TECHNISCHE BERICHTE
TECHNICAL REPORTS

ISSN 36629-211X

TUD-FI17-02 November 2017

M. Küttler, M. Roitzsch, Cl.-J. Hamann, M. Völp
Professur Betriebssysteme

Probabilistic Analysis of Low-Criticality Execution

PROBABILISTIC ANALYSIS OF LOW-CRITICALITY EXECUTION

Martin Küttler, Michael Roitzsch, Claude-Joachim Hamann[*]
Marcus Völp[†]

December 3, 2017

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.

[*]Operating Systems Group
Department of Computer Science
Technische Universität Dresden
01062 Dresden, Germany
Email: {mkuettle,mroi,hamann}@os.inf.tu-dresden.de
[†]CritiX: Critical and Extreme Security and Dependability Research Group
Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg
L-2721 Luxembourg
Email: marcus.voelp@uni.lu

1 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.

Drones for wood-fire detection combine flight control and image processing subsystems on one platform [2]. However, the image processing and classification tasks may be dropped to free up resources for the tasks concerned with flying and landing the vehicle safely. Other examples can be found in autonomous cars, where complex trajectory finding and obstacle avoidance algorithms [3] may have to yield resources to simpler safety-preserving fail-safes such as stopping the car before hitting an obstacle.

Often the assignment of tasks to criticality levels follows the separation between *safety-critical* and *mission-critical* functionality. Under normal operating conditions, both safety requirements and the mission objective should be accomplished. But whenever safety is at risk, mission-related functionality may be sacrificed.

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 [4]. 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) [5] 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 probabilistic execution time distribution. We describe the task model in Section 3.

This paper presents two approaches to provide probabilistic guarantees for low-criticality tasks. Our goal is to quantify, how low-criticality tasks behave in case of high-criticality jobs overrunning their optimistic low-criticality reservation. We calculate probabilities for jobs of low-criticality tasks meeting their deadline when the system operates in high criticality mode. We make the following contributions:

1. In Section 4, we present an analytical solution for restricted task sets with criticality-monotonic priority assignment and harmonic periods.

2. For general task sets with arbitrary periods and priority assignments, we introduce in Section 5 an algorithmic analysis approach we call *symbolic scheduling.* The algorithm extends the restricted analytical solution and is comparable to a specialized model checker, which symbolically evaluates possible schedules. It supports constrained deadline systems as well as changing priorities at job releases times and in case of criticality misses. Symbolic scheduling is thus applicable to a wider range of existing mixed-criticality scheduling regimes. We also describe purpose-built optimizations to curb the combinatorial explosion usually occurring when traversing all possible schedules.

In Section 6, 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 p of jobs to succeed in low criticality mode. Potential future Extensions to our model are discussed in Section 7 before we conclude in Section 8.

2 Related Work

This section surveys the large body of related works on probabilistic task models and schedulers, and on model-checking-like approaches for analyzing real-time systems.

2.1 Probabilistic Task Models

Lehoczky [6] was first to characterize execution times as random variables and to describe them through probability density functions. However, as realized by Griffin and Burns [7], 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) [8, 9, 10, 11, 12, 13, 14] 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. [15] present a technique based on random sampling for determining pWCETs. Iverson et al. [16] suggest a purely statistical analysis, whereas David and Puaut [17] propose a combined static and measurement-based analysis. For our analysis we propose a characterization of task execution times based on over-approximated probabilistic execution times, leading to an optimistic view on mixed-criticality systems [18] where low-criticality execution time estimates can deliberately disregard some exceptional behavior of high-criticality tasks, like fault recovery.

2.2 Probabilistic Scheduling

Statistical and probabilistic techniques have been used for real-time analysis in the past. In a way, Lehoczky [6] pioneered probabilistic mixed-criticality scheduling by regarding hard real-time tasks as highly and soft real-time tasks as less critical, offering only probabilistic guarantees to the latter. A more recent work along these lines is by Mollison et al. [19], who execute soft real-time tasks in a low-criticality band using global EDF whereas hard real-time tasks are scheduled by a partitioned EDF scheduler.

Support for multiple execution time estimates was introduced by Lin et al. [20] in their imprecise computation framework. A mandatory part of a task was scheduled with hard real-time guarantees whereas additional execution time was given to optional parts to improve the computation result. Later, Hamann et al. [21, 22] extended imprecise computations to derive budgets for optional parts that guarantee a certain completion probability. Our initial idea of characterizing low-criticality-tasks as optional parts of preceding high-criticality tasks [23] did not extend to general mixed-criticality systems due to task dependencies, which we resolve here with the help of schedule time slices.

In their probabilistic schedulability analysis, Guo et al. [24] consider a low-criticality miss rate for high-criticality tasks. They develop a scheduling algorithm and admission test to guarantee that the rate of a high-criticality task missing its low-criticality deadline is less than a permitted system failure probability. Our approach provides quantifiable probabilistic guarantees for low-criticality tasks when criticality misses occur, while maintaining the non-probabilistic high-criticality completion guarantees.

Atlas and Bestavros [25] also calculated task success rates. However, their method uses a different scheduling scheme: a job is not cancelled, when overrunning its deadline, but it is not eligible for release, if meeting the deadline cannot be guaranteed. Thus, a priori knowledge of exact execution times at each release instant is required, while our analysis only needs a known distribution.

Recent work on probabilistic mixed-criticality analysis by Maxim et al. [26] 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.

2.3 Analysis by Model-Checking

Our work extends such WCRT 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, optimized for our narrow task of performing a quantitative analysis of low-criticality job success rates. Salmon et al. [27] derive an analysis based on timed Petri Nets for classical periodic tasks with resource dependencies. Behrmann et al. [28] and Igna [29] gained valuable insights in scheduler design by modeling real-time systems as timed automata and checking them with UPPAAL.

3 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 [30] for this interpretation of the mixed-criticality framework. Our paper follows this interpretation.

3.1 Standard Model

In the standard deterministic model, a task $\tau = (L, T, D, C(\ell))$ is assumed to be strictly periodic with a criticality level L, a period T, a relative deadline D, and a worst-case execution time (WCET) $C(\ell)$ for each criticality level $\ell \leqslant L$. To simplify the presentation, we only discuss the dual criticality case, with the two criticality levels named LO and HI.

- The execution time bound is criticality-dependent, denoted by $C(LO)$ and $C(HI)$ with $C(LO) \leqslant C(HI)$. Other research also considers criticality-dependent interrelease times [31] or deadlines [32]. We limit ourselves to criticality-dependent execution times.
- While our analytical solution requires harmonic periods and implicit deadlines, symbolic scheduling does not restrict the relation of T and D, so $D \geqslant T$ is possible. We do not consider sporadic or aperiodic release processes.
- During execution of the task set, the system runs in either LO- or HI-criticality mode, initially starting in LO mode. Scheduling is performed according to an existing mixed-criticality priority regime, with a prior admission test ensuring the schedulability of the task set.
- 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 [33], but are not considered here.

The guarantee conveyed by the mixed-criticality system with levels $\ell \in \{LO, HI\}$ is that all jobs receive enough time between their release and deadline to complete their ℓ-level execution requirement $C(\ell)$, given that the following condition holds: No job of criticality higher than ℓ exceeds its execution time bound $C(\ell)$. In this paper, we contribute quantifiable probabilistic job execution, when this condition does not hold.

