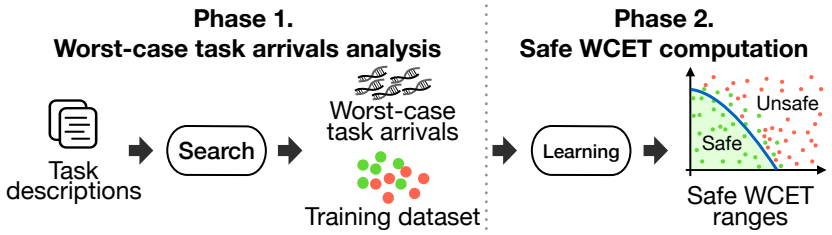


Fig. 2. An overview of our Safe WCET Analysis method For real-time task schEdulability (SAFE).

of exhaustiveness, we rely on statistical and machine learning techniques to provide probabilistic estimates indicating how confident we are that a given set of tasks are schedulable.

4 APPROACH

Figure 2 shows an overview of our Safe WCET Analysis method For real-time task schEdulability (SAFE). Phase 1 of SAFE aims at searching worst-case task-arrival sequences. A task-arrival sequence is worst-case if deadline misses are maximised or, when this is not possible, tasks complete their executions as close to their deadlines as possible. Building on existing work, we identify worst-case task-arrival sequences using a search-based approach relying on genetic algorithms. Phase 2 of SAFE, which is the main contribution of this article, aims at computing safe WCET ranges under which tasks are likely to be schedulable. To do so, relying on logistic regression and an effective sampling strategy, we augment the worst-case task-arrival sequences generated in Phase 1 to compute safe WCET ranges with a certain deadline miss probability, indicating a degree of risk. We describe in detail these two phases next.

4.1 Phase 1: Worst-case task arrivals

The first phase of SAFE finds worst-case sequences in the space of possible sequences of task arrivals, defined by their inter-arrival time characteristics. As SAFE aims to provide conservative, safe WCET ranges, we optimise task arrivals to maximise task completion times and deadline misses, and indirectly minimise safe WCET ranges (see the safe area visually presented in Figure 2). We address this optimisation problem using a single-objective search algorithm. Following standard practice [30], we describe our search-based approach for identifying worst-case task arrivals by defining the solution representation, the scheduler, the fitness function, and the computational search algorithm. We then describe the dataset of sequences generated by search and then used for training our logistic regression model to compute safe WCET ranges in the second phase of SAFE.

Our approach in Phase 1 is based on past work [17], where a specific genetic algorithm configuration was proposed to find worst-case task arrival sequences. One important modification though is that we account for uncertainty in WCET values through simulations for evaluating the magnitude of deadline misses.

Representation. Given a set Γ of tasks to be scheduled, a feasible solution is 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 a task τ_i. Thus, a solution A represents a valid sequence of task arrivals of Γ (see valid $a_{i,k}$ computation in Section 3). Let $\mathbb{T} = [0, \mathbf{t}]$ be the time period during which a scheduler receives task arrivals. The size of A is equal to the number of task arrivals over the $\mathbb{T}$ time period. Due to the varying inter-arrival times of aperiodic tasks (Section 3), the size of A will vary across different solutions.

Scheduler. SAFE uses a simulation technique for analysing the schedulability of tasks to account for the uncertainty in WCET values and scalability issues. For instance, an inter-arrival time of a

software update task in a satellite system is approximately at most three months. In such cases, conducting an analysis based on an actual scheduler is prohibitively expensive. Instead, SAFE uses a real-time task scheduling simulator, named SafeScheduler, which samples WCET values from their ranges for simulating task executions and applies a scheduling policy, i.e., single-queue multi-core scheduling policy [7], based on discrete simulation time events. Note that we chose the single-queue multi-core scheduling policy for SafeScheduler since our case study systems (described in Section 5.2) rely on this policy.

SafeScheduler takes a feasible solution A for scheduling a set Γ of tasks as an input. It then outputs a schedule scenario as a set S of tuples $(\tau_i, a_{i,k}, e_{i,k})$ where $a_{i,k}$ and $e_{i,k}$ are the kth arrival and end time values of a task τ_i, respectively. Recall from Section 3 that SafeScheduler computes task arrivals based on periodic tasks' offsets and periods and aperiodic tasks' inter-arrival times. For each task τ_i, SafeScheduler computes $e_{i,k}$ based on its scheduling policy and a selected WCET value for τ_i within the WCET range $[C_i^{min}, C_i^{max}]$, while accounting for resource-dependency relationships (see Section 3). Hence, each run of SafeScheduler for the same input solution A will likely produce a different schedule scenario.

SafeScheduler implements a single-queue multi-core scheduling policy [7], which schedules a task τ_i with explicit priority P_i and deadline D_i. When tasks arrive, SafeScheduler puts them into a single queue that contains tasks to be scheduled. At any simulation time, if there are tasks in the queue and multiple cores are available to execute tasks, SafeScheduler first fetches a task τ_i from the queue in which τ_i has the highest priority P_i. SafeScheduler then allocates task τ_i to any available core. Note that if task τ_i shares a resource with a running task τ_j in another core, i.e., the $dp(\tau_i, \tau_j)$ resource-dependency relationship holds, SafeScheduler follows standard task-blocking rules [45], i.e., τ_i will be blocked until τ_j releases the shared resource.

SafeScheduler works under the assumption that context switching time is free, which is also a working assumption in many scheduling analysis methods [8, 26, 44]. Note that the assumptions are practically valid and useful at an early development step in the context of real-time analysis. For instance, our collaborating partner accounts for the waiting time of tasks due to context switching between tasks through adding some extra time to WCET ranges at the task design stage. Note that SAFE can be applied with any scheduling policy, including those that account for context switching time and multiple queues.

Fitness. Given a feasible solution A for a set Γ of tasks, we formulate a fitness function, $f(A, \Gamma^\delta, ns)$, to quantify the degree of deadline misses regarding a set $\Gamma^\delta \subseteq \Gamma$ of target tasks, where ns is a number of SafeScheduler runs to account for the uncertainty in WCET. SAFE provides the capability of selecting target tasks Γ^δ as practitioners often need to focus on the most critical tasks. We denote by $dist(\tau_i, k)$ the distance between the end time and the deadline of the kth arrival of task τ_i and define $dist(\tau_i, k) = e_{i,k} - a_{i,k} + D_i$ (see Section 3 for the notation end time $e_{i,k}$, arrival time $a_{i,k}$, and deadline D_i).

To compute the $f(A, \Gamma^\delta, ns)$ fitness value, SAFE runs SafeScheduler ns times for A and obtains ns schedule scenarios $S_1, S_2, \ldots, S_{ns}$. For each schedule scenario S_h, we denote by $dist_h(\tau_i, k)$ the distance between the end and deadline time values corresponding to the kth arrival of the task τ_i observed in S_h. We denote by $lk(\tau_i)$ the last arrival index of a task τ_i in A. SAFE aims to maximise the $f(A, \Gamma^\delta, ns)$ fitness function defined as follows:

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

We note that soft deadline tasks also require to execute within reasonable execution time ranges. Hence, engineers also estimate safe WCET ranges for soft deadline tasks. As the above fitness

Table 1. An example operation of SafeCrossover. It swaps all task arrivals of task τ_1 and τ_2 between two parent solutions A_p and A_q to produce offspring A'_p and A'_q.

	Task τ_1	Task τ_2	Task τ_3
Parent A_p	$(\tau_1, 5)$, $(\tau_1, 11)$	$(\tau_2, 8)$, $(\tau_2, 16)$	$(\tau_3, 4)$, $(\tau_3, 10)$
Parent A_q	$(\tau_1, 3)$, $(\tau_1, 7)$, $(\tau_1, 14)$	$(\tau_2, 6)$, $(\tau_2, 13)$	$(\tau_3, 5)$, $(\tau_3, 8)$, $(\tau_3, 13)$
Child A'_p	$(\tau_1, 3)$, $(\tau_1, 7)$, $(\tau_1, 14)$	$(\tau_2, 6)$, $(\tau_2, 13)$	$(\tau_3, 4)$, $(\tau_3, 10)$
Child A'_q	$(\tau_1, 5)$, $(\tau_1, 11)$	$(\tau_2, 8)$, $(\tau_2, 16)$	$(\tau_3, 5)$, $(\tau_3, 8)$, $(\tau_3, 13)$

function returns a quantified degree of deadline misses, SAFE uses such function for both soft and hard deadline tasks.

Computational search. SAFE employs a steady-state genetic algorithm [47]. The algorithm breeds a new population for the next generation after computing the fitness of a population. The breeding for generating the next population is done by using the following genetic operators: (1) *Selection.* SAFE selects candidate solutions using a tournament selection technique, with the tournament size equal to two which is the most common setting [31]. (2) *Crossover.* Selected candidate solutions serve as parents to create offspring using a crossover operation. (3) *Mutation.* The offspring are then mutated. Below, we describe our crossover and mutation operators.

Crossover. A crossover operator is used to produce offspring by mixing traits of parent solutions. SAFE modifies the standard one-point crossover operator [47] as two parent solutions A_p and A_q may have different sizes, i.e., $|A_p| \neq |A_q|$. Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a set of tasks to be scheduled. Our crossover operator, named SafeCrossover, first randomly selects an aperiodic task $\tau_i \in \Gamma$. For all $j \in [1, i]$ and $\tau_j \in \Gamma$, SafeCrossover then swaps all τ_j arrivals between two solutions A_p and A_q. As the size of Γ is fixed for all solutions, SafeCrossover can cross over two solutions that may have different sizes.

Table 1 shows an example operation of SafeCrossover using a system with three aperiodic tasks, τ_1, τ_2, and τ_3. Let two parent solutions A_p and A_q be as follows: $A_p = \{(\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 10)\}$ and $A_q = \{(\tau_1, 3), \ldots, (\tau_2, 6), \ldots, (\tau_3, 13)\}$, where (τ_i, t) denotes task τ_i arrives at time t. Given the two parents A_p and A_q, SafeScheduler randomly selects a task, i.e., τ_2 in this example, then it swaps all arrivals of τ_1 and τ_2 between A_p and A_q. As shown in Table 1, SafeCrossover then generates the offspring A'_p and A'_q as follows: $A'_p = \{(\tau_1, 3), \ldots, (\tau_2, 6), \ldots, (\tau_3, 10)\}$ and $A'_q = \{(\tau_1, 5), \ldots, (\tau_2, 8), \ldots, (\tau_3, 13)\}$. The shaded (resp. unshaded) cells in Table 1 indicate which task arrivals in child A'_q (resp. A'_p) come from which parent.

Mutation operator SAFE uses a heuristic mutation algorithm, named SafeMutation. For a solution A, SafeMutation mutates the kth task arrival time $a_{i,k}$ of an aperiodic task τ_i with a mutation probability. SafeMutation chooses a new arrival time value of $a_{i,k}$ based on the $[T_i^{min}, T_i^{max}]$ inter-arrival time range of τ_i. If such a mutation of the kth arrival time of τ_i does not affect the validity of the k+1th arrival time of τ_i, the mutation operation ends. Specifically, let $a^*_{i,k}$ be a mutated value of $a_{i,k}$. In case $a_{i,k+1} \in [a^*_{i,k} + T_i^{min}, a^*_{i,k} + T_i^{max}]$, SafeMutation returns the mutated A solution.

