Probabilistic Analysis of Low-Criticality Execution

Martin Küttler, Michael Roitzsch, Claude-Joachim Hamann
Operating Systems Group
Department of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {mkuettle,mroi,hamann}@os.inf.tu-dresden.de

Marcus Völp
CritiX: Critical and Extreme Security and Dependability
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg
L-2721 Luxembourg
Email: marcus.voelp@uni.lu

***Abstract*—The mixed-criticality toolbox promises system architects a powerful framework for consolidating real-time tasks with different safety properties on a single computing platform. Thanks to the research efforts in the mixed-criticality field, guarantees provided to the highest criticality level are well understood. However, lower-criticality job execution depends on the condition that all high-criticality jobs complete within their more optimistic low-criticality execution time bounds. Otherwise, no guarantees are made. In this paper, we add to the mixed-criticality toolbox by providing a probabilistic analysis method for low-criticality tasks. While deterministic models reduce task behavior to constant numbers, probabilistic analysis captures varying runtime behavior. We introduce a novel algorithmic approach for probabilistic timing analysis, which we call *symbolic scheduling*. For restricted task sets, we also present an analytical solution. We use this method to calculate per-job success probabilities for low-criticality tasks, in order to quantify, how low-criticality tasks behave in case of high-criticality jobs overrunning their optimistic low-criticality reservation.**

I. Introduction

Mixed-criticality systems [1] promise size, weight and power savings by consolidating safety-critical tasks with different certification requirements on a single computing platform. Examples can be found in many traditional and emerging application scenarios. To formalize such systems and to reason about their behavioral properties, mixed-criticality was invented. It allows designers to rank tasks by criticality levels, which expresses the confidence in task parameters such as worst-case execution times and also indicates, which tasks to drop in case a subset of these task parameters are violated at runtime. Highly critical tasks with high parameter confidence can then be isolated from lower-criticality tasks with relaxed parameter confidence. System designers can use this formalism to meet two otherwise competing goals: criticality levels provide the *separation* needed for safe operation, while the admission of low-criticality tasks according to more optimistic task parameters provides the *resource sharing* needed for efficient operation.

A large body of research has explored many aspects of the mixed-criticality toolbox [2]. In this paper, we contribute new analysis results for the following mixed-criticality scheduling discipline: When a high-criticality job overruns its more optimistic low-criticality execution time bound, all low-criticality tasks drop in priority, so any task with higher criticality takes precedence. Demoting the priority of low-criticality tasks constitutes a straightforward extension of the classical Adaptive Mixed-Criticality (AMC) [3] scheduling discipline, which drops low-criticality jobs altogether after an execution time overrun of a high-criticality job.

Classical mixed-criticality scheduling is based on deterministic worst-case assumptions, which are pessimistic, because at runtime job parameters are rarely constant, but follow a distribution. We present a probabilistic analysis method to capture such varying job behavior. Our task model expresses job execution using a pWCET distribution. We describe the task model in Section III.

In this paper, we make the following contributions:

1) We present an algorithmic approach for probabilistic timing analysis of per-job behavior in mixed-criticality systems, which we call *symbolic scheduling* (Section IV). For restricted task sets with criticality-monotonic priority assignment and harmonic periods, we also present an analytical solution (Section V).
2) We use this analysis to quantify how low-criticality tasks behave in case of high-criticality jobs overrunning their optimistic low-criticality reservation. We calculate probabilities p for jobs of low-criticality tasks meeting their deadline when the system operates in high criticality mode.

In Section VI, we evaluate our analysis using randomly generated task sets. We show that our probabilistic analysis can quantify the low-criticality task execution. These success probabilities can, for example, be used for a more permissive admission test that requires only a given percentage q of jobs to succeed in low criticality mode.

II. Related Work

Lehoczky [4] was first to characterize execution times as random variables and to describe them through probability density functions. However, as realized by Griffin and Burns [5], modern processor architectures often violate the independence assumptions required for scheduling based on probabilistic execution times to remain mathematically tractable. Recent works on probabilistic worst-case execution times (pWCET) [6], [7], [8], [9], [10], [11], [12] thus describe the confidence in WCET estimates of a task as the exceedance probability derived from the generalized extreme-value distribution of observed maxima.