3.2 Probabilistic Model

We extend the standard task model to a probabilistic one similar to Maxim et al. [26]. In addition to the above parameters, each task is described by a probabilistic execution time X. We consider a task system of m HI-tasks τ_i and n LO-tasks τ_{m+j}. The tasks are thus described by the following parameters:

$$\begin{aligned} \tau_i &= (HI, T_i, D_i, X_i, C_i(LO), C_i(HI))\,, & i &= 1,\dots,m \\ \tau_{m+j} &= (LO, T_{m+j}, D_{m+j}, X_{m+j}, C_{m+j}(LO))\,, & j &= 1,\dots,n \end{aligned}$$

With $k = 1,\dots,m+n$, these parameters are:

T_k	period length (job inter-arrival separation)
D_k	relative deadline
X_k	job execution time, random variable with $X_k > 0,\ X_i \leqslant C_i(HI),\ X_{m+j} \leqslant C_{m+j}(LO)$
C_k	criticality-dependent execution time bounds for admission, $C_i(LO) \leqslant C_i(HI)$

The execution time is therefore characterized as a stochastic process. Stated more precisely, each job $J_{k,l}$ within one periodic task τ_k is described by its own random variable $X_{k,l}$. Because we imagine all jobs to run the same code in each period, we assume the per-job random variables within the same task to be independent and identically distributed. X_k is therefore used as one common representative of these random variables.

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 $X_k = [2 : 0.8,\ 5 : 0.2]$, 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 demand all job execution times to be bounded by the respective $C_i(HI)$ or $C_{m+j}(LO)$, so that a proper mixed-criticality admission performed using these bounds still yields deterministic rather than probabilistic guarantees, predicated by the rely condition. We use the probabilities only to analyze the success rates of LO-jobs after a criticality miss. They do not weaken the classical mixed-criticality guarantees stated above. We believe our method can be extended to soft-real-time mixed-criticality task sets, but we leave such an extension for future work.

4 Analytical Solution

We now consider a subset of mixed-criticality systems and analytically derive probabilities for successful completion of individual LO-jobs. For the analysis, we require the following restrictions:

- The tasks use implicit deadlines $D_k = T_k$.
- 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 the bands, rate monotonic priority assignment is used. However, between the bands, rate-monotonicity is not assumed. The longest HI-period may be longer than the shortest LO-period. Accumulating success probabilities across this discontinuity separates our analysis from traditional non-mixed-criticality analysis.

- Furthermore, we require harmonic periods across all tasks and criticality levels. The length of the hyperperiod for HI-tasks shall be named h, therefore $h = T_m$.

We lift these restrictions with symbolic scheduling in Section **??**, where we show an algorithmic solution for arbitrary periods and dynamic priority scheduling, including non-criticality-monotonic assignments.

Because we assume independent job execution and therefore independent random variables, we use the n-fold convolution of an execution time distribution X to model the consecutive execution of n jobs. We call this operation the n-fold sum ($n \in \mathbb{N}^+$) of X and write:

$$n * X := \sum_{k=1}^{n} X$$

When operating with random variables, constants are regarded as a degenerate distribution, which only takes a single value.

We continue by examining the success probabilities of LO-jobs in three cases of increasing complexity, where a formula-based analysis is tractable. Following that, we describe an algorithm that calculates success probabilities for all cases within the bounds of the restricted task model. We show these different and partially redundant approaches to offer easier solutions for special cases and to inform the reader of pitfalls along the route to a more general solution, with the most powerful and most complex being the symbolic scheduler described in the next section.

We start with the simple case of all LO-task periods covering the entire HI-hyperperiod h:

Case 1: $T_{m+1} \geqslant h$

Periods of LO-tasks are then integer multiples of h. For the shortest LO-task τ_{m+1}, we have

$$X_{HI} = \sum_{i=1}^{m} \frac{h}{T_i} * X_i$$

as the aggregate execution time of all HI-tasks within the HI-hyperperiod. Note that $\frac{h}{T_i} \in \mathbb{N}$. The probability of a job of LO-task τ_{m+j} completing its execution shall be p_j. We see:

$$p_1 = \text{Pr}\left(\tfrac{T_{m+1}}{h} * X_{HI} + X_{m+1} \leqslant T_{m+1}\right)$$

For other LO-tasks, we need to consider the respective higher-priority LO-tasks. Not just the successful jobs, but all jobs of higher-priority LO-tasks can influence execution, because all jobs are started first and then may be cancelled if they reach the end of their period. All HI-tasks and one job of the highest-priority LO-task thus have an aggregate random computation requirement S_1:

$$S_1 := \min\left(T_{m+1},\ \tfrac{T_{m+1}}{h} * X_{HI} + X_{m+1}\right)$$

From this, we derive:

$$p_2 = \text{Pr}\left(\tfrac{T_{m+2}}{T_{m+1}} * S_1 + X_{m+2} \leqslant T_{m+2}\right)$$

We now generalize:

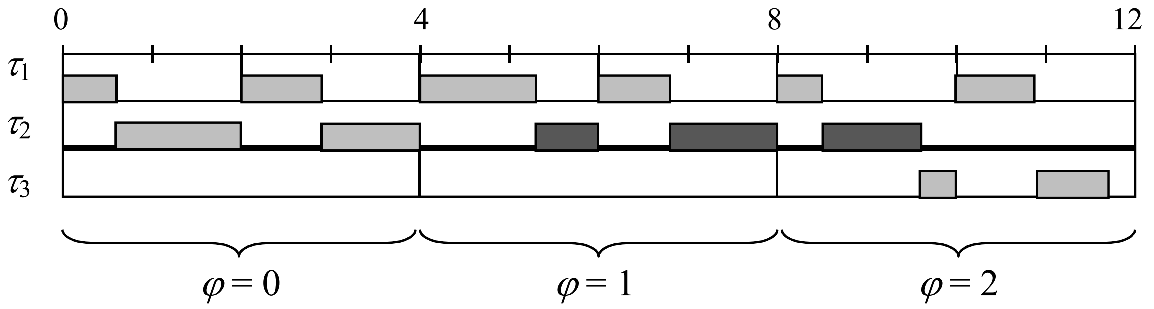

Figure 1: Job phases and remaining execution time of task τ_2 at the beginning of phase 1 of τ_3

Proposition. *Given $T_{m+1} \geqslant h$, the probability of successful execution p_j of a job of LO-task τ_{m+j} is:*

$$p_j = \Pr\left(\frac{T_{m+j}}{T_{m+j-1}} * S_{j-1} + X_{m+j} \leqslant T_{m+j}\right),\ j = 1, \dots, n$$
$$S_j = \min\left(T_{m+j},\ \frac{T_{m+j}}{T_{m+j-1}} * S_{j-1} + X_{m+j}\right),\ S_o = X_{HI}$$

In the more complex case of LO-period lengths being shorter than h, the success probability of a LO-job depends on its relative position within the HI-hyperperiod. We shall call this position the *phase* φ of a job, counting from 0 to simplify indexing in formulae further below. Phase φ of a LO-job of τ_{m+j} thus starts with its release at φT_{m+j} and ends with the deadline $(\varphi + 1)T_{m+j}$. Figure 1 illustrates two HI-tasks τ_1, τ_2 and a LO-task τ_3 with $T_1 = 2,\ T_2 = 12,\ T_3 = 4$.