After mutating the kth arrival time $a_{i,k}$ of a task τ_i in a solution A, if the k+1th arrival becomes invalid, SafeMutation 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 $a^*_{i,k}$, SafeMutation 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. SafeMutation 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 Table 1, SafeMutation, for example, mutates a child solution $A'_p = \{(\tau_1, 3), (\tau_1, 7), (\tau_1, 14), \ldots, (\tau_3, 10)\}$. Let $[T_1^{min}, T_1^{max}] = [2, 8]$ be the inter-arrival time range of task τ_1, $\mathbb{T} = [0, 18]$ be the time period during which SafeScheduler receives task arrivals, and SafeMutation selects the second arrival of task τ_1, i.e., $(\tau_1, 7)$ in Table 1, to mutate. Based on the inter-arrival time range of τ_1, SafeMutation randomly chooses a new arrival time, e.g., 5, for the second arrival of τ_1. The third arrival $(\tau_1, 14)$ of τ_1 then became invalid due to the mutated second arrival $(\tau_1, 5)$; i.e., τ_1 cannot arrive at time 14 because $14 \notin [5 + 2, 5 + 8]$, 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 $(\tau_1, 12)$ as $12 = 14 + (5 - 7)$, where 14, 5, and 7 are, respectively, the original thrid arrival time of τ_1, the original second arrival time of τ_1, and the mutated second arrival time of τ_1. As SafeScheduler can receive new arrivals of τ_1 after time 12, SafeMutation may add new arrivals of τ_1 based on the inter-arrival time range of τ_1.

We note that when a system is only composed of periodic tasks, SAFE will skip searching for worst-case arrival sequences as arrivals of periodic tasks are deterministic (see Section 3), but will nevertheless generate the labelled dataset described below. When needed, SAFE can be easily extended to manipulate varying offset (and period) values for periodic tasks, in a way identical to how we currently handle inter-arrival times.

Labelled dataset. SAFE infers safe WCET ranges using a supervised learning technique [62] which requires a labelled dataset, namely logistic regression. In our context, a supervised learning technique creates a model that correlates tasks' WCET values with schedulability results indicating whether these tasks meet their deadlines or not. Supervised learning is conducted based on pairs of tasks' WCET values and a schedulability result, i.e., a labelled dataset. Specifically, SAFE uses logistic regression because it allows engineers to have probabilistic interpretation of safe WCET ranges and to investigate trade-off relationships among different tasks' WCETs. Section 4.2 describes this learning process in detail.

Recall from the fitness computation described above, SAFE runs SafeScheduler *ns* times to obtain schedule scenarios $S=\{S_1, S_2, \ldots, S_{ns}\}$, and then computes a fitness value of a solution A based on S. We denote by W_h a set of tuples (τ_i, C_i) representing that a task τ_i has the C_i WCET value in the S_h schedule scenario. Let $\vec{L}$ be a labelled dataset to be created by the first phase of SAFE. We denote by ℓ_h a label indicating whether or not a schedule scenario S_h has any deadline miss for any of the target tasks in Γ^{δ}, i.e., ℓ_h is either *safe* or *unsafe* which denotes, respectively, no deadline miss or deadline miss. For each fitness computation, SAFE adds *ns* number of tuples (W_h, ℓ_h) to $\vec{L}$. Specifically, for a schedule scenario S_h, SAFE adds $(W_h, unsafe)$ to $\vec{L}$ if there are $\tau_i \in \Gamma^{\delta}$ and $k \in [1, lk(i)]$ such that $dist_h(\tau_i, k) > 0$; otherwise SAFE adds $(W_h, safe)$ to $\vec{L}$.

4.2 Phase 2: Safe ranges of WCET

In Phase 2, SAFE computes safe ranges of WCET values under which target tasks are likely to be schedulable. To do so, SAFE applies a supervised machine learning technique to the labelled dataset generated by Phase 1 (Section 4.1). Specifically, Phase 2 executes SafeRefinement (Algorithm 1) which has following steps: complexity reduction, imbalance handling and model refinement.

Complexity reduction. The "reduce complexity" step in Algorithm 1 reduces the dimensionality of a labelled dataset $\vec{L}$ obtained from the first phase of SAFE (line 2). It predicts initial safe WCET ranges based on the WCET variables for the tasks in Γ (line 3) that have the most significant effect on deadline misses for target tasks. A labelled dataset $\vec{L}$ obtained from the first phase of SAFE contains tuples (W, ℓ) where W is a set of WCET values for tasks in Γ and ℓ is a label of W indicating either no deadline miss (safe) or deadline miss (unsafe) (Section 4.1). Note that some WCET values in W may not be relevant to determine ℓ. Hence, $\vec{L}$ may contain irrelevant variables to predict ℓ.

Algorithm 1: SafeRefinement. An algorithm for computing safe WCET ranges under which target tasks are schedulable. The algorithm consists of three steps as follows: "reduce complexity", "handle imbalanced dataset", and "refine model" steps.

Input: - $\vec{L}$: Labelled dataset obtained from the SAFE search
- G: Worst solutions obtained from the SAFE search
- ns: Number of WCET samples per solution
- nl: Number of logistic regression models
- pt: Precision threshold

Output: - m: Safe WCET model
- p: Probability of deadline misses

1: //step 1. reduce complexity
2: $\vec{L}^r \leftarrow$ `ReduceDimension(`$\vec{L}$`)` //feature reduction
3: $m \leftarrow$ `StepwiseRegression(`$\vec{L}^r$`)` //term selection
4: $p \leftarrow$ `Probability(`m, $\vec{L}^r$`)`
5: //step 2. handle imbalanced dataset
6: $\vec{L}^b \leftarrow$ `HandleImbalance(`$\vec{L}^r$, m`)`
7: //step 3. refine model
8: **for** nl times **do**
9: //step 3.1. add new data instances
10: **for** each solution $A \in G$ **do**
11: $\{S_1, S_2, \ldots, S_{ns}\} \leftarrow$ `RunSafeScheduler(`A, m, p, ns`)`
12: **for** each scenario $S_h \in \{S_1, S_2, \ldots, S_{ns}\}$ **do**
13: **if** S_h has any deadline miss **then**
14: $\vec{L}^b \leftarrow$ `Add(`$\vec{L}^b$, $(WCET(S_h), unsafe)$`)`
15: **else**
16: $\vec{L}^b \leftarrow$ `Add(`$\vec{L}^b$, $(WCET(S_h), safe)$`)`
17: **end if**
18: **end for**
19: **end for**
20: //step 3.2. learn regression model
21: $m \leftarrow$ `Regression(`m, $\vec{L}^b$`)`
22: $p \leftarrow$ `Probability(`m, $\vec{L}^b$`)`
23: **if** `PrecisionByCrossValidation(`$m, \vec{L}^b$`)` $> pt$ **then**
24: **break**
25: **end if**
26: **end for**
27: **return** m, p

To decrease computational complexity for the remaining steps, SafeRefinement creates a reduced dataset $\vec{L}^r$ which contains the same number of data instances (tuples) as $\vec{L}$ while including only WCET values with a significant effect on ℓ. To that end, SafeRefinement employs a standard feature reduction technique: random forest feature reduction [16] which has been successfully applied to high-dimensional data [39, 59]. Given the labelled dataset $\vec{L}$, random forest creates a set of decision trees based on the parameter values such as the number of trees and tree depth. Decision trees obtained by random forest allow us to rank features, i.e., task WCETs, based on their importance as

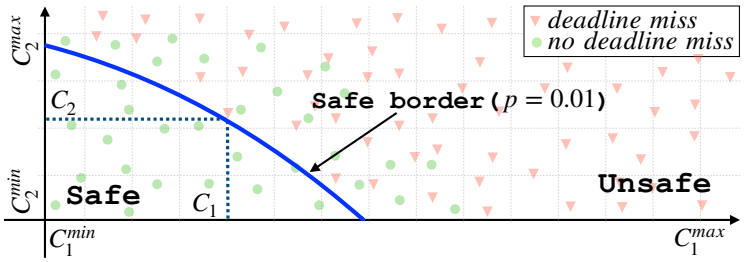


Fig. 3. A safe border line of WCET values for the τ_1 and τ_2 tasks. The safe border is determined by a deadline miss probability of 0.01. C_1 and C_2 determine safe WCET ranges of τ_1 and τ_2 under which they likely satisfy their deadlines.

measured by Gini impurity [16]. Hence, by setting a particular threshold for importance, we can select a subset of the features. Note that Section 5.6 describes the parameter values for the feature reduction step in detail.

After reducing the dimensionality of the input dataset $\vec{L}$ in Algorithm 1, resulting in the reduced dataset $\vec{L}^r$, SafeRefinement learns an initial model to predict safe WCET ranges. SafeRefinement uses logistic regression [41] because it enables a probabilistic interpretation of safe WCET ranges and the investigation of relationships among different tasks' WCETs. For example, Figure 3 shows a *safe border* determined by an inferred logistic regression model m with a probability p of deadline misses. Note that a safe range, e.g., $[C_1^{min}, C_1]$ of task τ_1 in Figure 3, is determined by a point on the safe border in a multidimensional WCET space. A safe border distinguishes safe and unsafe areas in the WCET space. After inferring a logistic regression model m from the input dataset, SafeRefinement selects a probability p maximising the safe area under the safe border determined by m and p while ensuring that all the data instances, i.e., sets of WCET values, classified as safe using the safe border are actually observed to be safe in the input dataset, i.e., no false positives (lines 3–4). We note that engineers can also select an adequate probability, which may yield false positives or not maximise the area under the safe border, depending on their needs.

SafeRefinement uses a second-order polynomial response surface model (RSM) [56] to build a logistic regression model. RSM is known to be useful when the relationship between several explanatory variables (e.g., WCET variables) and one or more response variables (e.g., safe or unsafe label) needs to be investigated [48, 56]. RSM contains linear terms, quadratic terms, and 2-way interactions between linear terms. Let V be a set of WCET variables v_x in $\vec{L}^r$. Then, the logistic regression model of SafeRefinement is defined as follows:

$$\log \frac{p}{1-p} = c_0 + \sum_{x=1}^{|V|} c_x v_x + \sum_{x=1}^{|V|} c_{xx} v_x^2 + \sum_{y>x} c_{xy} v_x v_y$$

As shown in the above equation, an RSM equation, i.e., the right-hand side, built on the reduced dataset $\vec{L}^r$ has a higher number of dimensions, i.e., the number of coefficients to be inferred, than $|V|$ as RSM additionally accounts for quadratic terms (v_x^2) and 2-way interactions ($v_x v_y$) between linear terms. Hence, SafeRefinement employs a stepwise regression technique (line 3), e.g., stepwise AIC (Akaike Information Criterion) [75], in order to select significant explanatory terms from the RSM equation. This allows the remaining "refine model" step of SafeRefinement to execute efficiently as it requires to run SafeScheduler and logistic regression multiple times within a time budget (line 8), both operations being computationally expensive.

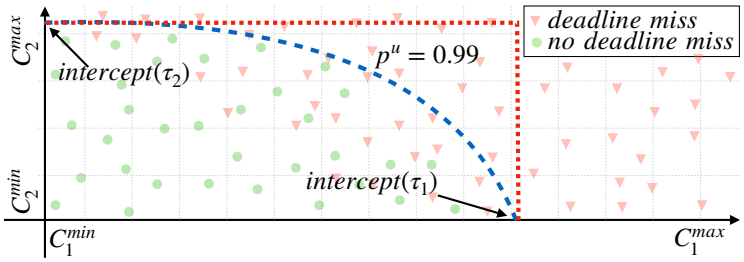


Fig. 4. Handling imbalanced dataset by excluding unsafe WCET values based on logistic regression intercepts.

Imbalance handling. Recall from Section 4.1 that SAFE searches for worst-case sequences of task arrivals and is guided by maximising the magnitude of deadline misses, when they are possible. Therefore, the major portion of $\vec{L}$, the dataset produced by the first phase of SAFE, is a set of task arrival sequences leading to deadline misses. Supervised machine learning techniques (including logistic regression) typically produce unsatisfactory results when faced with highly imbalanced datasets [11]. SafeRefinement addresses this problem with the "handle imbalanced dataset" step in Algorithm 1 (lines 5–6) before refining safe WCET ranges. SafeRefinement aims to identify WCET ranges under which tasks are likely to be schedulable. This entails that WCET ranges under which tasks are highly unlikely to be schedulable can be safely excluded from the remaining analysis. Specifically, SafeRefinement prunes out WCET ranges with a high probability of deadline misses above a high threshold p^u and thus creates a more balanced dataset $\vec{L}^b$ compared to the original imbalanced dataset $\vec{L}^r$ (line 6). SafeRefinement automatically finds a minimum probability p^u which leads to a safe border classifying no false unsafe (negative) instances in $\vec{L}^r$. SafeRefinement then updates the maximum WCET C_i^{max} of a task τ_i based on the intercept of the logistic regression model m (with a probability of p^u) on the WCET axis for τ_i. Figure 4 shows an example dataset $\vec{L}^r$ with a safe border characterised by a high deadline miss probability, i.e., $p^u = 0.99$, to create a more balanced dataset $\vec{L}^b$ within the restricted ranges $[C_1^{min}, intercept(\tau_1)]$ and $[C_2^{min}, intercept(\tau_2)]$.