Different methods to derive probabilistic representations of execution time have been explored. For example, Yue et al. [13] present a technique based on random sampling for determining pWCETs. Iverson et al. [14] suggest a purely statistical analysis, whereas David and Puaut [15] propose a combined static and measurement-based analysis.

Statistical and probabilistic techniques have been used for real-time analysis in the past. Atlas and Bestavros [16] calculate task success rates when exact execution times are known at the release instant. Guo et al. [17] perform admission tests under given failure probabilities. Hamann et al. [18], [19] extend imprecise computations [20] to derive budgets for optional parts that guarantee a certain completion probability.

Recent work on probabilistic mixed-criticality analysis by Maxim et al. [21] calculates worst-case response times (WCRT) for static and adaptive mixed-criticality scheduling based on critical instant analysis. However, WCRT alone is an inadequate quality metric for a mixed-criticality schedule. After a criticality switch to high-criticality mode, the low-criticality job following the critical instant may never execute, whereas all following jobs of the same task could always be successful. Therefore, a success probability based on WCRT is arbitrarily pessimistic.

Our work extends this analysis by calculating success probabilities for every job individually and aggregating them to a per-task value that is less pessimistic. We propose *symbolic scheduling,* which can be viewed as a specialized form of probabilistic model checking.

III. Task Model

In his seminal work, Vestal proposed to describe tasks with different certification requirements through vectors of increasingly pessimistic scheduling parameters [1]. Admission and scheduling must ensure that a higher-criticality task failing to meet the more optimistic requirements from a lower certification level can still meet its deadline when it adheres to the more pessimistic parameters at its high certification level. Baruah et al. coined the term certification-cognizant scheduling [22] for this interpretation of the mixed-criticality framework. Our paper follows this interpretation.

In the standard deterministic model, a task $\tau = (T, D, L, C(\ell))$ is assumed strictly periodic with period T, a relative deadline D, a criticality level L and a worst-case execution time (WCET) $C(\ell)$ for each criticality level $\ell \leqslant L$. These WCETs must satisfy $C(\ell_1) \leqslant C(\ell_2)$ for $\ell_1 \leqslant \ell_2$. Our analysis does not restrict the number of criticality levels or the relation of T and D, so $D \geqslant T$ is possible. To simplify the presentation, we only discuss the dual criticality case, with the two criticality levels named LO and HI. Other research also considers criticality-dependent interrelease times [23] or deadlines [24]. We limit ourselves to criticality-dependent execution times.

We extend the standard task model to a probabilistic one similar to Maxim et al. [21]. In addition to the above parameters, each task is described by a probabilistic worst case execution time (pWCET) X. The CDF of this random variable describes the probability for a job to not exceed a given execution time bound, which allows us to compute probabilistic guarantees for low-criticality jobs. We write $X = [2 : 0.8, 5 : 0.2]$, e.g., meaning that with a probability of 0.8 the job will have finished executing until 2 time units, and with a probability of 0.2 it may exceed 2 time units and take up to 5 time units.

We assume an active enforcement of execution times by the runtime system. Whenever an execution time bound or deadline is reached, the system aborts jobs or drops them in priority. Whenever a LO-criticality job executes beyond $C(LO)$, it is aborted. $C(HI)$ is therefore not meaningful for LO-jobs. Whenever a HI-criticality job exceeds its $C(LO)$ execution time, the system switches to HI-criticality mode. We call this situation *criticality miss.* As part of this mode switch, the priorities of all current and future LO-jobs are changed such that all HI jobs dominate all LO-jobs. We allow switching back from HI mode to LO mode only at a simultaneous release instant at the beginning of a hyperperiod. Protocols allowing earlier recovery have been presented [25], but are not considered here.

IV. Symbolic Scheduling

We propose the concept of *symbolic scheduling,* which we present here and which we have implemented[1]. Symbolic scheduling draws on ideas from actual runtime scheduling, in that it tracks runs of jobs along a time axis to figure out which meet their deadline. Unlike a scheduler, though, it does not observe concrete execution times, but it keeps track of possible executions and their probabilities.