Calculating the success probability of a LO-job needs the execution time of all HI-jobs released within its phase and the remaining execution time of HI-jobs released before the phase. Those latter jobs are released by a HI-task with a period longer than that of the LO-job in question and reach into the LO-job phase. Note that this remaining execution time is itself a random variable and any concrete realization may cause the HI-job to end prematurely.

We first look at a simplified case, where the system contains only a single LO-task next to multiple HI-tasks. But before, we observe that we can assume $T_1 \geqslant T_{m+1}$ without loss of generality. Otherwise, a largest index $s > 1$ with $T_s \leqslant T_{m+1}$ would exist, such that all tasks $\tau_1, \dots, \tau_s$ can be replaced by a single task $\tilde{\tau}$ with $\tilde{X} = \sum_{i=1}^{s} {T_{m+1}}/{T_i} * X_i$, in other words the total execution time of all these HI-tasks within one phase of the highest-priority LO-task τ_{m+1}. The mixed-criticality admission guarantees that these HI-jobs never exceed the ends of their periods, so their remaining execution time is 0 due to the harmonic structure of the periods.

Case 2: $T_{m+1} < h,\ n = 1$

Figure 2 shows the general situation and illustrates the method. We need to calculate the aggregate execution time of all HI-tasks within a phase of the LO-task as well as the remaining execution times of HI-jobs at the beginning of the following phase, from which we derive the LO-job success probabilities. We perform this calculation by way of schedule time slices of duration T_{m+1}. We successively determine the following variables:

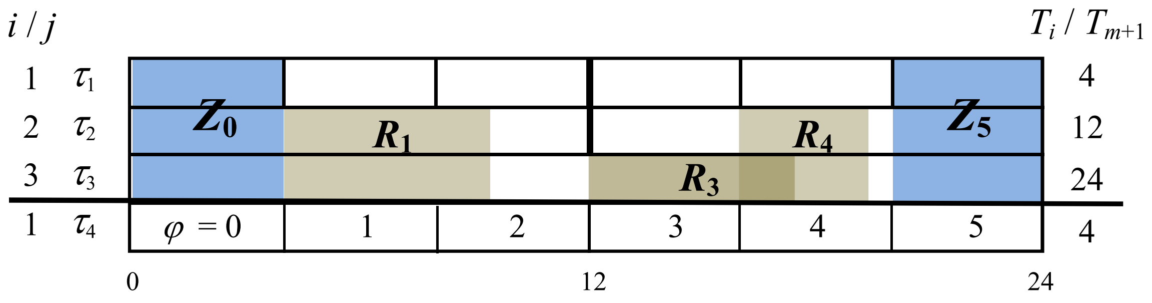

Figure 2: Aggregating execution times

Z_φ	aggregate execution time of all HI-tasks $\tau_1, \ldots, \tau_m$ during phase φ of LO-task τ_{m+1}
R_φ	remaining execution time of the HI-jobs at the beginning of phase φ of LO-task τ_{m+1}
p_φ	success probability of job in phase φ of LO-task τ_{m+1}

We now walk through the calculation using the example from Figure 2. The aggregate time demand Z_0 of the HI-tasks τ_1 to τ_3 during phase $\varphi = 0$ is the sum of their corresponding random execution times $X_1+X_2+X_3$. This sum is restricted by the phase length of $4 = T_{m+1}$. Therefore:

$$Z_0 = \min\left(4, X_1 + X_2 + X_3\right)$$

Accordingly, the remaining execution time is zero, if $X_1 + X_2 + X_3$ does not exceed the length of one phase. Otherwise, the remaining execution time is the sum, reduced by the phase length:

$$R_1 = (X_1 + X_2 + X_3) \dot{-} 4$$

The operator $\dot{-}$ denotes a non-negative subtraction, i.e., negative differences are replaced with zeroes.

In Figure 2, the job of τ_3 is neither finished at time $t = 12$, nor at time $t = 16$, where the remaining execution time of the second job of τ_2 needs to be added. In general, Z_φ follows from the sum of remaining execution times R_φ of the preceding HI-jobs as well as the execution time of those HI-jobs, whose period starts at the beginning of the current phase.

Lemma. *Given $T_{m+1} < h$ for $\varphi = 0, \ldots, \frac{h}{T_{m+1}} - 1$ and $R_0 = 0$, the aggregate execution time Z_φ and remaining execution times $R_{\varphi+1}$ of the HI-tasks $\tau_1, \ldots, \tau_m$ at the beginning of phase φ of task τ_{m+1} are:*

$$\begin{aligned} Z_\varphi &= \min\left(T_{m+1}, R_\varphi + \sum_{i:\frac{\varphi T_{m+1}}{T_i} \in \mathbb{N}} X_i\right) \\ R_{\varphi+1} &= \left(R_\varphi + \sum_{i:\frac{\varphi T_{m+1}}{T_i} \in \mathbb{N}} X_i\right) \dot{-} T_{m+1} \end{aligned}$$

Note that $\sum_{i:...} X_i$ can be zero, in Figure 2 for Z_5 in case T_1 would be 12.

The success probability p_φ for a job of LO-task τ_{m+1} in phase φ can now be derived. In Figure 2, we see:

$$p_1 = \Pr\left(Z_1 + X_4 \leqslant 4\right), \quad \text{with } 4 = T_{m+1}$$

Proposition. *For the LO-task τ_{m+1} with $T_{m+1} < h$, the probability of successful execution p_φ of the job in phase φ for $\varphi = 0, \ldots, \frac{h}{T_{m+1}} - 1$ is:*

$$p_\varphi = \Pr\left(Z_\varphi + X_{m+1} \leqslant T_{m+1}\right), \quad \textit{with } Z_\varphi \textit{ according to the lemma above}$$

This proposition also applies to task τ_{m+1}, if the system consists of more than one LO-task.

A global success probability p_1 for the LO-task τ_{m+1} can be stated as

$$p_1 = \frac{1}{{}^{h}\!/_{T_{m+1}}} \sum_{\varphi=0}^{{}^{h}\!/_{T_{m+1}}-1} p_\varphi$$

assuming that the phases occur uniformly distributed in time.

Unfortunately, the method of slicing the schedule into phases cannot be applied to multiple LO-tasks. The aggregate execution time of HI-tasks τ_1 to τ_3 during the first two phases of τ_4 appears to be $Z_0 + Z_1$, but this would be wrong. We shall illustrate with an example: Imagine a system containing a single job with an execution time distribution X encompassing the possible values 1 and 4. During a period of length 4, exactly these two values can occur. If we divide this period into two equal-length phases, the possible execution times in the first phase are 1 and 2, remaining execution times in the second are 0 and 2. The sum of both random variables now also enables the aggregate execution time $2 + 0 = 2$, contradicting X. This effect is caused by the remaining execution time in a phase depending on the execution time in the preceding phase. Both random variables are therefore not independent and can thus not be added by a simple convolution. Formally, a conditional probability is needed, which can be calculated by differentiating possible cases. We now demonstrate this method with a single HI-task.