Model refinement. The "refine model" step in Algorithm 1 refines an inferred logistic regression model by sampling additional schedule scenarios selected according to a strategy that is expected to improve the model. As described in Section 4.1, the SAFE search produces a set G (population) of worst-case arrival sequences of tasks Γ which likely violate deadline constraints of target tasks $\Gamma^\delta \subseteq \Gamma$. For each arrival sequence A in G, SafeRefinement executes SafeScheduler ns times to add ns new data instances to the dataset $\vec{L}^b$ based on the generated schedule scenarios and their schedulability results (lines 9–19). After adding $ns \cdot |G|$ new data instances to $\vec{L}^b$, SafeRefinement runs logistic regression again to infer a refined logistic regression model m and computes a probability p that ensures no false safe instances (positives) in $\vec{L}^b$ and maximises the safe area under the safe border defined by m and p (lines 20–25).

In the second phase of SAFE, SafeScheduler selects WCET values for tasks in Γ to compute a schedule scenario based on a distance-based random number generator, which extends the standard uniform random number generator. The distance-based WCET value sampling aims at minimising the Euclidean distance between the sampled WCET points and the safe border defined by the inferred model m and the selected probability p. SafeScheduler iteratively computes new WCET values using the following distance-based sampling procedure: (1) generating nr random samples in the WCET space, (2) computing their distance values from the safe border, and (3) selecting the closest point to the safe border.

SafeRefinement stops model refinements either by reaching an allotted analysis budget (line 8 of Algorithm 1) or when a precision reaches an acceptable level *pt*, e.g., 0.99 (lines 23–25). SafeRefinement uses the standard precision metric [71] as described in Section 5.5. In our context, practitioners need to identify safe WCET ranges at a high level of precision to ensure that identified safe WCET ranges can be trusted. To compute a precision value, SafeRefinement uses a standard k-fold cross-validation [71]. In k-fold cross-validation, $\overrightarrow{L}^b$ is partitioned into k equal-size splits. One split is retained as a test dataset, and the remaining k-1 splits are used as a training dataset. The cross-validation process is then repeated k times to compute a precision of inferred safe borders which are determined by a logistic regression model m and a probability p (lines 21 and 22)

Selecting WCET ranges. A safe border defined by an inferred logistic regression model and a deadline miss probability of p represents a (possibly infinite) set of points, corresponding to safe WCET ranges of tasks, e.g., $[C_1^{min}, C_1]$ and $[C_2^{min}, C_2]$ in Figure 3. In practice, however, engineers need to choose a specific WCET range for each task to conduct further analysis and development. How to choose optimal WCET ranges depends on the system context. At early stages, however, such contextual information may not be available. Hence, SAFE proposes a *best-size point*, i.e., WCET ranges, on a safe border which maximises the volume of the hyperbox the point defines. In general, the larger hyperbox, the greater flexibility the engineers have in selecting appropriate WCET values. Choosing the point with the largest volume is helpful when no domain-specific information is available to define other selection criteria. In general the inferred safe border enables engineers to investigate trade-off among different tasks' WCET values.

5 EVALUATION

We evaluate SAFE using an industrial case study from the satellite domain. Our full evaluation package is available online [43].

5.1 Research Questions

RQ1 (baseline comparison): *How does SAFE perform compared with a baseline approach?* With RQ1, we investigate whether SAFE can outperform WCET estimation based on random search. Note that such RQ is an important *sanity check* for search-based solutions in general [6, 36]. Our conjecture is that SAFE, although computationally expensive, will significantly outperform a random search solution with respect to estimating safe WCET ranges with a higher degree of confidence.

RQ2 (effectiveness of distance-based sampling): *How does SAFE, based on distance-based sampling, perform compared with random sampling?* We compare our distance-based sampling procedure described in Section 4.2 and used in the second phase of SAFE with a naive random sampling. Our conjecture is that distance-based sampling, although expensive, is needed to improve the quality of the training data used for logistic regression. RQ2 assesses this conjecture by comparing distance-based and random sampling.

RQ3 (usefulness): *Can SAFE identify WCET ranges within which tasks are highly likely to satisfy their deadline constraints?* In RQ3, we investigate whether SAFE identifies acceptably safe WCET ranges in practical time. We further discuss our insights regarding the usefulness of SAFE from the feedback obtained from engineers in LuxSpace.

RQ4 (scalability): *Can SAFE find safe WCET ranges for large-scale systems with a practical time budget?* In this RQ, we study the relationship between the execution time of SAFE and the parameters of study subjects. We use several synthetic subjects to be able to freely control key real-time systems' parameters.

Table 2. Description of the three industrial subject systems: number of periodic and aperiodic tasks, resource dependencies, and platform cores. The full task descriptions are available online [43].

System	Periodic tasks	Aperiodic tasks	Dependencies	Cores
ADCS	15	19	0	1
ICS	3	3	3	3
UAV	12	4	4	3

5.2 Industrial Study Subjects

We evaluated SAFE by applying it to our motivating case study subject, i.e., the satellite attitude determination and control system (ADCS) described in Section 2, as well as two industrial study subjects from the literature [61, 65]. Table 2 summarizes the relevant attributes of these subjects, presenting the number of periodic and aperiodic tasks, resource dependencies, and processing cores. The subjects are characterized by real-time parameters, e.g., priorities, WCETs, periods and deadlines, described in Section 3. The full task descriptions of the subjects are available online [43]. The main missions of the three subjects are describe as follows:

- ADCS is a satellite system that aims at orienting a satellite in a proper position on time to ensure that the satellite provides normal service correctly (see Section 2). LuxSpace, our industry partner, developed ADCS for an ESA project.
- ICS is an ignition control system that checks the status of an automotive engine and corrects any errors of the engine [61]. The system was developed by Bosch GmbH[1].
- UAV is a mini unmanned air vehicle that follows dynamically defined way-points and communicates with a ground station to receive instructions [65]. The system was developed in a collaboration with the University of Poitiers France and ENSMA[2].

LuxSpace is a leading system integrator of micro satellites and aerospace systems. ADCS includes a set of 15 periodic and 19 aperiodic tasks. Eight tasks out of the 19 aperiodic tasks are constrained by hard deadlines, i.e., sporadic tasks. Out of the 34 tasks, engineers provided single WCET values for eight tasks. For the remaining 26 tasks, engineers estimated WCET ranges due to uncertain decisions, e.g., implementation choices and hardware specifications, made at later development stages (see Section 2). The differences between the estimated WCET maximum and minimum values across the 26 tasks varies from 0.1ms to 20000ms. Our collaboration with LuxSpace enabled us to discuss SAFE results with engineers to draw important qualitative conclusions and to assess the benefits of SAFE (see Section 5.7).

For the experiments with ICS and UAV, we used the task descriptions reported in a previous study [26] and modified their tasks' WCETs from point values to ranges. Though the problem of schedulability analysis of real-time tasks has been widely studied [3, 15, 26, 35, 69], none of the prior work addresses the same problem (see Section 3) as that addressed by SAFE. Hence, the public study subjects in the literature do not fit our study's requirements. In particular, none of the public real-time system case studies [26] contains estimated WCET ranges in their task descriptions. These ranges, however, are necessary to apply SAFE and to evaluate its effectiveness. In order to evaluate SAFE in various and realistic system contexts, we chose to apply SAFE to existing industrial subjects, i.e., ICS and UAV, described in prior work [26] and made necessary changes

[1]Bosch GmbH: https://www.bosch.com/
[2]ENSMA: https://www.ensma.fr/

only to task WCETs of the subjects as described below. Compared to ADCS, ICS and UAV have different task characteristics, such as resource dependencies and number of processing cores.

We note that estimating (practically valid) WCET ranges requires significant domain expertise. For public domain case study systems such as ICS and UAV, however, we do not have any access to the engineers who have developed those subjects. Hence, we chose to apply a simple and straightforward method to convert a point WCET value to a WCET range as follows: (Step 1) We first check whether the system under analysis is schedulable or not. For a task τ_i in the system, we denote by C_i an original point WCET value of τ_i. (Step 2) If the system is evaluated to be schedulable, it indicates that the system's tasks may be able to handle higher execution times than their estimated WCETs. Hence, we simply define the WCET range of τ_i by $[C_i,\ r{\cdot}C_i]$, where $r > 1$, as input WCET ranges for SAFE. This modification enables SAFE to find more relaxed safe WCET ranges. (Step 2′) Otherwise, if the system is evaluated to be unschedulable, we define the WCET range of τ_i by $[r'{\cdot}C_i,\ C_i]$, where $r' < 1$, as input for SAFE. This modification allows SAFE to find appropriate WCET estimates, ensuring the system is likely to be schedulable under the WCET ranges found by SAFE. As ICS and UAV are likely to be schedulable [26], for all task τ_i in ICS and task τ_j in UAV, we created the modified task descriptions of ICS and UAV based on Step 2. We conducted experiments using simulations to set the r values for ICS and UAV by configuring r to 1.1, 1.2, ..., 1.5 incrementally until we could find deadline misses in each system, i.e., unsafe WCET values. Recall from Section 4.2 that SAFE relies on logistic regression to partition the given WCET ranges into safe and unsafe sub-ranges for a selected deadline miss probability. Hence, we modified the estimated WCET ranges of ICS and UAV to include both safe and unsafe WCET ranges. Given the experiment results, we set the WCET ranges of ICS and UAV to $[C_i, 1.2{\cdot}C_i]$ and $[C_j, 1.5{\cdot}C_j]$, respectively. The full original and modified task descriptions of ICS and UAV are available online [43].

5.3 Synthetic study subjects

We evaluated the scalability of SAFE using synthetic systems, following the common scalability analysis practice applied in many real-time system studies [23, 25, 28, 32, 70, 78]. As shown in Algorithm 2, we synthesise a set of real-time tasks by varying key task parameters as described below. The algorithm first synthesises a set of periodic tasks (lines 2-8) and then converts some of these tasks to aperiodic tasks (lines 9-10). Last, the algorithm configures some tasks with WCET ranges (lines 11-12).

As shown on line 3 of Algorithm 2, the algorithm first creates a set **U** of task utilisation values by using the UUniFast-Discard algorithm [23] that is devised to give an unbiased distribution of task utilisation values. The UUniFast-Discard algorithm takes as input the number of tasks to be synthesised, n, and a target utilisation value, u^t. It then outputs n utilization values, $U_1 \ldots U_n$, where $0 < U_i < 1$ for all U_i and $\sum_{i=1}^{n} U_i = u^t$.

As for line 4 in Algorithm 2, 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., given a task period (random variable) T_i, $log\ T_i$ 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 period ranges [10ms, 100ms] and [100ms, 1000ms]. The parameter g is used to determine the granularity of period values as multiples of g. Lines 5-7 of Algorithm 2 describe how the algorithm synthesises tasks' WCET values. Specifically, for each task τ_i, the algorithm computes the WCET value C_i of τ_i as $C_i = U_i \cdot T_i$.

Given the task periods **T** and the WCET values **C**, line 8 of Algorithm 2 synthesizes a set Γ of periodic tasks accounting for offsets, priorities, and deadlines. A periodic task τ_i is characterised by a period T_i, a WCET C_i, an offset O_i, a priority P_i, and a deadline D_i (see Section 3). A task offset O_i is randomly selected from an input range $[0, \theta]$ of offset values. The algorithm relies on the

Algorithm 2: An algorithm for creating a synthetic subject while accounting for the task characteristics described in Section 3.