Given a set of tasks as described in Section III, there is a conceptually simple but computationally expensive way to analyze the behavior of each job: Try every possible combination of execution times, and keep track of the respective probabilities. This leads to a tree, where each path from the root to a leaf is a possible execution trace of the system, and each node branches into as many subtrees as the respective job has possible values for its execution. The obvious disadvantage is that this tree will grow huge, and that many paths through it will be equivalent for practical purposes. Such equivalent paths may differ in execution times, but agree in the succession of jobs and them finishing before their deadline. Symbolic scheduling takes advantage of these equivalences by trying to merge branches that only differ in timings, but not in the jobs' order and success. More precisely, it does not branch unless there is an immediate and important difference. However, depending on taskset parameters our current implementation can take multiple hours to produce a result.

A. Example

Consider the following example with two tasks:

$$\tau_1 : T = D = 8, X = [2 : 0.8, 5 : 0.2]$$
$$\tau_2 : T = D = 16, X = [1 : 0.6, 11 : 0.4]$$

We ignore criticality for now, so we can ignore L and C. The two Jobs of task τ_1 that run in the first hyperperiod are called J_{11} and J_{12}, the job of τ_2 is called J_{21}.

Assuming EDF scheduling, the symbolic scheduler starts at time $t = 0$ and first select job J_{11}. Since it is a job of the task of highest priority, it will run until completion at either time

[1]https://github.com/mkuettler/symbolic-scheduler

$$\begin{tikzcd}
 & & J_{12} \arrow[r, "0.6"] & \varnothing & \\
J_{11} \arrow[r, "1"] & J_{21} \arrow[ur, "0.6"] \arrow[dr, "0.4"'] & & & \varnothing \\
 & & J_{12} \arrow[r, "0.4"'] & J_{21} \arrow[ur, "0.256"] \arrow[dr, "0.144"'] & \\
 & & & & \varnothing
\end{tikzcd}$$

Figure 1. Event tree for the example in IV-A

2 or 5. Note that these two cases only differ in their time of occurrence and probability — the ready jobs and completed jobs are exactly the same. Thus we do not want to distinguish these cases as different timelines, because the full combinatorial tree that would ensue is prohibitively huge. Instead, the symbolic scheduler represents the current time using (partial) distributions: after scheduling J_{11} we are at time $t = [2 : 0.8, 5 : 0.2]$. These distributions describe the probability of the currently investigated situation. Note that the accumulated probability of such a *partial distribution* can be less than 1.

Next, job J_{21} runs. The total runtime of these first two jobs can be 3, 6, 13, or 16. Only the first two results are possible times for J_{21} to complete, because J_{12} becomes ready at time 8 and will interrupt J_{21} if it is still running. Thus, the symbolic scheduler needs to branch into two different possible timelines.

Like before, we do not want to branch needlessly. We only need to distinguish whether J_{21} finishes before J_{12} arrives. If it does — i.e., when J_{21} executes only for 1 time unit — we are at time $t = [3 : 0.48, 6 : 0.12]$, with J_{12} as the only job left. The scheduler will wait until the arrival of J_{12} at 8, which leaves us at $t = [8 : 0.6]$, since the total probability is $0.6 = 0.48 + 0.12$. Then J_{12} can run, and finishes at $t = [10 : 0.48, 13 : 0.12]$. This trace corresponds to the topmost branches in Figure 1.

If J_{21} does not finish before J_{12} arrives, it is interrupted at $t = [8 : 0.4]$, because J_{12} has higher priority. But J_{21} is not done yet and still remains in the list of ready jobs. It ran for $[6 : 0.8, 3 : 0.2]$ time units already, so the remaining time is $[5 : 0.8, 8 : 0.2]$. Now J_{12} runs to completion at $t = [10 : 0.32, 13 : 0.08]$. After that the remaining part of J_{21} is scheduled, and runs until $[15 : 0.256, 18 : 0.128, 21 : 0.016]$. But since the deadline of J_{21} is at 16, the job will finish successfully with a probability of 0.256, and miss its deadline with probability $0.128 + 0.016 = 0.144$. This simplified example illustrates the main concept of symbolic scheduling. To give a formal description we need to introduce some notation.

B. Notation

Let d, d_1, d_2 be potentially partial distributions, and x be a number.