Case 3: $T_{m+n} \leqslant h,\ m = 1$

Differing from the formalism introduced in Section 3, we denote the HI-task with τ_0, its execution time X_0, and its period T_0. LO-tasks are named τ_j with $j = 1, \ldots, n$. We determine the following quantities, dependent on the value x of X_0:

$p_{j0}(x)$	success probability of job in phase 0 of LO-task τ_j
$U_{j0}(x)$	execution time consumed by tasks $\tau_1, \ldots, \tau_j$ in phase 0 of LO-task τ_j

The corresponding values $p_{j\varphi}(x)$ and $U_{j\varphi}(x)$ for an arbitrary phase φ of LO-task τ_j immediately follow due to the harmonic periodicity.

Figure 3 displays an example.

For the first LO-task, we have:

$$p_{10}(x) = \Pr\left(x + X_1 \leqslant T_1\right), \quad U_{10}(x) = \min\left(T_1,\ x + X_1\right)$$

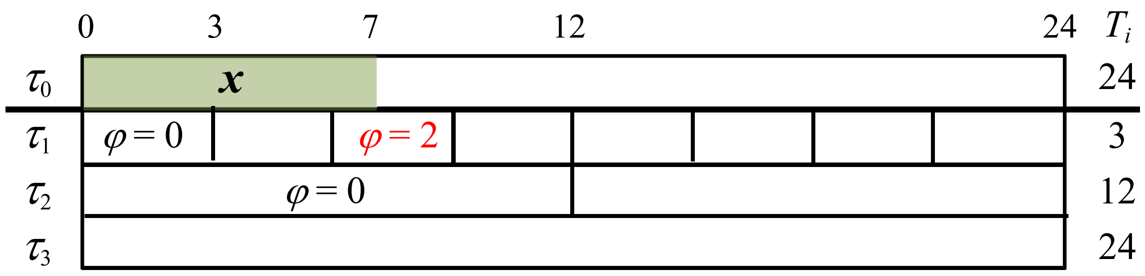

Figure 3: System with multiple LO-tasks

Specifically, we can confirm:

$$p_{10}(0) = \Pr\left(X_1 \leqslant T_1\right) = 1, \quad U_{10}(0) = \min\left(T_1,\ X_1\right) = X_1,$$
$$p_{10}(x) = 0, \quad U_{10}(x) = T_1 \quad \text{for } x > T_1$$

Values for other phases follow due to the mentioned periodic repetition. For example, the case of $x = 7$ in phase 2 of τ_0 is equal to the case $x = 1 = 7 - 2 \cdot 3$ in phase 0. This result enables a simple, closed representation for the other phases of τ_1:

$$p_{1\varphi}(x) = p_{10}(x \dot{-} \varphi T_1), \quad U_{1\varphi}(x) = U_{10}(x \dot{-} \varphi T_1)$$

Examining τ_2 in phase 0 clarifies the structure of the following formulae. They consist of three components:

1. phases of higher priority LO-tasks that are completely covered by X_0 and where no LO-jobs are running,

2. followed by one partially covered phase, where higher priority LO-jobs are running and consume processor time $U_{j0}(x)$,

3. and finally phases, where the HI-job has finished and only LO-jobs are executed with total processor time V_j. These jobs will always complete due to the successful LO-admission.

In Figure 3, we can observe for the LO-task τ_2 in phase 0 at $x = 7$:

$$p_{20}(7) = \Pr\left(6 + U_{12}(7) + 1 * X_1 + X_2 \leqslant 12\right)$$

So in general with $k = \left\lfloor \frac{x}{T_1} \right\rfloor$ and $V_1 = X_1$:

$$p_{20}(x) = \Pr\left(kT_1 + U_{1k}(x) + \left(\tfrac{T_2}{T_1} - 1 - k\right) * V_1 + X_2 \leqslant T_2\right)$$
$$U_{20}(x) = \min\left(T_2,\ kT_1 + U_{1k}(x) + \left(\tfrac{T_2}{T_1} - 1 - k\right) * V_1 + X_2\right)$$

This also holds for $x > T_2$, due to $\frac{T_2}{T_1} \in \mathbb{N}$ we have $kT_1 \geqslant T_2$ and thus $p_{20}(x) = 0$ and $U_{20}(x) = T_2$. Results for other phases follow from the periodic repetition. The general situation is:

Lemma. *Given $T_j \leqslant T_0$ for a LO-task τ_j in a system with one HI-task τ_0, the probability of successful execution $p_{j\varphi}(x)$ of a job of LO-task τ_j in phase φ for $\varphi = 1, \ldots, {}^{T_n}/_{T_j} - 1$ depending on a value x from X_0 is:*

$$p_{10}(x) = \Pr\left(x + X_1 \leqslant T_1\right), \qquad U_{10}(x) = \min\left(T_1, x + X_1\right)$$
$$p_{1\varphi}(x) = p_{10}(x \dot{-} \varphi T_1), \qquad U_{1\varphi}(x) = U_{10}(x \dot{-} \varphi T_1)$$

and with $j = 2, \ldots, n$ and $k = \left\lfloor \frac{x}{T_{j-1}} \right\rfloor$

$$\begin{aligned}
p_{j0}(x) &= \Pr\left(kT_{j-1} + U_{j-1,k}(x) + \left(\tfrac{T_j}{T_{j-1}} - 1 - k\right) * V_{j-1} + X_j \leqslant T_j\right) \\
U_{j0}(x) &= \min\left(T_j,\ kT_{j-1} + U_{j-1,k}(x) + \left(\tfrac{T_j}{T_{j-1}} - 1 - k\right) * V_{j-1} + X_j\right) \\
V_j &= \tfrac{T_j}{T_{j-1}} * V_{j-1} + X_j, \quad \textit{with } V_1 = X_1 \\
p_{j\varphi}(x) &= p_{j0}(x \dot{-} \varphi T_1), \\
U_{j\varphi}(x) &= U_{j0}(x \dot{-} \varphi T_1).
\end{aligned}$$

To calculate overall success probabilities for LO-jobs $p_{j\varphi}$ independent of individual x, we need to include the probabilities of those x to occur within X_0.

Proposition. *Given a system with a single HI-task τ_0 and n LO-tasks $\tau_1, \ldots, \tau_n$ with $T_j \leqslant T_0$, the probability of successful execution $p_{j\varphi}$ of a job of τ_j in phase φ is:*

$$p_{j\varphi} = \sum_{x \in X_0} \Pr(X_0 = x) \cdot p_{j\varphi}(x)$$

with $\varphi = 0, \ldots, \frac{T_j}{T_{j-1}} - 1$, $j = 1, \ldots, n$ and $p_{j\varphi}(x)$ according to the above lemma.

Finally, we address systems consisting of arbitrary numbers of HI- and LO-tasks, but with no LO-task period exceeding the longest HI-task period.

Case 4: $T_{m+n} \leqslant T_m = h,\ m, n$ arbitrary

A potential solution would be to combine all HI-tasks into one single virtual HI-task by aggregating their random variables and then continuing like in the previous case. However, this solution would not work, because it is not possible to add the execution times of HI-tasks in consecutive periods. To show this, let us consider four tasks with $m = n = 2$. The HI-tasks τ_1 and τ_2 shall have periods of 8 and 16, the LO-tasks τ_3 and τ_4 as well. The HI-tasks can execute for 2 or 4 time units in the case of τ_1, and 1 or 5 time units in the case of τ_2. Both LO-tasks execute for 3 time units.