Input: - n: number of tasks
- u^t: target utilisation
- T^{min}: minimum task period
- T^{max}: maximum task period
- g: granularity of task periods
- θ: maximum offset value
- γ: ratio of aperiodic tasks
- μ: range factor to determine inter-arrival times
- ω: number of WCET ranges
- λ: range factor to determine WCET ranges

Output: - Γ: set of tasks

1: $\Gamma \leftarrow \{\}, \mathbf{C} \leftarrow \{\}$
2: //synthesise a set of periodic tasks
3: $\mathbf{U} \leftarrow$ `UUniFast_discard`(n, u^t)
4: $\mathbf{T} \leftarrow$ `generate_task_periods`(n, T^{min}, T^{max}, g) //task periods
5: **for** each $i \in [1, n]$ **do**
6: $\quad \mathbf{C} \leftarrow \mathbf{C} \cup \{U_i \cdot T_i\}$, where $U_i \in \mathbf{U}$ and $T_i \in \mathbf{T}$ //WCETs
7: **end for**
8: $\Gamma \leftarrow$ `generate_task_set`$(\mathbf{T}, \mathbf{C}, \theta, g)$
9: //convert some periodic tasks to aperiodic tasks
10: $\Gamma \leftarrow$ `convert_to_aperiodic_tasks`(Γ, γ, μ)
11: //convert some WCET point values to WCET ranges
12: $\Gamma \leftarrow$ `convert_to_WCET_ranges`$(\Gamma, \omega, \lambda)$
13: **return** Γ

rate-monotonic scheduling policy [44] to decide task priorities and deadlines. Specifically, tasks with shorter periods are given higher priorities and tasks' deadlines are equal to their periods.

Line 10 of Algorithm 2 synthesises aperiodic tasks. The algorithm converts some periodic tasks into aperiodic tasks according to a ratio γ of aperiodic tasks among all tasks. The algorithm then uses a range factor μ to determine minimum and maximum inter-arrival times of 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, $[T_i^{min}, T_i^{max}] = [27.5, 72.5]$.

To synthesise tasks' WCET ranges, line 12 of Algorithm 2 randomly selects ω tasks in Γ to convert their WCET point values to WCET ranges. For a selected task τ_i, 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 λ is a range factor to determine the WCET ranges and $\lambda \in (0, 1)$. For example, if $\lambda = 0.25$ and $C_i = 10$ for a task τ_i, $[C_i^{min}, C_i^{max}] = [7.5, 12.5]$.

5.4 Experimental Setup

To answer RQ1, RQ2, and RQ3 described in Section 5.1, we rely on case study data pertaining to ADCS, provided by LuxSpace, as well as the ICS and UAV subjects described in Section 5.2. To

answer RQ4, we used 800 synthetic subjects (see Section 5.3). We conducted four experiments, EXP1, EXP2, EXP3, and EXP4, as described below.

EXP1. To answer RQ1, we developed a baseline solution that estimates task WCETs based on random search (RS). The baseline replaces the GA in Phase 1 with RS and does not infer a safe border using logistic regression. Note that the baseline uses the same fitness function (see Section 4.1) and also maintains the best population during search; however, it does not employ any genetic operators, i.e., crossover and mutation. The baseline solution also produces a labelled dataset $\overrightarrow{L}$ that contains tuples (W, ℓ) where W is a set of task WCETs and ℓ is a label of W indicating either safe or unsafe (see Section 4.1). Given the labelled dataset, the baseline selects the best task WCETs that are safe and maximise the volume of the hyperbox they define. Specifically, the baseline finds a particular tuple (W_s, ℓ_s) in $\overrightarrow{L}$ that maximises the volume of the hyperbox defined by W_s while satisfying the following condition: For all tuples (W_x, ℓ_x) in $\overrightarrow{L}$, the hyperbox defined by W_x is contained in the hyperbox defined by W_s, $\ell_x = \mathit{safe}$, and $\ell_s = \mathit{safe}$.

EXP1 compares the results obtained from executing SAFE and the baseline. For comparison, SAFE selects a best-size point, i.e., WCET ranges, on a safe border that maximises the volume of the hyperbox the point defines (see Section 4.2). Given two solutions, i.e., estimated WCET ranges, obtained by SAFE and the baseline, EXP1 checks the schedulability of the two solutions using simulations. To do so, we ran simulations multiple times by varying task arrivals and task execution times within their estimated WCET ranges and checked whether there was a deadline miss in each simulation result.

EXP2. To answer RQ2, EXP2 compares our distance-based WCET sampling technique (described in Section 4.2) with the naive random WCET sampling technique, for the second phase of SAFE. To this end, EXP2 first creates an initial training dataset by running the first phase of SAFE. EXP2 then relies on this initial training data for model refinement (Section 4.2) by using both distance-based and naive random sampling. For comparison, EXP2 creates a test dataset by randomly sampling WCET values, which is independently created from the second phase of SAFE, and then compares the accuracy of the two sampling approaches in identifying safe WCET ranges for the test dataset.

EXP3. To answer RQ3, EXP3 computes precision values for SAFE, obtained from 10-fold cross-validation (see Section 4.2), over each model refinement for the ADCS subject. We note that EXP3 focuses on ADCS to evaluate the practical usefulness of SAFE as we do not have any access to the engineers who have developed the other study subjects, i.e., ICS and UAV. In our study context, i.e., developing safety-critical systems, engineers require very high precision, i.e., ideally no false positives, (see Section 4). Hence, EXP3 measures precision over model refinements to align with such practice. EXP3 then measures whether SAFE can compute safe WCET ranges within practical execution time and at an acceptable level of precision.

EXP4. To answer RQ4, EXP4 measures the execution time of SAFE with 800 synthetic systems. We use the task generation algorithm described in Section 5.3 to create synthetic systems. In order to conduct controlled experiments to study correlations between the execution time of SAFE and a particular system parameter (e.g., number of tasks), we first create a baseline synthetic system by setting the parameters of Algorithm 1 as follows: (1) We set the number of tasks n to 20, the ratio of aperiodic tasks γ to 0.45, and the maximum offset θ to 0. Note that these parameter values are the average values of the parameters in our industrial subjects. (2) With regard to task periods, we set the range $[T^{min}, T^{max}]$ of minimum and maximum periods to [10ms, 1s], which are common values in many real-time subjects [10]. The granularity of task periods g is set 10ms in order to increase realism as most of the task periods in our industrial subjects are multiples of 10ms. (3) For the range factor to determine inter-arrival times for aperiodic tasks μ, the number of WCET ranges ω, the range factor to determine WCET ranges λ, and the target utilisation per processing core u^t,

we assign μ = 0.25, ω=2, λ = 0.25, and u^t = 0.9, respectively. We set these parameter values based on initial experiments to ensure that the executions of the synthetic systems examined in EXP4 sometimes violate their deadlines. Recall from Section 4 that SAFE relies on logistic regression and a labelled dataset, containing both safe (positive) and unsafe (negative) data instances. (4) We set the number of processing cores ϵ to 1 as a baseline. (5) For the simulation time of SafeScheduler (see Section 4.1), we assign 30s in order to ensure that any aperiodic task arrives at least once and all possible arrivals of periodic tasks are analysed during that time.

Given the baseline system, we create several synthetic systems to be examined in EXP4 by varying the parameters' values as follows: (1) number of tasks, $n \in \{5, 10, ..., 50\}$, (2) ratio of aperiodic tasks, $\gamma \in \{0.05, 0.1, ..., 0.5\}$, (3) range factor to determine inter-arrival times for aperiodic tasks, $\mu \in \{0.05, 0.1, ..., 0.5\}$, (4) number of WCET ranges, $\omega \in \{1, 2, ..., 10\}$, (5) range factor to determine WCET ranges, $\lambda \in \{0.05, 0.1, ..., 0.5\}$, (6) maximum offset value $\theta \in \{200ms, 400ms, ..., 2000ms\}$, (7) number of processing cores, $\epsilon \in \{1, 2, ..., 10\}$, and (8) simulation time, $\mathbf{t} \in \{30s, 1m, ..., 5m\}$. Note that we chose to study the effect of these parameters because they are controlled by engineers to design tasks in real-time systems. Simulation time $\mathbf{t}$ obviously impacts the execution time of SAFE as well. Resource dependencies are not controlled when generating synthetic systems as they do not impact SAFE's searching and learning (see Section 4) but only simulations, which are investigated by varying simulation time. Due to the degree of randomness in our approach to generating synthetic systems (see Section 5.3), we create ten synthetic systems for each control parameter.

5.5 Metrics

We use the standard precision and recall metrics [71] to measure the accuracy in our experiments. To compute precision and recall in our context, for EXP2, we created a synthetic test dataset for each study subject containing tuples of WCET values and a flag indicating the presence or absence of deadline miss obtained from running SafeScheduler. Note that creating a test dataset by running an actual study subject with varying task WCETs is prohibitively expensive. We therefore used a set of task arrival sequences obtained from the first phase of SAFE for each subject as we aim at testing sequences of task arrivals which are more likely to violate their deadlines. We then ran SafeScheduler to simulate task executions for the set of task arrival sequences with randomly sampled WCET values. We note that WCET values were sampled within the restricted WCET ranges after the "handling imbalance" step in Algorithm 1. Parts of the WCET ranges under which tasks are unlikely to be schedulable are therefore not considered when sampling. For EXP3, we used 10-fold cross-validation based on the training dataset at each model refinement step (phase 2).

We define the precision and recall metrics as follows: (1) precision $P = TP/(TP+FP)$ and (2) recall $R = TP/(TP + FN)$, where TP, FP, and FN denote the number of true positives, false positives, and false negatives, respectively. A true positive is a test instance (a set of WCET values) labelled as safe and correctly classified as such. A false positive is a test instance labelled as unsafe but incorrectly classified as safe. A false negative is a test instance labelled as safe but incorrectly classified as unsafe. We prioritise precision over recall as practitioners require (ideally) no false positives – an unsafe instance with deadline misses is incorrectly classified as safe – in the context of mission-critical, real-time satellite systems. For EXP2, precision and recall values are measured based on a synthetic test dataset. For EXP3, precision values are computed using collective sets of true positives and false positives obtained from 10-fold cross-validation at each model refinement.

Due to the inherent degree of randomness in SAFE, we repeat our experiments 50 times. For EXP1, we ran 40000 simulations to check the schedulability of the solutions obtained by SAFE and the baseline. To statistically compare our results, we use the non-parametric Mann-Whitney U-test [49] and Vargha and Delaney's $\hat{A}_{12}$ effect size [66]. Mann-Whitney U-test determines whether

two independent samples are likely or not to belong to the same distribution. We set the level of significance, α, to 0.05. Vargha and Delaney's $\hat{A}_{12}$ measures probabilistic superiority – effect size – between search algorithms. Two algorithms are considered to be equivalent when the value of $\hat{A}_{12}$ is 0.5.

5.6 Implementation and Parameter Tuning

To implement the feature reduction step of Algorithm 1, we used the random forest feature reduction [16] as it has been successfully applied to high-dimensional data [39, 59]. For the stepwise regression step of Algorithm 1, we used the stepwise AIC regression technique [75] which has been used in many applications [52, 79]. Recall from Section 4.2 that our distance-based sampling and best-size region recommendation require a numerical optimisation technique to find the nearest WCET sample and a maximum safe region size based on an inferred safe border. For such optimisations, we applied a standard numerical optimisation method, i.e., the Nelder-Mead method [58].