- $d_1 + d_2$ denotes the convolution of d_1 and d_2.
- $d \Lsh x$ is the part of d that lies to the left of x, including x. Conversely, $d \Rsh x$ is the part of d that lies to the right of x, excluding x.
- $\operatorname{sum}(d)$ denotes the total probability of all values of d. Thus $\operatorname{sum}(d) = \operatorname{sum}(d \Lsh x) + \operatorname{sum}(d \Rsh x)$ for all d and x.
- $d_1 \cup d_2$ is defined to be the distribution that contains all the points in d_1 and d_2, with their respective probabilities. Hence $\operatorname{sum}(d_1 \cup d_2) = \operatorname{sum}(d_1) + \operatorname{sum}(d_2)$, which must be $\leqslant 1$ for this operation to make sense.
- $d \rightarrowtail x := (d \Rsh x) \cup [x : \operatorname{sum}(d \Lsh x)]$

Algorithm 1 Symbolic Scheduling without criticality misses

```
function sym_sched(t, jobs) {
    J = next_job(jobs)
    if J is None: return
    t = t ↣ J.release
    s = next_sched_event(jobs)
    t1 = (t + J.X) ↰ s
    t2 = (t + J.X) ↱ s
    if not empty(t1) {
        J.success += sum(t1)
        new_jobs = jobs.remove(J)
        sym_sched(t1, new_jobs)
    }
    if not empty(t2) {
        diff = sum(t2)-sum(t ↱ s)
        t_next = (t ↱ s) ∪ [s: diff]
        new_jobs = jobs.remove(J)
        if s ≠ J.deadline {
            elapsed = (s-time) ↣ 0
            J.X = (J.X - elapsed) ↱ 0
            J.X = normalize(J.X)
            new_jobs.insert(J)
        }
        sym_sched(t_next, new_jobs)
    }
}
```

With this notation, we can formally describe symbolic scheduling for the simplified scenario where criticality misses do not change job priorities, i.e., priorities are criticality monotonic. The general formulation is more complex and due to spatial constraints, we refer the reader to our technical report [26].

C. Simplified Formal Description

Let t be the current time distribution, and J the next job, i.e., the ready job with the highest priority. Let s be the time of the next scheduling event, i.e., either the deadline of J or the release of a job with higher priority. Note that s is not a distribution, but a scalar value. To calculate the next time point t_{next}, there are two cases to consider:

- J finishes before s: $t_{\text{next}} = (t + X(J)) \Lsh s$.
- J does not finish before s. Intuitively, t_{next} should be $[s : \operatorname{sum}((t + X(J)) \Rsh s)]$, like in the example above. But that only works if $t \leqslant s$ (i.e. $t \Rsh s$ is empty), otherwise the next time point would lie in the past. Branching into two different timelines would be an option, but for performance reasons we want to reduce branches. Instead, we can handle this case as follows:

$$t_{\text{next}} = t \Rsh s \cup \left[s : \operatorname{sum}\big((t + X(J)) \Rsh s\big) - \operatorname{sum}(t \Rsh s)\right].$$

In this case J is not done, so unless s is its deadline it must be kept in the list of ready jobs. But to account for the time it did run, its remaining execution time must be set to $(X(J) - ((s - t) \rightarrowtail 0)) \Rsh 0$, normalized to total probability of 1.

The algorithm is shown in Algorithm 1. Starting at $t = 0$, it selects the ready job with the highest priority, and, for each of

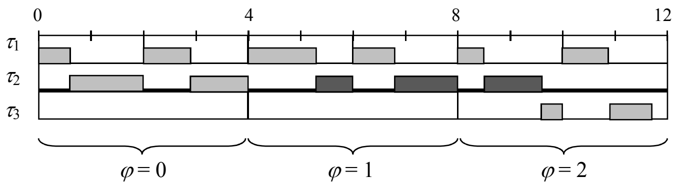

Figure 2. Job Phases and Remaining Execution Time

the two cases, updates the time and ready list, and repeats. In a general mixed-critically system there is one additional case to consider: A job may trigger a critically miss, i.e., it may overrun its $C(LO)$, thus causing the system to switch to HI-mode. This situation can be covered by a third branch to be followed and analyzed. We present more details in the technical report [26].