Assuming that τ_1 uses 2 time units in both its periods and τ_2 uses 5 time units. As shown in the left part of Figure 4, the job of τ_4 can finish successfully. If however, the execution of τ_1 takes 4 time units for each job and τ_2 uses one time unit, then the execution of τ_4 will be truncated, despite the fact that the combined time usage of τ_1 and τ_2 is 9 in both cases.

Consequently, general task systems can only be analyzed by distinguishing all possible cases. We base this analysis on event trees. In step one, we construct the event tree $\tilde{A}_0$ describing all

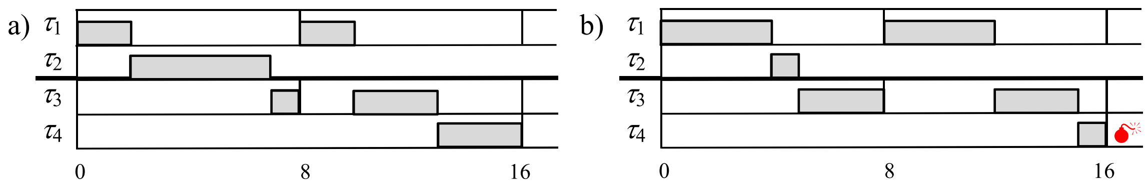

Figure 4: Success of LO-tasks depending on time allocation of HI-tasks

HI-tasks in schedule slices of length T_{m+1}. Nodes of this tree are random variables A, edges within the tree are annotated with the probabilities of the respective case. The random variables, which we assume to be discrete, have the following semantics: 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 $-a$.

We illustrate our approach using the following example task set with $m = n = 2$:

$$\begin{aligned} HI: \quad & \tau_1: \quad T_1 = 8, X_1 = [2:0.8,\ 5:0.2] \\ & \tau_2: \quad T_2 = 32, X_2 = [1:0.6,\ 11:0.4] \\ & \text{the lower value of } X_i \text{ shall be } C_i(LO) \\ LO: \quad & \tau_3: \quad T_3 = 8, X_3 = [1:0.9,\ 3:0.1] \\ & \tau_4: \quad T_4 = 16, X_4 = [2:0.7,\ 4:0.3] \end{aligned}$$

We denote the resulting random variables $A_{0\varphi}^{r_1 \ldots r_\varphi}$, with φ being the phase of τ_3 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_3 we have:

$$A_{00} = 8 - (X_1 + X_2) = [5:0.48,\ 2:0.12,\ -5:0.32,\ -8:0.08]\,,$$

which determines the root node of the event tree shown in Figure 5. To simplify the figure, we do not show probabilities for the values within the nodes. Here and in the following, the edge annotations for non-negative values of random variables result from adding the probabilities. We now determine A_{01}^r with $r = 0, 5, 8$ during phase $\varphi = 1$ of τ_3:

$$A_{01}^0 = 8 - X_1, \qquad A_{01}^5 = A_{01}^0 - 5, \qquad A_{01}^8 = A_{01}^0 - 8$$

Now we can continue with:

$$A_{02}^{00} = A_{01}^0, \quad A_{02}^{50} = A_{02}^{00}, \quad A_{02}^{52} = A_{02}^{00} - 2, \quad A_{02}^{82} = A_{02}^{00} - 2, \quad A_{02}^{85} = A_{02}^{00} - 5$$
$$A_{03}^{000} = A_{01}^0 = A_{03}^{500} = A_{03}^{520} = A_{03}^{820} = A_{03}^{850}, \quad A_{03}^{852} = A_{03}^{850} - 2$$

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 $r \geqslant T_{m+1}$,

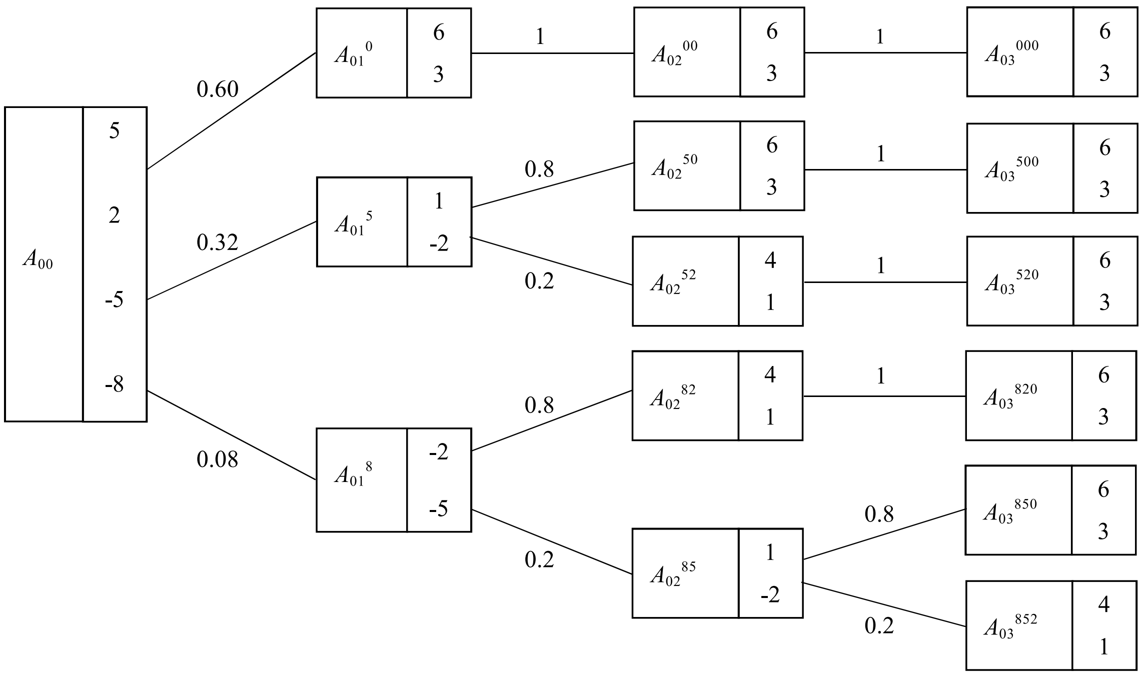

Figure 5: $\tilde{A}_0$ – Available execution times for highest-priority LO-task and remaining execution times of HI-tasks

which can anyways lead to further remaining execution times (A_{01}^{5} and A_{01}^{8}) which are carried onward into the next phase.

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

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

We have chosen a representation for the lemma that allows efficient calculation. Evidently we have $A_{00} = T_{m+1} - \sum_{i=1}^{n} X_i$.

In the second step, we calculate the corresponding random variables for all other LO-tasks, resulting in success probabilities for each LO-job. The simple difference approach $A_{10} = A_{00} - X_3$ unfortunately leads to wrong results, which we substantiate at the end of the following explanation of the correct formulation.