To compute the GA fitness, we set the number of SafeScheduler runs (Section 4.1) for each solution (A in Section 4.1) to 20. This number was chosen based on our initial experiments. We observed that 20 runs of SafeScheduler per solution A keeps execution time under a reasonable threshold, i.e., <1.2m for all the subjects, and is sufficient to compute the fitness of SAFE. SafeScheduler schedules 34 tasks in ADCS for 1800s, 6 tasks in ICS for 150ms, and 16 tasks in UAV for 1500ms during which SafeScheduler advances its simulation clock by 0.1ms, 0.01ms, 0.01ms, respectively, for adequate precision. We chose the time periods to ensure that all the tasks in each subject can be executed at least once.

For the GA search parameters, we set the population size to 10, the crossover rate to 0.7, and the mutation rate to 0.2, which are consistent with existing guidelines [38]. We ran GA for 1000 iterations after which we observed that fitness reached a plateau in our initial experiments. Note that for the baseline comparison to be fair, we ran RS for 1500 iterations to ensure that the generated dataset $\vec{L}$ contained 30000 data instances, which are the same number of data instances obtained by SAFE.

Regarding the feature reduction step of Algorithm 1, we set the random forest parameters as follows: (1) the tree depth parameter is set to $\sqrt{|F|}$, where $|F|$ denotes the number of features, i.e., 26 WCET ranges in ADCS, 6 WCET ranges in ICS, and 16 WCET ranges in UAV, based on guidelines [37]. (2) The number of trees is set to 100 based on our initial experiments. We observed that learning more than 100 trees does not provide additional gains in terms of reducing the number of features.

Note that all the parameters mentioned above can probably be further tuned to improve the performance of SAFE. However, since with our current setting, we were able to convincingly and clearly support our conclusions, we do not report further experiments on tuning those parameters.

We ran our experiments over the high-performance computing cluster [67] at the University of Luxembourg. To account for randomness, we repeated each run of SAFE 50 times for all the experiments. Each run of SAFE was executed on a different node of the cluster. It took around 35h for us to create a synthetic test dataset with 50000 instances. When we set 1000 GA iterations for the first phase of SAFE and 10000 new WCET samples (100 refinements $\times$ 100 new WCET samples per refinement) for the second step of SAFE, each run of SAFE took at most 27.1h – phase 1: 16.361h and phase 2: 10.74h. The running time is acceptable as SAFE can be executed offline in practice.

Table 3. Comparing SAFE and our baseline method using (1) the volumes of the hyperboxes that are defined by the best-size points computed by each method, (2) the number of simulation runs that contain deadline misses out of total 40000 runs, and (3) the execution times of each method. The results are obtained from 50 runs of Safe and Baseline.

	SAFE			Baseline			p-value			$\hat{A}_{12}$		
	ADCS	ICS	UAV	ADCS	ICS	UAV	ADCS	ICS	UAV	ADCS	ICS	UAV
Best-size volumes	5.18e-11 ms^{26}	9.55e-01 ms^{6}	1.20e-02 ms^{16}	5.78e-12 ms^{26}	1.10e+00 ms^{6}	2.33e-04 ms^{16}	0.0000	0.0383	0.0000	**1.0000**	0.3796	**1.0000**
Deadline misses	**0.00**	**5.42**	**1.36**	114.92	13.2	32.32	0.0005	0.0023	0.0439	**0.3900**	**0.3394**	**0.04084**
Execution times	25.14 hours	1.37 hours	1.62 hours	24.29 hours	0.11 hours	0.31 hours	0.0000	0.0000	0.0000	0.8960	1.0000	1.0000

5.7 Experiment Results

RQ1. Table 3 compares SAFE and our baseline method (Baseline) using the following three metrics: (1) the volume of the hyperbox that is defined by the best-size point (see Sections 4.2 and 5.4) computed by each method, i.e., SAFE and Baseline, (2) the number of simulation runs, out of 40000 runs, that contain any deadline misses when tasks execute within their estimated WCET ranges defined by the best-size points, and (3) the execution time of SAFE and Baseline to estimate WCET ranges. The results presented in the table are the mean values obtained from 50 runs of SAFE and Baseline for each of the three subjects. To enable accurate comparisons, we ran 40000 simulations of each execution of SAFE and Baseline, aiming at evaluating their estimated WCET ranges as described in Section 5.4. Statistical comparisons of the results obtained from 50 runs of SAFE and Baseline are summarized using p-values and $\hat{A}_{12}$ values as described in Section 5.5.

As shown in Table 3, compared to Baseline, SAFE provides more relaxed WCET ranges for ADCS and UAV. Note that the larger the hyperbox, the greater flexibility the engineers have in selecting appropriate WCET values. For ICS, however, Baseline finds a larger hyperbox than the best-size hyperbox produced by SAFE. This is likely due to the fact that ICS has a small number of tasks and is therefore much simpler than the other two subjects. Further, we recall that SAFE also provides engineers with a probability of deadline misses and trade-off relations between task WCETs based on an inferred logistic regression model (see Section 4.2). We further discuss these benefits from a practitioner's perspective in RQ3.

EXP1 evaluates the estimated WCET ranges that are defined by the best-size points obtained from 50 runs of SAFE and Baseline for each subject. The estimated WCET ranges are examined through 40000 simulation runs by varying task arrivals and their execution times within the estimated WCET ranges. The "Deadline misses" row in Table 3 shows the mean number of simulation runs (out of 40000 runs) containing deadline misses. Across all the subjects, the differences between SAFE and Baseline are statistically significant as the p-values are less than 0.05. The $\hat{A}_{12}$ values show that SAFE is probabilistically superior to Baseline with respect to minimising the number of deadline misses.

Regarding the execution times of SAFE and Baseline, SAFE took more time than Baseline for all the subjects as shown in Table 3. Estimating safe WCET ranges for ADCS requires the largest execution time (on average, 25.14h) compared to the other subjects. We note that such execution time is acceptable as SAFE can be executed offline in practice.

Figure 5 shows probability distributions obtained from 50 runs of SAFE and simulations for ADCS, ICS, and UAV. As described in Section 4.2, SAFE partitions the given WCET ranges into safe and unsafe sub-ranges using a safe WCET border that is defined by an inferred logistic regression model

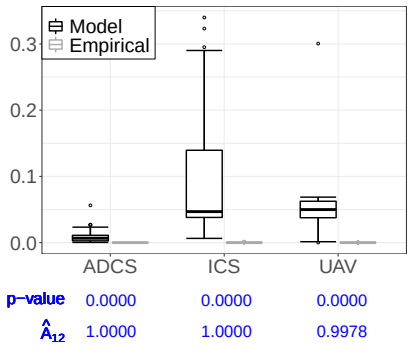
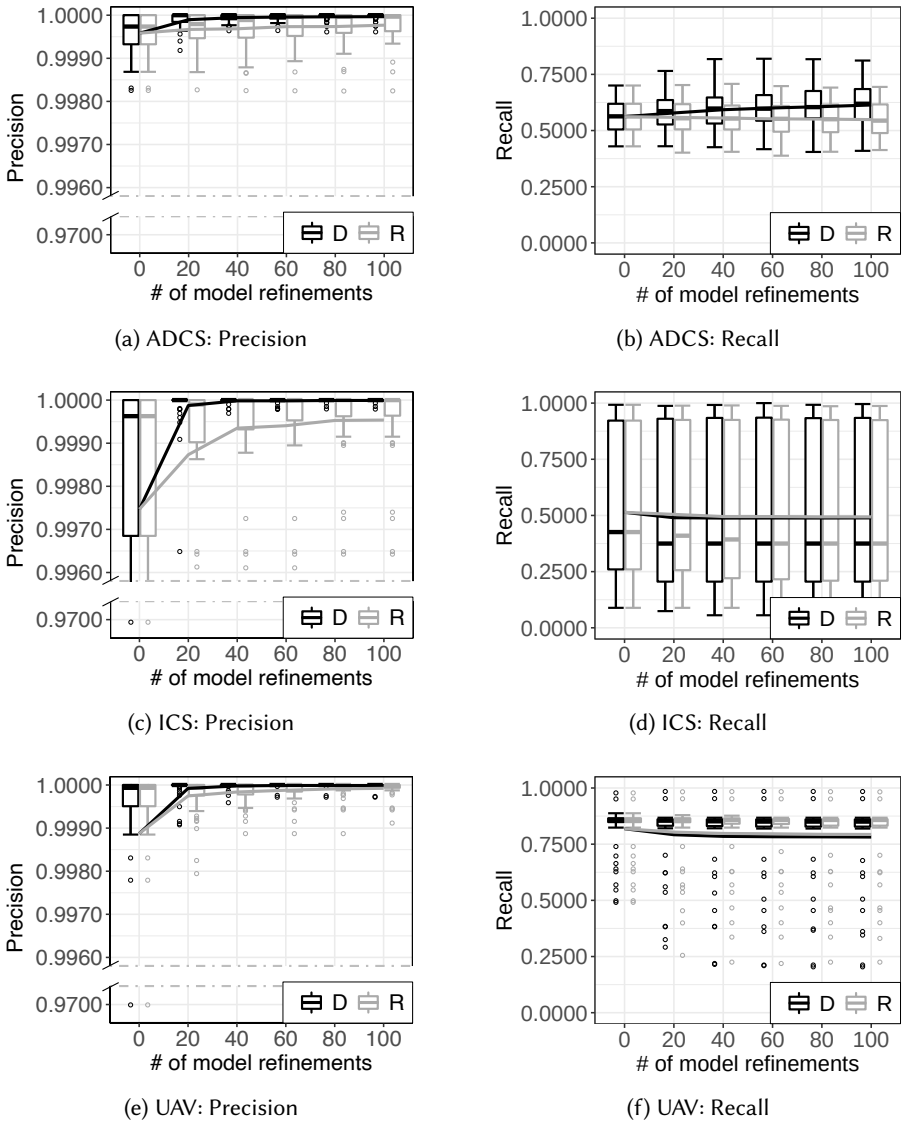


Fig. 5. Comparing probability values of deadline misses computed by SAFE and simulations for ADCS, ICS and UAV. SAFE uses a logistic regression model and selects a deadline miss probability to find the best-size WCET point. Empirical probability values of deadline misses are estimated based on 40000 simulations for each output, i.e., WCET ranges, of SAFE. The boxplots (20%-50%-75%) show probability values obtained from 50 runs of SAFE (see Model) and 50 simulation-based evaluations (see Empirical), i.e., 50 × 40000 simulation runs.

and a selected probability of deadline misses. In EXP1, SAFE selects a deadline miss probability that maximises the safe area under the safe border while ensuring that all the WCET points, i.e., sets of WCETs, classified as safe using the safe border are actually observed to be safe in the input dataset of logistic regression. The estimated WCET ranges, i.e., 50 best-size WCET points obtained by SAFE, are then evaluated through 40000 simulation runs for each WCET point by varying task arrivals and their execution time within their estimated WCET ranges. The empirical probability, i.e., relative frequency, of deadline misses is computed by the ratio of the number of simulation runs containing any deadline misses to the total number of simulation runs, i.e., 40000. The probability comparison depicted in Figure 5 shows that the selected probability of deadline misses by SAFE is larger than the empirical probability computed by simulation-based evaluations. SAFE infers a logistic regression model, providing a probabilistic interpretation, based on a labelled dataset evaluated by worst-case task arrivals. The inferred logistic regression model, therefore, likely fits the system executions when task arrivals are worst with respect to maximising the magnitude of deadline misses. However, the empirical probability estimated through simulations is based on system executions when tasks randomly arrive within their inter-arrival time ranges. Hence, a logistic regression model obtained by SAFE enables more conservative probabilistic interpretations of the estimated WCET ranges than simulation-based evaluations for the WCET ranges. This implies that actual deadline miss probabilities tend to be lower than SAFE probability estimates, which is in practice a desirable property.

The answer to **RQ1** *is that* SAFE significantly outperforms the baseline method with respect to minimising the number of deadline misses when using the estimated WCET ranges. Across our experiments, SAFE takes at most 27.1h (while the baseline takes 26.4h) to estimate the best-size WCET ranges. The execution time is acceptable as SAFE can be executed offline in practice.