V. Analytical Solution

For determining per-job success probabilities analytically, we first restrict the task model to allow reasonable calculation effort and formula complexity. We consider systems with implicit deadlines and harmonic periods, consisting of m HI-tasks τ_i and n LO-tasks τ_{m+j}. The scheduling algorithm uses criticality-monotonic priority assignment: Tasks are assigned a static priority, such that all HI-tasks dominate all LO-tasks, thus forming two priority bands. Within each band, rate monotonic priority assignment is used. A criticality miss is thus inconsequential. The typical approach to switch back from HI to LO-mode at processor idle time results in LO-jobs always being executed as soon as the processor is done executing HI jobs.

Due to limited space, we again only sketch the central case, which would entail further special cases described in the technical report [26]. The central case is characterized by all LO-tasks having period lengths at most the length of the hyperperiod of the HI-tasks, hence $T_{m+n} \leqslant T_m$. Clearly, the success probability p_j of a job of task τ_{m+j} depends on its position within this hyperperiod. We call this position the job's phase φ, counting from 0 as shown in Figure 2 for two HI-tasks τ_1, τ_2 and one LO-task τ_3, as well as $T_1 = 2$, $T_2 = 12$, $T_3 = 4$.

Crucial for determining p_j is the calculation of the time demanded by all HI-tasks within one phase as well as the remaining execution times of those HI jobs, whose period exceeds that of the currently considered LO job. According to Figure 2, those HI jobs start earlier, but extend into the phase of the considered LO job. Any such remaining execution time is also a random variable, so a concrete realization may already finish in phase $\varphi = 1$.

Analyzing the highest priority LO-task τ_{m+1} is comparably simple: The calculation is performed in *schedule slices* of length T_{m+1}. Unfortunately, applying the same idea to all lower priority LO-tasks is dangerously misleading, because remaining execution times or remaining processor capacity across multiple phases are not stochastically independent. A formal description must be based on conditional probabilities, which can be calculated by distinction of multiple cases. Similar to the symbolic scheduler, we employ event trees to solve this problem. We continue in two steps.

In step one, we construct the event tree $\tilde{A}_0$ describing all HI-tasks in schedule slices of length T_{m+1}. Nodes of this tree are random variables A with the intuitive meaning 'total execution time of all HI-tasks minus period length'. Formally, the semantics is as follows: For $A = a$ with $a \geqslant 0$, the highest priority LO-task receives a processor capacity of a within the respective phase and no remaining execution time of HI-jobs occurs. In case $a < 0$, the processor is completely occupied by the HI-tasks, which further contribute a remaining execution time of $-a$. Edges within the tree are annotated with the probabilities of the respective case.

We denote the resulting random variables $A_{0\varphi}^{r_1 \ldots r_\varphi}$, with φ being the phase of τ_{m+1} and $r_1, \ldots, r_\varphi$ the remaining execution times occurring up until this phase. We call this sequence the history H of $A_{0\varphi}$. For the HI-tasks within phase $\varphi = 0$ of τ_{m+1} we have $A_{00} = T_{m+1} - \sum_{i=1}^{n} X_i$, which determines the root node of the event tree. For the general case we observe that a remaining execution time $r < T_{m+1}$ leaves us with a remaining processor capacity of $T_{m+1} - r$. This amount is reduced by the time demand of all HI-jobs that are released at the beginning of this phase. This conclusion also applies to $r \geqslant T_{m+1}$, which can anyways lead to further remaining execution time which is carried onward into the next phase.

Lemma. *After execution of all HI-jobs, the highest priority LO-task receives processor capacity in phase* $\varphi = 0, \ldots, {}^{T_m}/_{T_{m+1}}$ *depending on remaining execution times* $r_1, \ldots r_{\varphi-1}, r$ *of the preceding phases:*

$$\begin{aligned} A_{0\varphi}^{r_1 \ldots r_{\varphi-1} r} &= A_{0\varphi}^{r_1 \ldots r_{\varphi-1} 0} - r \quad \textit{with} \\ A_{0\varphi}^{r_1 \ldots r_{\varphi-1} 0} &= T_{m+1} - \sum_{i:\, \frac{\varphi T_{m+1}}{T_i} \in \mathbb{N}} X_i \end{aligned}$$