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 $A_{00} - X_3$ 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 nodes of $\tilde{A}_0$ as shown in Figure 6. If successor nodes of a node are split up as well, the original probability is copied to both branches, e.g., $A_{00}(5)$ with the successors $A_{01}^5(0)$ and $A_{01}^5(2)$.
- For the highest-priority LO-task, partial distributions are calculated:

  $$B_{1\varphi}^H(r) = A_{0\varphi}^H(r) - X_3$$

  Summation leads to the success probability of a LO-job in phase φ. Now $B_{1\varphi}^H(r)$ is transformed into the final partial distribution $A_{1\varphi}^H(r)$ by replacing potential negative values in $B_{1\varphi}^H(r)$ with 0 and accumulating the accompanying probabilities. For all lower-priority LO-tasks, the procedure is the same.

It may be irritating that we allow distribution functions of the form $[0 : 0.8]$. These partial distributions enable a relatively simple, closed form representation and saves us from distinguishing a lot of extra cases. Also, we write $Z(0)$ even if Z does not contain any negative values. We note that partial distributions do not necessarily represent a random variable, because their probabilities are less than 1. Also note that $A_{1\varphi}^H(r)$ can be calculated directly as $A_{1\varphi}^H(r) = A_{0\varphi}^H(r) \dot{-} X_3$ and analogously for other LO-tasks. But evaluation of this equation leads to an additional convolution, which our approach avoids.

We end up with a tree A_1, which is structurally identical to A_0, but the distributions kept at the nodes have changed. We continue with our example and obtain:

$$B_{10}(0) = A_{00}(0) - X_3 = [4 : 0.432,\ 2 : 0.048,\ 1 : 0.108,\ -1 : 0.012]$$

This immediately gives us $p_{10}(0) = 0.588$ and further:

$$A_{10}(0) = [4 : 0.432,\ 2 : 0.048,\ 1 : 0.108,\ 0 : 0.012]$$

From which follows:

$$p_{10}(5) = p_{10}(8) = 0, \qquad A_{10}(5) = [0 : 0.32]\,, \qquad A_{10}(8) = [0 : 0.08]$$

and thus a success probability of $p_{10} = 0.588$.

The procedure continues for the other phases in the same way:

$$B_{11}^5(0) = A_{01}^5(0) - X_3 = [1 : 0.8] - [1 : 0.9,\ 3 : 0.1] = [0 : 0.72,\ -2 : 0.08]\,,$$

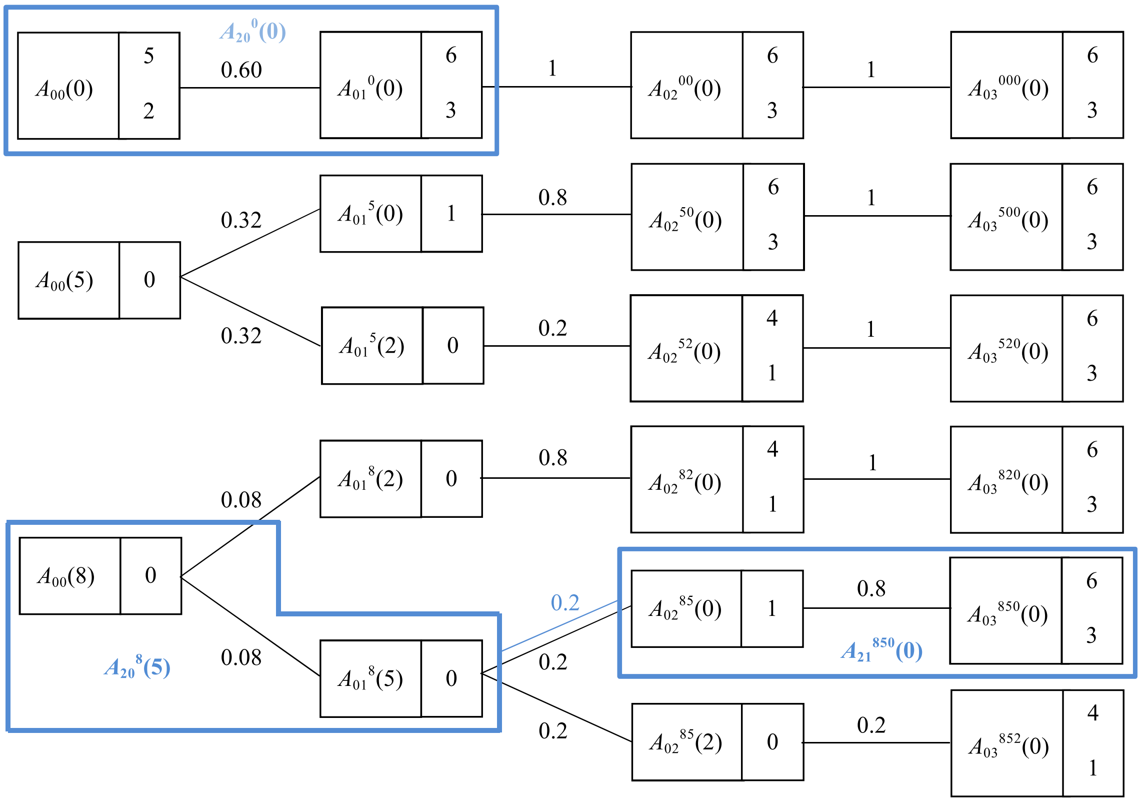

Figure 6: A_0 – Available execution times for highest-priority LO-task and aggregation for second LO-task; blue boxes represent nodes of the tree A_2

resulting in $p^5_{11}(0) = 0.72$ and $A^5_{11}(0) = [0 : 0.8]$. We then get $p^0_{11}(0) = 1$, all other success probabilities are 0, such that weighing those values with the product of the probabilities along the path from the root node finally yields $p_{11} = 0.8304$.

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. Therefore, we calculate for phase $\varphi = 0$ of τ_4:

$$B^0_{20}(0) = \big(A_{10}(0) + A^0_{11}(0)\big) - X_4$$

We obtain $p^0_{20}(0) = 0.590544$. We continue with

$$B^5_{20}(0) = \big(A_{10}(5) + A^5_{11}(0)\big) - X_4$$

and so on. Here, all resulting probabilities are 0, leading to a total of $p_{20} = 0.590544$. Accordingly, we determine $p_{21} = 0.99032576$. The necessary weights result from the products of the probabilities along the paths from the root node to the considered leaf node, in our example $q^{850}_{21}(0) = 0.08 \cdot 0.2$ for $A^{850}_{21}(0)$. Because τ_4 is the final task, the A-variables are not relevant here. All values presented here for this example where confirmed by the symbolic scheduler, which we present in the next section.

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. Symbols for probability distributions also denote their corresponding node in the tree. 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,

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

We now return to the wrong solution $A_{10} = A_{00} - X_3$ mentioned above, which supposedly describes the time remaining after executing the first LO-task in phase $\varphi = 0$. In our example we would see:

$$A_{00} - X_3 = [4 : 0.432,\ 2 : 0.048,\ 1 : 0.108,\ -1 : 0.012,$$
$$-6 : 0.288,\ -8 : 0.032,\ -9 : 0.072,\ -11 : 0.008]$$

The first four values necessarily match $B_{10}(0)$, the other values are incorrect, because they stem from negative values of A_{00}. The remaining processor time is never negative but 0 in this case. However, aggregating all negative values — as in the transition from $B_{10}(0)$ to $A_{10}(0)$ — is again wrong, because these values originate from different system states that need to be handled separately.