RQ2. Figure 6 depicts distributions of precision (Figures 6a, 6c, and 6e) and recall (Figures 6b, 6d, and 6f) obtained from EXP2 with the ADCS, ICS, and UAV subjects. The boxplots in Figures 6a, 6c, and 6e (resp. Figures 6b, 6d, and 6f) show distributions (25%-50%-75% quantiles) of precision (resp. recall) values obtained from 50 executions of SAFE with either distance-based sampling (D)



(a) ADCS: Precision

(b) ADCS: Recall

(c) ICS: Precision

(d) ICS: Recall

(e) UAV: Precision

(f) UAV: Recall

Fig. 6. Distributions of precision and recall over 100 model refinements when SAFE employs either our distance-based sampling (D) or random sampling (R) for (a,b) ADCS, (c,d) ICS, and (e,f) UAV. The boxplots (25%-50%-75%) show precision (a,c,e) and recall (b,d,f) values obtained from 50 runs of SAFE with each sampling strategy. The lines represent average trends.

or simple random sampling (R). The solid lines represent the average trends of precision and recall value changes over 100 regression model refinements.

As shown in Figures 6a, 6c, and 6e, across over 100 model refinements, SAFE with D achieves higher precision values than those obtained by R for the three subjects. Also, Figures 6a, 6c, and 6e show that the variance of precision with D tends to be smaller than that of R. On average, D's

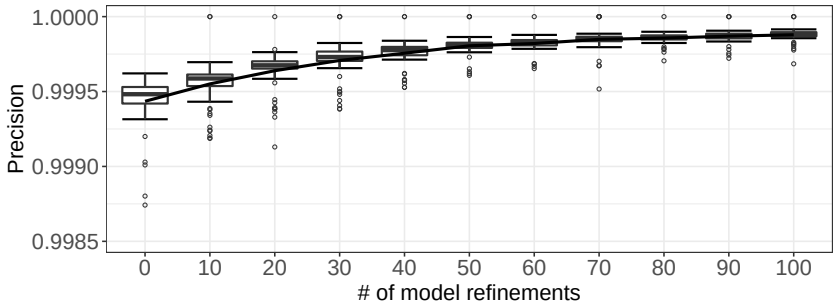


Fig. 7. Precision values computed from 10-fold cross validation at each model refinement for ADCS. The boxplots (25%-50%-75%) show precision values obtained from 50 runs of SAFE. The line represents an average trend.

precision converges toward 1 with model refinements; however, precision with R shows a markedly different trend that is not converging to 1, an important property in our context. Based on statistical comparisons, the difference in precision values between D and R becomes statistically significant after only 10, 4, and 11 model refinements, respectively, for ADCS, ICS, and UAV.

Regarding recall comparisons between D and R for ADCS, as shown in Figure 6b, D produces higher recall values over 100 model refinements than those of R. The difference in recall values between D and R becomes statistically significant after 36 model refinements. Regarding ICS (Figure 6d) and UAV (Figure 6f), their differences in recall values for D and R are not statistically significant even after 100 model refinements. This may be explained by their much smaller number of tasks compared with ADCS. Across our experiments, for 100 model refinements, SAFE took, at most, 10.86h and 10.54h with D and R, respectively.

The answer to **RQ2** *is that* SAFE with distance-based sampling significantly outperforms SAFE with random sampling in achieving higher precision. Only distance-based sampling can achieve a precision close to 1 within practical time, an important requirement in our context.

RQ3. Figure 7 shows precision values obtained from 10-fold cross-validation at each model refinement for the ADCS subject. Recall from Section 4.2 that SAFE stops model refinements once a precision value reaches a desired value. As shown in Figure 7, precision values tend to increase with additional WCET samples. Hence, practitioners are able to stop the model refinement procedure once precision reaches an acceptable level, e.g., >0.999. At 100 model refinement, SAFE reaches, on average, a precision of 0.99986. For EXP3, SAFE took, at most, 16.36h for phase 1 and 10.74h for phase 2.

As described in Section 4.2, SAFE reduces the dimensionality of the WCET space through a feature reduction technique based on random forest. The computed importance scores of each task's WCET in our dataset are as follows: 0.773 for T30, 0.093 for T33, 0.016 for T23, and $\leq$0.005 for the remaining 31 tasks. Based on a standard feature selection guideline [37], only the WCET values of two tasks, i.e., T30 and T33, are deemed to be important enough to retain as their score is higher than the average importance, i.e., 0.0385. Hence, SAFE computes safe WCET ranges of these two tasks in the next steps described in Algorithm 1.

Figure 8 shows the inferred safe border which identifies safe WCET ranges within which all 34 tasks are schedulable with an estimated deadline miss probability of 1.97%. Given the safe border, we found a best-size point which restricts the WCET ranges of T30 and T33 as follows: T30 [0.1ms,

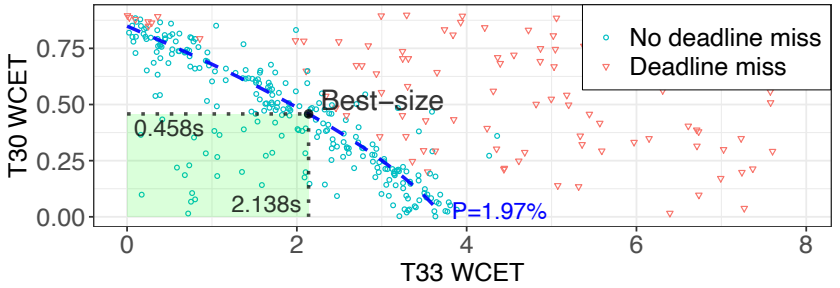


Fig. 8. An inferred safe border and best-size WCET regions for tasks T30 and T33. The safe border determines WCET ranges within which tasks are likely to be schedulable with a deadline miss probability of 1.97%.

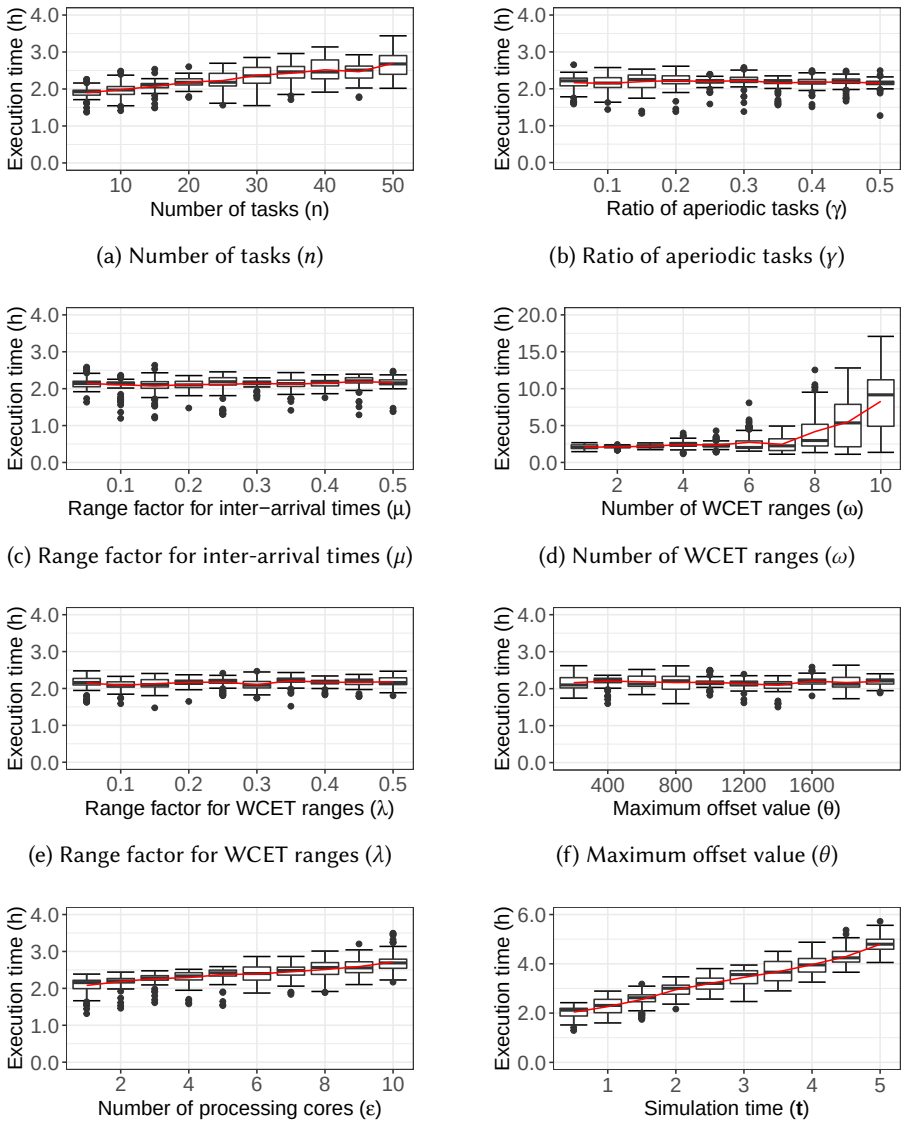
458.0ms] and T33 [0.1ms, 2138.1ms]. We note that the initial estimated WCET ranges of the two tasks are as follows: T30 [0.1ms, 900.0ms] and T33 [0.1ms, 20000.0ms]. SAFE therefore resulted in safe WCET ranges representing a significant decrease of 49.11% and 89.31% of initial maximum WCET estimates, respectively. This information is therefore highly important and can be used to guide design and development.

The answer to **RQ3** *is that* SAFE helps compute safe WCET ranges that have a much lower maximum than practitioners' initial WCET estimates. Our case study showed that SAFE determined safe maximum WCET values that were only 51% or less the original estimate. Further, these safe WCET ranges have a deadline miss probability of 1.97% based on the inferred logistic regression model. More restricted ranges can be selected to reduce this probability. SAFE took, on average, 25.14h to compute such safe WCET regions, which is acceptable for offline analysis in practice.

RQ4. Figure 9 shows the distributions (25%-50%-75%) of execution times obtained from 10 × 10 runs of SAFE, i.e., 10 runs for each of the 10 synthetic systems with the same experimental setting (see Section 5.4). The red solid lines in Figure 9 represent the mean value changes of the execution times of SAFE over varying values of the following control parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor for inter-arrival times μ, (d) number of WCET ranges ω, (e) range factor for WCET ranges λ, (f) maximum offset value θ, (g) number of processing cores ϵ, and (h) simulation time $\mathbf{t}$. Note that all the experiments in EXP4 took at most 16.7h, which is acceptable as SAFE is an offline analysis technique.

As shown in Figures 9a, 9g, and 9h, the execution time of SAFE is linear in the number of tasks n, the number of processing cores ϵ, and the simulation time $\mathbf{t}$. However, regarding the ratio of aperiodic tasks γ (Figure 9b), the range factor for inter-arrival times μ (Figure 9c), the range factor for WCET ranges λ (Figure 9e), the maximum offset value θ (Figure 9f), the results indicate that they have no correlations with the execution time of SAFE. Therefore, we expect SAFE to scale well as the number of tasks, the number of processing cores, and the simulation time increase.