In the second step, we calculate the corresponding random variables for all other LO-tasks, resulting in success probabilities for each LO-job. We begin with the case of A_{00} assuming a negative value, meaning that the CPU is already fully loaded with an available capacity of 0. The first job of LO-task τ_{m+1} is therefore not running, thus having success probability $p_{10} = 0$. Remaining execution times represented by A_{00} remain unchanged. In the other case, the value a of $A_{00} - X_{m+1}$ describes, whether the LO-job is successful ($a \geqslant 0$) or unsuccessful ($a < 0$). Unsuccessful execution does not add to the value of p_{10} and no more processor time is available. Hence, those values are replaced with 0. Because LO-jobs are discarded at the end of their period, they do not contribute remaining execution time and the event tree $\tilde{A}_0$ remains unchanged. To consider cases with remaining execution being carried into the current phase, the event tree needs to be transformed for the highest-priority LO-task and aggregation for all lower-priority LO-tasks as well as modification of node random variables are required. We summarize those consequences in the following:

- For a discrete random variable Z with negative values, we form a partial distribution: Let $Z(0)$ be the partial distribution containing all non-negative values of Z. Further, let $Z(r)$ for $r > 0$ be the partial distribution solely containing the value r if Z contains the value $-r$. In both cases, the respective probabilities are copied from Z unchanged.

- The tree $\tilde{A}_0$ is transformed into the tree A_0 by splitting the root node A_{00} into nodes $A_{00}(r)$, analogously for all non-root nodes.
- For the highest-priority LO-task τ_{m+1}, partial distributions are calculated:

$$B^H_{1\varphi}(r) = A^H_{0\varphi}(r) - X_{m+1}$$

Summation leads to the success probability of a LO-job in phase φ. Now $B^H_{1\varphi}(r)$ is transformed into the final partial distribution $A^H_{1\varphi}(r)$ by replacing negative values in $B^H_{1\varphi}(r)$ with 0 and accumulating the accompanying probabilities.

We end up with a tree A_1, which is structurally identical to A_0, but the distributions kept at the nodes have changed. For every phase of τ_{m+1} we receive a probability based on the corresponding history H. Determining the success probabilities $p_{1\varphi}$ requires weighing those values with the product of the probabilities along the path from the root node to the considered leaf node.

For all further LO-tasks, we can continue in the same manner. The stochastic independence of execution times in successive phases is guaranteed by the separation into distinct cases in the event tree. Therefore, connected random variables can be added, causing related phases of A_1 to be aggregated, which leads to a new tree A_2.

For a generalized formulation, let $q^H_{j\varphi}(r)$ be the probability along the path from the root of the tree A_j to the considered leaf node and let the parameter r of $A^H_{j\varphi}(r)$ be called the state of the job. Then:

Proposition. *In case $T_{m+n} \leqslant T_m$, the success probability $p^H_{j\varphi}(r)$ of a job from LO-task τ_{m+j} in state r within phase φ and with history H for $\varphi = 0, \ldots, {}^{T_m}\!/_{T_{m+j}} - 1$, $j = 1, \ldots, n$ is*

$$p^H_{j\varphi}(r) = \Pr(B^H_{j\varphi}(r) \geqslant 0)$$

with

$$B^H_{1\varphi}(r) = A^H_{0\varphi}(r) - X_{m+1},$$

$$B^H_{j\varphi}(r) = \sum_{k=0}^{q_j} A^H_{j,q_j\varphi+k}(r) - X_{m+j},$$

$$\textit{for } q_j = {}^{T_{m+j}}\!/_{T_{m+j-1}}.$$

The final success probability of a job from LO-task τ_{m+j} in phase φ follows:

$$p_{j\varphi} = \sum_{H,r} q^H_{j\varphi}(r) \cdot p^H_{j\varphi}(r).$$

Further,

$$A^H_{j\varphi}(r) = \sum_{k=0}^{q_j} A^H_{j,q_j\varphi+k}(r) \dot{-} X_{m+j}$$

(operator $\dot{-}$ denotes non-negative subtraction).