Case 5: $T_{m+n} > T_m = h$

Finally, we sketch an approach for the remaining case of unrestricted periods in relation to the hyperperiod h of the HI-tasks. Let s be an index with $1 \leqslant s < n$ and $T_{m+s} \leqslant h$, $T_{m+s+1} > h$. In the case $T_{m+s} < h$, we form a fictitious task $\tau_{m+s+1/2}$ with period h and a constant execution time of 0 with probability 1 to calculate the partial distributions $A^H_{m+s+1/2,0}(r)$. We weigh the probabilities with the corresponding path probabilities and concatenate the tables of possible values, coalescing potential duplicates. The sum of all path probabilities in one phase is now always

1, the result therefore represents a random variable A_{HP} of processor time remaining within the hyperperiod after executing the tasks $\tau_1, \ldots, \tau_{m+s}$. The random variable $Z_{HP} = h - A_{HP}$ thus describes the consumed processor time. Z_{HP} is used instead of X_{HI} for tasks beginning with τ_{m+s+1}. For $T_{m+s} = h$, no such fictitious task is needed.

Even in this most general case, our analysis is limited to harmonic periods and criticality-monotonic priority assignment. In the following section, we therefore present a novel algorithmic approach that is applicable without these restrictions.

5 Symbolic Scheduling

An obvious way to obtain approximate success rates of probabilistic task systems is a Monte-Carlo simulator: Given distributions of execution times of all jobs, a concrete schedule is executed with execution times randomly drawn from these distributions. With enough such samples the probability of a job finishing successfully is approximately the ratio of samples in which it succeeded to the number of samples taken.

We propose the concept of *symbolic scheduling,* which we present here and which we have implemented and published (https://github.com/mkuettler/symbolic-scheduler). Symbolic scheduling draws on ideas from such a simulation, in that it tracks runs of jobs to figure out which meet their deadline. Unlike the simulator, though, it does not randomly choose execution times, but it keeps track of every possible execution and its probability. For discrete probabilities there is a conceptually simple but computationally expensive way to do this: For every job, try every possible runtime separately. This leads to a tree, where each path from the root to a leaf is a possible execution 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 equivalencies 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 important difference. Instead, it keeps multiple executions combined in one path as long as possible.

Symbolic scheduling will insert at most three branches at any node because there are three distinct outcomes for a job: It could finish, it could experience a criticality miss, or it could hit a scheduling event, e.g., its deadline or the release of a job of higher priority. Readers familiar with model checkers will recognize that symbolic scheduling could be implemented using a model checker. We refrained from that idea for two main reasons: First, implementing the algorithm in a general purpose language (C++ in this case) was easier for us than creating a formal model. This clearly is a personal preference, although one we expect many people to share. Secondly, we expected to get much better performance out of our own code. Our algorithm does not match the way general model checkers would approach the problem, because we only need to solve a much more specific problem.

We will first explain a simplified version of the algorithm using an example, followed by the main ideas for extending to the general case.

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

Figure 7: Event tree for the presented example

5.1 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 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 we use partial distributions, so the accumulated probability 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 7.

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.

5.2 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.
- $\text{sum}(d)$ denotes the total probability of all values of d. Thus $\text{sum}(d) = \text{sum}(d \Lsh x) + \text{sum}(d \Rsh x)$ 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 $\text{sum}(d_1 \cup d_2) = \text{sum}(d_1) + \text{sum}(d_2)$, which must be $\leqslant 1$ for this operation to make sense.
- $d \rightarrowtail x := (d \Rsh x) \cup [x : \text{sum}(d \Lsh x)]$

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.

5.3 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 $[s : \text{sum}((t + X(J)) \Rsh s)]$, 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:
$$t_{\text{next}} = t \Rsh s \cup \big[s : \text{sum}\big((t + X(J)) \Rsh s\big) - \text{sum}(t \Rsh s)\big] .$$
  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 the two cases, updates the time and ready list, and repeats.