Figure 9d shows that the execution time of SAFE is quadratically correlated with the number of WCET ranges ω in a system. Recall from Section 4 that the number of tasks characterising their WCETs as ranges (instead of point values) determine the size of a labelled dataset. Specifically, SAFE uses a second-order polynomial response surface model (RSM) to build a logistic regression model. RSM contains linear terms, quadratic terms, and 2-way interactions between linear terms (see Section 4.2). Hence, the number of coefficients in an RSM to be inferred by logistic regression is the sum of the numbers of constants, linear terms, quadratic terms, and 2-way interactions in an RSM, i.e., $1 + \omega + \omega + \binom{\omega}{2}$ (see the RSM equation presented in Section 4.2), which impacts



(a) Number of tasks (n)

(b) Ratio of aperiodic tasks (γ)

(c) Range factor for inter-arrival times (μ)

(d) Number of WCET ranges (ω)

(e) Range factor for WCET ranges (λ)

(f) Maximum offset value (θ)

(g) Number of processing cores (ϵ)

(h) Simulation time (**t**)

Fig. 9. Execution times of SAFE when varying the values of the following parameters: (a) number of tasks n, (b) ratio of aperiodic tasks γ, (c) range factor for inter-arrival times μ, (d) number of WCET ranges ω, (e) range factor for WCET ranges λ, (f) maximum offset value θ, (g) number of processing cores ϵ, and (h) simulation time **t**. The boxplots (20%-50%-75%) show the distributions of execution time values obtained from 100 runs of SAFE, i.e., 10 runs for each of the 10 synthetic systems with the same configuration. The red line in each figure connects the mean values of the execution times of SAFE over the parameter values.

the execution time of logistic regression dominating the execution time of SAFE. Note that the execution time of logistic regression is linear in the number of coefficients of a regression model

(e.g., RSM) [40]. Hence, the execution time of logistic regression in SAFE is quadratically correlated with the number of WCET ranges as explained above. In addition, the results presented in Figure 9d show that the magnitude of the execution time variation increases when the number of WCET ranges in a system increases. As written in Section 4.2, SAFE employs a feature reduction technique and a stepwise regression technique to efficiently infer logistic regression models (see lines 1-4 of Algorithm 1). The outputs of the two techniques depend on the number of tasks whose WCETs significantly impact deadline misses. Such outputs then impact the execution times of the following steps in Algorithm 1: imbalance handling, sampling, and regression. Note that the synthetic systems generated by setting $\omega = 10$ are more diverse in terms of WCET ranges when compared to the systems created by setting $\omega = 1$.

In EXP4, SAFE analyses all tasks in a system. However, recall from Section 4 that SAFE provides the capability of selecting target tasks as engineers often need to focus on the most critical ones. In such cases, the execution time of SAFE can significantly decrease.

The answer to **RQ4** *is that* the execution time of SAFE is linear in the number of tasks, the number of processing cores, and the simulation time. However, the execution time of SAFE is quadratically correlated with the number of tasks whose WCETs are given as ranges. Across our experiments, SAFE took at most 17.1h, which is acceptable for offline analysis in practice.

Benefits from a practitioner's perspective. Investigating practitioners' perceptions of the benefits of SAFE is necessary to adopt SAFE in practice. To do so, we draw on the qualitative reflections of three software engineers at LuxSpace, with whom we have been collaborating on this research. The reflections are based on the observations that the engineers made throughout their interactions with the researchers.

SAFE produces a set of worst-case sequences of task arrivals (see Section 4.1). Engineers deemed them to be useful for further examinations by experts. The current practice is to use an analytical schedulability test [44] which proves whether or not a set of tasks are schedulable. Such an analytical technique typically does not provide additional information regarding possible deadline misses. In contrast, worst-case task arrivals and safe WCET ranges produced by SAFE offer insights to engineers regarding deadline miss scenarios and the conditions under which they happen.

Engineers noted that some tasks' WCET are inherently uncertain and that such uncertainty is hard to estimate based on expertise. Hence, their initial WCET estimates were very rough and conservative. Further, estimating what WCET sub-ranges are safe is even more difficult. Since SAFE estimates safe WCET ranges systematically with a probabilistic guarantee, the engineers deem SAFE to improve over existing practice. Also, SAFE allows engineers to choose system-specific safe WCET ranges from the (infinite) WCET ranges modeled by the safe border, rather than simply selecting the best-size WCET range automatically suggested by SAFE (Figure 8). This flexibility allows engineers to perform domain specific trade-off analysis among possible WCET ranges and is useful in practice to support decision making with respect to their task design.

Given the fact that we have not yet undertaken rigorous user studies, the benefits highlighted above are only suggestive but not conclusive. We believe the positive feedback obtained from LuxSpace and our industrial case study shows that SAFE is promising and worthy of further empirical research with human subjects.

5.8 Threats to Validity

Internal validity. To ensure that our promising results cannot be attributed to the problem merely being simple, we compared SAFE with an alternative baseline using random search under identical parameter settings (see the RQ1 results in Section 5.7). Phase 1 of SAFE can indeed be replaced

with a random search, as we did, or even an exhaustive technique if the targeted system is small. However, there are no alternatives for Phase 2 – our main contribution – which infers safe WCET ranges and enables trade-off analysis. We present all the underlying parameters and provide our full evaluation package [43] to facilitate reproducibility. We mitigate potential biases and errors in our experiments by drawing on an industrial case study in collaboration with engineers at LuxSpace.

Recall from Section 5.7 that we compared probability values of deadline misses computed by SAFE and simulations. The results show that SAFE enables engineers to have more conservative probabilistic interpretations of the estimated WCET ranges than simulation-based evaluations for the WCET ranges. However, depending on the system's characteristics, e.g., hard real-time systems, engineers may need absolute guarantees for the WCET estimates. In such cases, once engineers find safe WCET ranges using SAFE, they can, in theory, obtain an absolute guarantee using exhaustive verification techniques, e.g., UPPAAL [53], on whether tasks always meet their deadlines for the given WCET ranges or not. Note that we performed an experiment using UPPAAL as it has often been used in the literature [54, 74, 76]. We applied UPPAAL to verify whether ADCS tasks are schedulable for the given WCET values. However, our experiment results showed that UPPAAL was not able to complete the analysis task, even after five days of execution. Hence, engineers should therefore consider such scalability issues when applying exhaustive analysis techniques to complement SAFE. Since this UPPAAL evaluation is not the main focus of this article, we point the reader to the UPPAAL specification of ADCS available online [43].

External validity. The main threat to external validity is that our results may not generalize to other contexts. We evaluated SAFE using early-stage WCET ranges estimated by practitioners at LuxSpace. However, SAFE can be applied at later development stages as well (1) to test the schedulability of the underlying set of tasks of a system and (2) to develop tasks under more precise constraints regarding safe WCETs. Future case studies covering the entire development process remain necessary for a more conclusive evaluation of SAFE. In addition, while motivated by ADCS (see Section 5.2) in the satellite domain, SAFE is designed to be generally applicable to other contexts. To evaluate the usefulness of SAFE in other contexts, in addition to our motivating case study system, we applied SAFE to two industrial systems from different domains, having very different system characteristics such as resource dependencies and multiple processing cores. As we described in Section 5.2, however, none of the public study subjects provide initial estimates of their task WCETs as ranges, which are required by SAFE as input. Hence, we had to modify the study subjects to include WCET ranges in their task descriptions but attempted to minimise any potential biases and errors in the experiments by converting a point WCET value to a WCET range in a systematic and straightforward way. In Section 5.2, we described the converting method in detail. Also, we made the modified task descriptions available online [43]. However, the general usefulness of SAFE needs to be further assessed in other contexts and domains.

6 RELATED WORK

This section discusses and compares SAFE with related work in the areas of schedulability analysis, as well as testing and verification of real-time systems.

Schedulability analysis has been widely studied for real-time systems [3, 4, 9, 12–15, 20, 33–35, 50, 51, 55, 57, 60, 64, 69, 72, 73]. Among them, the most related research strands study uncertain execution times [9, 14, 55, 72], probability of deadline misses [50, 51, 69], weakly hard deadlines [12, 60, 73], schedulability regions [4, 20, 64], and WCET estimation [3, 13, 15, 33–35] in the context of real-time task analysis.

Bini et al. [14] propose a theoretical sensitivity analysis method for real-time systems accounting for a set of periodic tasks and their uncertain execution times. Brüggen et al. [69] present an analytical method to analyse a deadline miss probability of real-time tasks using probability density

functions of approximated task execution times. In contrast to SAFE, most of these analytical approaches do not directly account for aperiodic tasks having variable arrival intervals; instead, they treat aperiodic tasks as periodic tasks using their minimum inter-arrival times as periods [24]. However, SAFE takes various task parameters, including irregular arrival times, into account without any unwarranted assumption. Also, our simulation-based approach enables engineers to explore different scheduling policies provided by real RTOS; however, these analytical methods are typically only valid for a specific conceptual scheduling policy model.

Bernat et al. [12] introduce the concept of weakly hard real-time systems that can tolerate occasional deadline misses. They precisely define weakly hard deadline constraints, specifying a maximum number of deadlines that can be missed during a time window, and provide the theoretical analysis of the properties and relationships between tasks with the temporal constraints. Xu et al. [73] develop an algorithm that accounts for sporadic task overload when analysing the number of deadlines a task can miss in a given sequence of consecutive task arrivals. Pazzaglia et al. [60] present an extended weakly hard analysis method by accounting for additional uncertainties such as task offsets and release jitters. SAFE complements the above research strands on weakly hard real-time systems since, instead of analysing the number of deadlines a task can afford to miss over a time window, SAFE provides probabilistic guarantees for deadline misses based on logistic regression models inferred from search and simulation outputs.

Cimatti et al. [20] develop an approach that computes the regions of the task parameter values guaranteeing tasks are schedulable, i.e., schedulability regions. Their approach uses parametric timed automata and an SMT solver, i.e., NuSMT [19], and is applied to a system that contains two periodic tasks. Sun et al. [64] propose a method, named IMITATOR, that aims at computing schedulability regions. IMITATOR is based on model checking of parametric timed automata with stopwatches. They evaluate the method by applying it to two test-case systems that contain at most two free parameters, i.e., task execution times, that are defined as variables. Note that the other parameters, e.g., task periods and deadlines, are defined as fixed values. The results show that IMITATOR covers the entire parameter space but does not scale well with the size of the problem. André et al. [4] developed a tool that translates a graphical specification of a real-time system to the input of IMITATOR such that it allows computation of some schedulability regions using IMITATOR. However, these methods that exhaustively search the problem space are often not amenable to analyse industrial systems in a scalable manner as they typically contain many tasks, different task types, complex task relationships, and multiple processing cores.

Hansen et al. [34] present a measurement-based approach to estimate WCET and a probability of estimation failure. The measurement-based WCET estimation technique collects actual execution time samples and estimates WCETs using linear regression and a proposed analytical model. To our knowledge, most of the research strands regarding WCET estimation are developed for later development stages at which task implementations are available. Note that relatively few prior works aim at estimating WCET at an early design stage; however, these work strands still require access to source code, hardware, compilers, and program behaviour specifications [3, 15, 33]. In contrast, SAFE uses as input estimated WCET ranges and then precisely restricts the WCET ranges within which tasks are schedulable with a selected deadline miss probability, by relying on a tailored genetic algorithm, simulation, feature reduction, a dedicated sampling strategy, and logistic regression.

Testing and verification are important to successfully develop safety-critical real-time systems [1, 17, 27, 42, 53, 77]. Some prior studies employ model-based testing to generate and execute tests for real-time systems [27, 53, 77]. SAFE complements these prior studies by providing safe WCETs as objectives to engineers implementing and testing real-time tasks. Constraint programming and model checking have been applied to ensure that a system satisfies its time constraints [1, 42]. These

techniques may be useful to conclusively verify whether or not a WCET value is safe. However, such exhaustive techniques are not amenable to address the analysis problem addressed in this article, which requires the inference of safe WCET ranges. To our knowledge, SAFE is the first attempt to accurately estimate safe WCET ranges to prevent deadline misses with a given level of confidence and offer ways to achieve different trade-offs among tasks' WCET values.

7 CONCLUSION

We developed SAFE, a two-phase approach applicable in early design stages, to precisely estimate safe WCET ranges within which real-time tasks are likely meet their deadlines with a high-level of confidence. SAFE uses a meta-heuristic search algorithm to generate worst-case sequences of task arrivals that maximise the magnitude of deadline misses, when they are possible. Based on the search results, SAFE uses a logistic regression model to infer safe WCET ranges within which tasks are highly likely to meet their deadlines, given a selected probability. SAFE is developed to be scalable by using a combination of techniques such as a genetic algorithm and simulation for the SAFE search (phase 1) and feature reduction, an effective sampling strategy, and polynomial logistic regression for the SAFE model refinement (phase 2). We evaluated SAFE on a mission-critical, real-time satellite system in collaboration with a satellite company as well as two industrial systems from different domains whose description was retrieved from the literature. The results indicate that SAFE is able to precisely compute safe WCET ranges for which deadline misses are highly unlikely, these ranges being much smaller than the WCET ranges initially estimated by engineers. Further, we evaluated the scalability of SAFE using a number of synthetic systems. The results indicate that SAFE scales to complex systems. Across the experiments on industrial and synthetic systems, SAFE took at most 27h, which is acceptable in practice as an offline analysis method.