An aggregated success probability p_j for LO-task τ_{m+j} can be given as

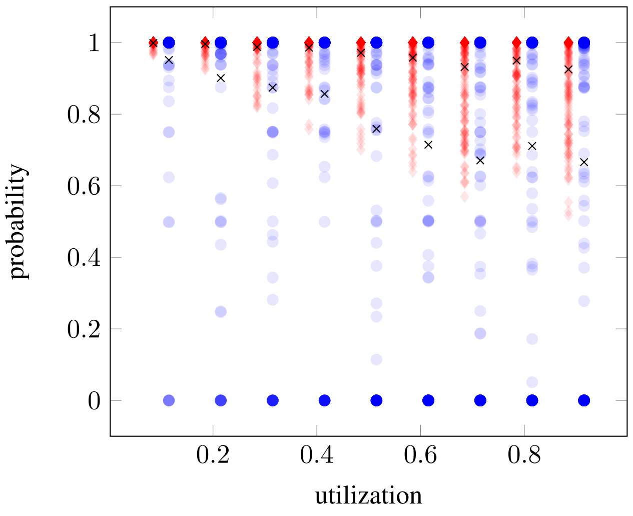

Figure 3. Average job success probability (red) and first job success probability (blue) for RMS in bands

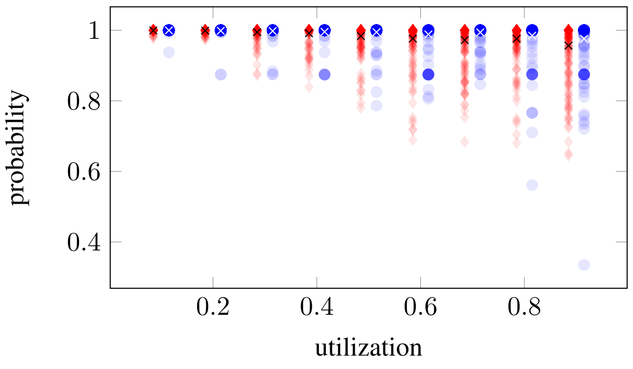

Figure 4. Average job success probability (red) and first job success probability (blue) for EDF

$$p_j = \frac{1}{{}^{T_m}\!/_{T_{m+j}}} \sum_{\varphi=0}^{{}^{T_m}\!/_{T_{m+j}}-1} p_{j\varphi}$$

using the assumption that phases occur uniformly distributed.

VI. Evaluation

In this section we show preliminary results of our per-job analysis of LO-tasks in mixed criticality systems. We determine success probabilities of LO-job execution. We compare our analysis results to critical instant response time analysis.

A. Task Generation

We chose $D = T$ from a list of 7 values that roughly follow a log-uniform distribution between 15 and 1000. $C(LO)$ is determined from a utilization generated by the UUnifast algorithm [27]. We vary the total utilization (see below). Tasksets have two criticality levels LO and HI, the chance for a task to have HI criticality is 50%. In this case $C(HI) = 1.6\,C(LO)$.

The pWCET distribution of each task can take values between $0.6\,C(LO)$ and $C(LO)$ (for LO-tasks) or $C(HI)$ (for HI-tasks). Values are uniformly spaced with a distance of $0.2\,C(LO)$, thus there are 3 values in LO-task distributions and 6 values in HI-task distributions. The corresponding probabilities start at 0.5 for $0.6\,C(LO)$ and are halved at each step except the last (so that the sum is 1). This way the pWCET distributions approximates an exponential tail distribution.

B. Results

For each utilization in $0.1, 0.2, \ldots, 0.9$ we generated 100 tasksets. They were scheduled twice, once with criticality-monotonic priorities and rate-monotonic ordering within each criticality band (Figure 3), and once with bands determined by the system criticality level (jobs of at least that criticality in the upper, all other in the lower band) and EDF priorities within the bands (Figure 4). In both figures, red marks denote aggregate task success probabilities, i.e., the average success probability across all jobs of a task, and blue marks denote the success probability at the critical instant (at time 0 in these examples). Marks are transparent to illustrate their distribution, crosses show the average within each column. Only LO jobs were considered in both plots, as high jobs must always finish in a valid schedule.

In Figure 3, where priorities are static, critical instant probabilities are a — sometimes very pessimistic — lower bound of the average probability. When priorities depend on the system criticality however, as in Figure 4, the critical instant with standard criticality does not provide a lower bound.

VII. Conclusion

In this work, we propose a new method for probabilistic analysis of low-criticality tasks. We implement our approach using *symbolic scheduling*. For restricted task sets, we also present an analytical solution. We use our analysis to show first results on success probabilities of low-criticality tasks after a criticality miss. We believe our analysis provides a useful new tool to designers of mixed-criticality systems.

Acknowledgments

This research was supported in part by the German Research Council (DFG) through the Cluster of Excellence *Center for Advancing Electronics Dresden* and through the German priority program 1648 *Software for Exascale Computing* via the research project *FFMK: A Fast and Fault-Tolerant Microkernel,* as well as by the Fonds National de la Recherche Luxembourg (FNR) through PEARL grant FNR/P14/8149128.

References

[1] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," in *28th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2007, pp. 239–243.

[2] A. Burns and R. I. Davis, "Mixed-criticality systems: A review," University of York, York, UK, Tech. Rep. 5th Edition, February 2015. [Online]. Available: http://www-users.cs.york.ac.uk/burns/review.pdf

[3] S. B. Alan, Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in *32nd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, November 2011, pp. 34–43.

[4] J. P. Lehoczky, "Real-time queueing theory," in *17th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 1996, pp. 186–195.

[5] D. Griffin and A. Burns, "Realism in statistical analysis of worst case execution times," in *10th International Workshop on Worst-Case Execution Time Analysis (WCET)*. OASIcs, July 2010, pp. 44–53.

[6] L. Cucu-Grosjean, "Independence: A misunderstood property of and for probabilistic real-time systems," in *Real-Time Systems: The Past, the Present and the Future*, N. Audsley and S. Baruah, Eds., March 2013, pp. 29–37.

[7] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *24th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, July 2012, pp. 91–101.

[8] R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean, "Analysis of probabilistic cache related pre-emption delays," in *25th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, July 2013, pp. 168–179.

[9] S. Edgar and A. Burns, "Statistical analysis of WCET for scheduling," in *22nd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2001, pp. 215–224.

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

[11] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis, "Static probabilistic timing analysis for real-time systems using random replacement caches," *Real-Time Systems*, vol. 51, no. 1, pp. 77–123, January 2015.

[12] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim, "PROARTIS: Probabilistically analyzable real-time systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2s, pp. 94:1–94:26, May 2013.

[13] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A new way about using statistical analysis of worst-case execution times," *SIGBED Review*, vol. 8, no. 3, pp. 11–14, September 2011.

[14] M. A. Iverson, F. Özgüner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, vol. 48, pp. 1374–1379, December 1999.

[15] L. David and I. Puaut, "Static determination of probabilistic execution times," in *16th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, June 2004, pp. 223–230.

[16] A. K. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *19th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 1998, pp. 123–132.

[17] Z. Guo, L. Santinelli, and K. Yang, "EDF schedulability analysis on mixed-criticality systems with permitted failure probability," in *21st International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, August 2015, pp. 187–196.

[18] C.-J. Hamann, J. Löser, L. Reuther, S. Schönberg, J. Wolter, and H. Härtig, "Quality-assuring scheduling: Using stochastic behavior to improve resource utilization," in *22nd IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 2001, pp. 119–128.

[19] C.-J. Hamann, L. Reuther, J. Wolter, and H. Härtig, "Quality-assuring scheduling," TU Dresden, Dresden, Germany, Tech. Rep. TUD-FI06-09, December 2006.

[20] K.-J. Lin, S. Natarajan, and J. W. S. Liu, "Imprecise results: Utilizing partial comptuations in real-time systems," in *8th IEEE Real-Time Systems Symposium (RTSS)*. IEEE, December 1987, pp. 210–217.

[21] D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran, "Probabilistic analysis for mixed criticality scheduling with SMC and AMC," in *4th International Workshop on Mixed Criticality Systems (WMC)*, November 2016.

[22] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, April 2010, pp. 13–22.

[23] S. Baruah, "Certification-cognizant scheduling of tasks with pessimistic frequency specification," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, June 2012, pp. 31–38.

[24] P. Huang, G. Giannopoulou, N. Stoimenov, and L. Thiele, "Service adaptions for mixed-criticality systems," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, January 2014, pp. 125–130.

[25] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *27th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, July 2015, pp. 259–268.

[26] M. Küttler, M. Roitzsch, C.-J. Hamann, and M. Völp, "Probabilistic analysis of low-criticality execution," TU Dresden, Dresden, Germany, Tech. Rep. TUD-FI17-02, November 2017. [Online]. Available: https://os.inf.tu-dresden.de/papers_ps/wmc2017-probmcs-tr.pdf

[27] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1-2, p. 129–154, May 2005.