For future work, we plan to extend SAFE in the following directions: (1) developing a real-time task modelling language to describe dependencies, constraints, behaviours of real-time tasks and to facilitate schedulability analysis and (2) building a decision support system to recommend a schedulable solution if a set of tasks are not schedulable, e.g., priority re-assignments. In the long term, we would like to more conclusively validate the usefulness of SAFE by applying it to other case studies in different domains.

ACKNOWLEDGMENT

This project has received funding from 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.

REFERENCES

[1] Stefano Di Alesio, Arnaud Gotlieb, Shiva Nejati, and Lionel C. Briand. 2012. Testing Deadline Misses for Real-Time Systems Using Constraint Optimization Techniques. In *Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST'12)*. 764–769.

[2] Stefano Di Alesio, Shiva Nejati, Lionel C. Briand, and Arnaud Gotlieb. 2013. Stress Testing of Task Deadlines: A Constraint Programming Approach. In *Proceedings of the 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE'13)*. 158–167.

[3] 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.

[4] Étienne André, Jawher Jerray, and Sahar Mhiri. 2019. Time4sys2imi: A Tool to Formalize Real-Time System Models Under Uncertainty. In *Proceedings of the 16th International Colloquium on Theoretical Aspects of Computing (ICTAC'19)*, Vol. 11884. 113–123.

[5] Saoussen Anssi, Sara Tucci Piergiovanni, Stefan Kuntz, Sébastien Gérard, and François Terrier. 2011. Enabling Scheduling Analysis for AUTOSAR Systems. In *Proceedings of the 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC'11)*. 152–159.

[6] 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.

[7] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. 2018. *Operating Systems: Three Easy Pieces* (1.00 ed.). Arpaci-Dusseau Books.

[8] Neil C. Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39–44.

[9] Jakob Axelsson. 2005. A Method for Evaluating Uncertainties in the Early Development Phases of Embedded Real-Time Systems. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*. 72–75.

[10] S.K. Baruah, A. Burns, and R.I. Davis. 2011. Response-Time Analysis for Mixed Criticality Systems. In *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, 34–43.

[11] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. *SIGKDD Explorations* 6, 1 (2004), 20–29.

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

[13] 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)*. 279–288.

[14] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. 2008. Sensitivity Analysis for Fixed-Priority Real-Time Systems. *Real-Time Systems* 39, 1 (2008), 5–30.

[15] 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)*. 5:1–5:9.

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

[17] Lionel C. Briand, Yvan Labiche, and Marwa Shousha. 2005. Stress Testing Real-time Systems with Genetic Algorithms. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO'05)*. 1021–1028.

[18] Alan Burns and Andrew J. Wellings. 2009. *Real-Time Systems and Programming Languages - Ada, Real-Time Java and C / Real-Time POSIX* (4th ed.). Addison-Wesley.

[19] Roberto Cavada, Alessandro Cimatti, Anders Franzén, Krishnamani Kalyanasundaram, Marco Roveri, and R. K. Shyamasundar. 2007. Computing Predicate Abstractions by Integrating BDDs and SMT Solvers. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design (FMCAD'07)*. 69–76.

[20] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. 2008. Symbolic Computation of Schedulability Regions Using Parametric Timed Automata. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS'08)*. IEEE Computer Society, 80–89.

[21] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. 2012. *Model Checking and the State Explosion Problem.* Springer. 1–30 pages.

[22] Francis Cottet, Joëlle Delacroix, Claude Kaiser, and Zoubir Mammeri. 2002. *Scheduling in Real-Time Systems.*

[23] 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.

[24] Robert I. Davis and Alan Burns. 2011. A Survey of Hard Real-Time Scheduling for Multiprocessor Systems. *ACM Computing Surveys (CSUR)* 43, 4 (2011), 35:1–35:44.

[25] 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.

[26] Stefano Di Alesio, Lionel C. Briand, Shiva Nejati, and Arnaud Gotlieb. 2015. Combining Genetic Algorithms and Constraint Programming to Support Stress Testing of Task Deadlines. *ACM Transactions on Software Engineering and Methodology* 25, 1 (2015), 4:1–4:37.

[27] Stefano Di Alesio and Sagar Sen. 2018. Using UML/MARTE to Support Performance Tuning and Stress Testing in Real-Time Systems. *Software and Systems Modeling* 17, 2 (2018), 479–508.

[28] 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.

[29] Jens Eickhoff. 2011. *Onboard Computers, Onboard Software and Satellite Operations: An Introduction.* Springer.

[30] Filomena Ferrucci, Mark Harman, Jian Ren, and Federica Sarro. 2013. Not Going to Take This Anymore: Multi-objective Overtime Planning for Software Engineering Projects. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 462–471.

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

[32] 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.
[33] 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)*. 308–319.
[34] 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)*. 1–11.
[35] Damien Hardy, Isabelle Puaut, and Yiannakis Sazeides. 2016. Probabilistic WCET Estimation in Presence of Hardware for Mitigating The Impact of Permanent Faults. In *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE'16)*. 91–96.
[36] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Computing Survey* 45, 1 (2012), 11:1–11:61.
[37] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2nd ed.). Springer.
[38] Randy L. Haupt and Sue Ellen Haupt. 1998. *Practical Genetic Algorithms*. John Wiley & Sons, Inc.
[39] Kawakubo Hideko and Yoshida Hiroaki. 2012. Rapid Feature Selection Based on Random Forests for High-Dimensional Data. In *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12)*. 704–710.
[40] David W. Hosmer and Stanley Lemeshow. 1986. *Applied Logistic Regression*. John Wiley & Sons, Inc.
[41] David W. Hosmer Jr., Stanley Lemeshow, and Rodney X. Sturdivant. 2013. *Applied Logistic Regression* (3rd ed.). Wiley.
[42] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV'11)*. 585–591.
[43] Jaekwon Lee, Seung Yeob Shin, Shiva Nejati, Lionel C. Briand, and Yago Isasi Parache. 2022. [Case study data] Estimating Probabilistic Safe WCET Ranges of Real-Time Systems at Design Stages. https://figshare.com/s/d63d32c8ee726912e3f0.
[44] 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.
[45] Jane W. S. Liu. 2000. *Real-Time Systems* (1st ed.). Prentice Hall.
[46] Douglas Locke, Lee Lucas, and John Goodenough. 1990. *Generic Avionics Software Specification*. Technical Report CMU/SEI-90-TR-008. Software Engineering Institute, Carnegie Mellon University.
[47] Sean Luke. 2013. *Essentials of Metaheuristics* (2nd ed.). Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.
[48] Ahmed Maged, Salah Haridy, Mohammad Shamsuzzaman, Imad Alsyouf, and Roubi Zaied. 2018. Statistical Monitoring and Optimization of Electrochemical Machining using Shewhart Charts and Response Surface Methodology. *International Journal of Engineering Materials and Manufacture* 3 (2018), 68–77.
[49] 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.
[50] Sorin Manolache, Petru Eles, and Zebo Peng. 2004. Schedulability Analysis of Applications with Stochastic Task Execution Times. *ACM Transactions on Embedded Computer System (TECS)* 3, 4 (2004), 706–735.
[51] Dorin Maxim and Liliana Cucu-Grosjean. 2013. Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters. In *Proceedings of the 2013 IEEE 34th Real-Time Systems Symposium (RTSS'13)*. 224–235.
[52] Michael R. May and Brian R. Moore. 2016. How Well Can We Detect Lineage-Specific Diversification-Rate Shifts? A Simulation Study of Sequential AIC Methods. *Systematic Biology* 65, 6 (2016), 1076–1084.
[53] Marius Mikucionis, Kim Guldstrand Larsen, and Brian Nielsen. 2004. T-UPPAAL: Online Model-based Testing of Real-Time Systems. In *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE'04)*. 396–397.
[54] Marius Mikučionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. 2010. Schedulability analysis using UPPAAL: Herschel-Planck case study. In *Proceedings of the International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'10)*. 175––190.
[55] Pranab K. Muhuri and Kaushal Kumar Shukla. 2009. Real-Time Scheduling of Periodic Tasks with Processing Times and Deadlines as Parametric Fuzzy Numbers. *Applied Soft Computing* 9, 3 (2009), 936–946.
[56] Jagannath Munda and Bijoy Bhattacharyya. 2008. Investigation into Electrochemical Micromachining (EMM) Through Response Surface Methodology Based Approach. *The International Journal of Advanced Manufacturing Technology* 35 (2008), 821–832. Issue 7.
[57] Shiva Nejati, Stefano Di Alesio, Mehrdad Sabetzadeh, and Lionel Briand. 2012. Modeling and Analysis of CPU Usage in Safety-Critical Embedded Systems to Support Stress Testing. In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MODELS'12)*. 759–775.

[58] John A. Nelder and Roger Mead. 1965. A Simplex Method for Function Minimization. *Comput. J.* 7, 4 (1965), 308–313.
[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 Trans. Embed. Comput. Syst.* 20, 1 (2021), 3:1–3:26.
[61] Marie-Agnès Peraldi-Frati and Yves Sorel. 2008. From high-level modelling of time in MARTE to real-time scheduling analysis. In *Proceedings of the MODELS'08 Workshop on Model Based Architecting and Construction of Embedded Systems (ACESMB)*, Vol. 503. 129–144.
[62] Stuart J. Russell and Peter Norvig. 2010. *Artificial Intelligence - A Modern Approach* (3rd ed.). Pearson Education.
[63] John A. Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C. Buttazzo. 1998. *Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms.* Springer, Boston, MA.
[64] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. 2013. Parametric Schedulability Analysis of Fixed Priority Real-Time Distributed Systems. In *Proceedings of the 2nd International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS'13)*, Vol. 419. 212–228.
[65] Karim Traore, Emmanuel Grolleau, and Francis Cottet. 2006. Simpler Analysis of Serial Transactions Using Reverse Transactions. In *Proceedings of the 2006 International Conference on Autonomic and Autonomous Systems (ICAS'06)*. 11.
[66] 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.
[67] 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)*. 959–967.
[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)*. 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. 6:1–6:22.
[70] 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 *30th Euromicro Conference on Real-Time Systems (ECRTS 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 106)*, Sebastian Altmeyer (Ed.). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 6:1–6:22.
[71] Ian H. Witten, Eibe Frank, and Mark A. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc.
[72] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. 2007. Energy-Aware Scheduling for Real-Time Multiprocessor Systems with Uncertain Task Execution Time. In *Proceedings of the 2007 44th ACM/IEEE Design Automation Conference (DAC'07)*. 664–669.
[73] 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)*. 247–256.
[74] Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg. 2019. An exact schedulability test for non-preemptive self-suspending real-time tasks. In *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. 1228–1233.
[75] 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.
[76] Fei Yu, Guoqiang Li, and Naixue Xiong. 2010. Schedulability analysis of multi-processor real-time systems using UPPAAL. In *Proceedings of the 2nd International Conference on Information Science and Engineering (ICISE'10)*. 1–6.
[77] Justyna Zander. 2008. *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain.* Ph.D. Dissertation. Fraunhofer FOKUS.
[78] Fengxiang Zhang and Alan Burns. 2009. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Trans. Comput.* 58, 9 (2009), 1250–1258.
[79] Zhongheng Zhang. 2016. Variable Selection with Stepwise and Best Subset Approaches. *Annals of Translational Medicine* 4, 7 (2016), 1–6.