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)
    }
}
```

5.4 General Case

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.

Tracking criticality misses complicates the algorithm considerably, so we only provide a rough description here. Every job branches into a maximum of three possible situations:

1. The job finishes in time. This case is similar to the description above, but it may only include times for which neither a scheduling event nor a criticality miss occurs.

2. The job hits a scheduling event. Unless it is the deadline, the job must be updated to continue later. This update includes modifying the criticality miss instant and the remaining runtime. The time to the criticality miss turns it into a distribution, because the time this job ran already is a distribution.

3. A criticality miss happens before a scheduling event. Like in the second case, the job is interrupted but not removed from the system. Thus, a modified version with new probabilistic execution time and criticality miss instant must be added. Also, the criticality mode of the system changes, which might alter priorities for all following jobs.

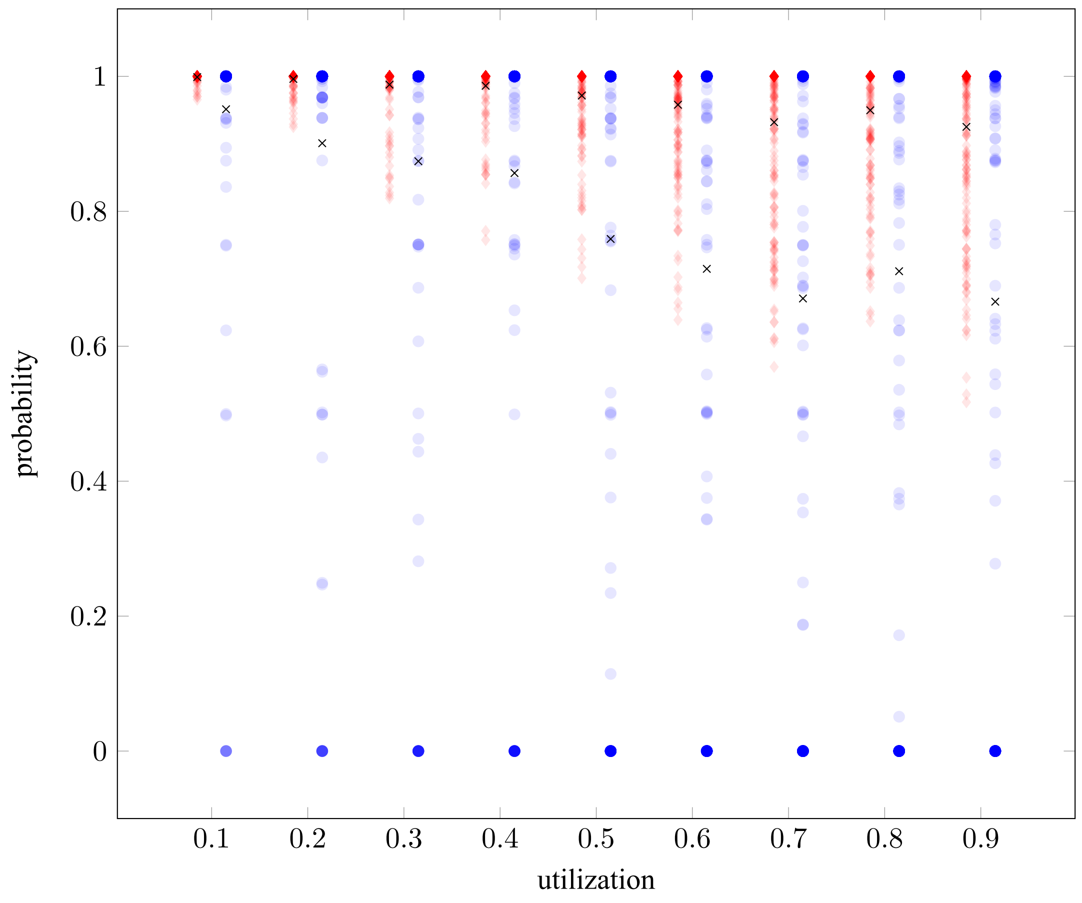

Figure 8: Average job success probability (red) and first job success probability (blue) for RMS in bands

The second point hints at one of the most difficult challenges about this scheme: Whereas initially, jobs have fixed criticality miss instants, which are part of the task description, the symbolic scheduler has to handle partly run jobs, which have a distribution as criticality miss instant. Therefore, jobs are described by multiple distributions, which are not independent. Special care has to be taken to never combine these. Also, the convolution operation we denoted as $+$ does not have an easily computable inverse operation, which complicates several conceptually simple ideas.

6 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.

6.1 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 [34]. We

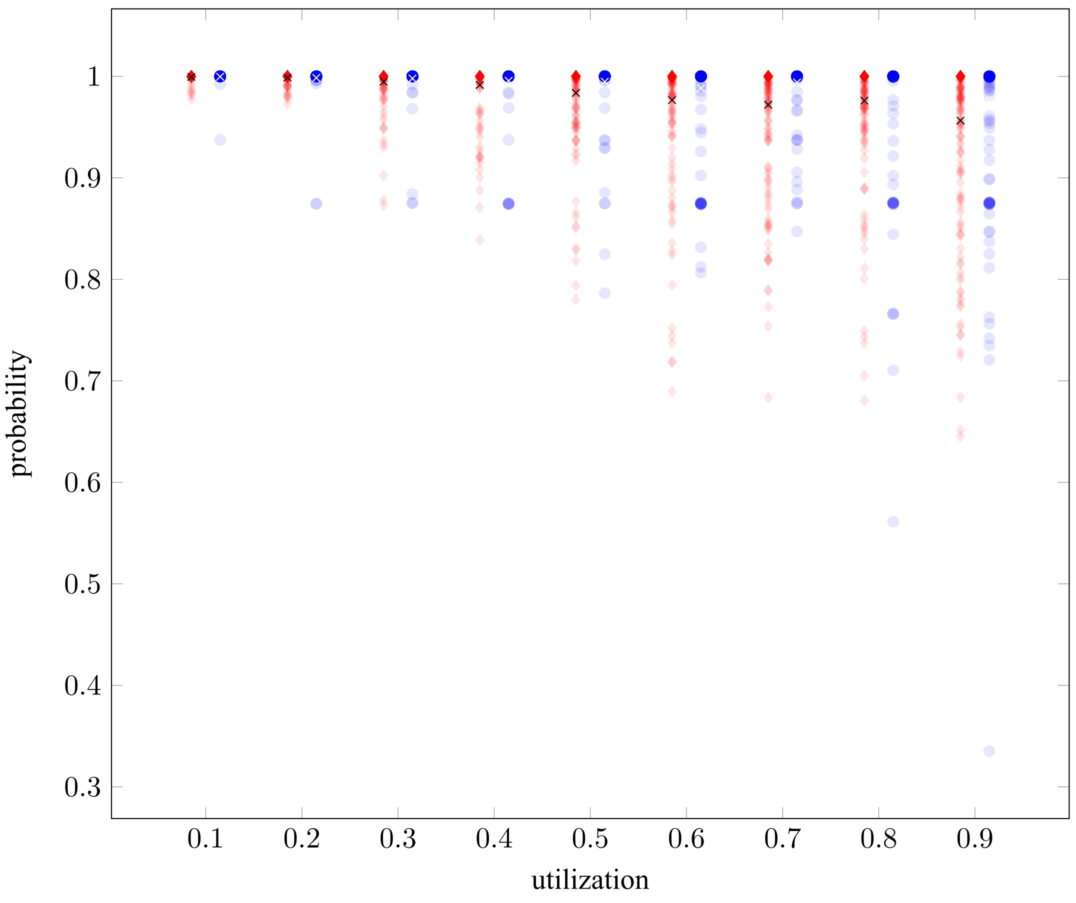

Figure 9: Average job success probability (red) and first job success probability (blue) for EDF

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.

6.2 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 8), 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 9). 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 8, 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 9, the critical instant with standard criticality does not provide a lower bound.

7 Model Extensions

Our task model and both the analytical and algorithmic analysis lend themselves to three interesting extensions, which we would like to propose as avenues for future research.

1. When estimating an execution time distribution for LO-tasks, this estimate may be reliable only up to a value x_{max}. An insecurity remains that this value may be exceeded at runtime with a small probability p_{exc}, but it is unknown, how large this exceedance might be. However, in a real system the job will at the latest be cancelled at the end of its period. Such a situation can be modeled by using the estimated distribution until x_{max}, scaled to a cumulative probability $1 - p_{exc}$ and adding the period length of the task as an extra value to the distribution with probability p_{exc}. The same approach can be applied to HI-tasks, if scheduled as soft-real-time mixed-criticality tasks.

2. Our model allows executing task sets with probabilistic guarantees, which deterministic mixed-criticality admission would reject. Take the following example of one HI-task τ_1 and one LO-task τ_2 with these parameters:

$$\begin{aligned} T_1 = 4,\, X_1 &= [1:0.3,\ 2:0.5,\ 3:0.2]\,, & C_1(HI) = 3 \\ T_2 = 2,\, X_2 &= [1:0.9,\ 2:0.1]\,, & C_2(LO) = 2 \end{aligned}$$

   Even with $C_1(LO) = 1$, this task set is unschedulable, due to $1 + 2 \cdot 2 > 4$. If it is executed with criticality-monotonic priorities, our analysis shows τ_2 to successfully execute in its two phases with 27% and 98%, respectively. In average more than every other job of τ_2 succeeds, which a system designer may consider adequate.

3. The presented analysis determines success probabilities from the task parameters. A modification to our method can control the success rate to a desired value for each LO-task, if feasible. Every LO-task receives an execution budget b and jobs are cancelled when it is depleted, causing lower-priority jobs to receive more execution time. Success probabilities then depend on b and can be calculated by replacing the distributions X with $\min(b, X)$ in our formalism. However, a fully developed solution is left for future work.

8 Conclusion

In this work, we propose two methods for quantifying the completion probabilities of LO-criticality jobs that are dropped in priority as a result of a criticality miss of HI-criticality tasks. We have proposed an analytical solution, which is applicable to harmonic task sets and criticality-monotonic priority assignment. To overcome these limitations, we presented *symbolic scheduling,* which branches on dissimilar job execution sequences while collapsing similar job sequences into a single branch. We use our analysis to show first results on success probabilities of low-criticality tasks. 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] L. Merino, F. Caballero, J. M. de Dios, J. Ferruz, and A. Ollero, "A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires," *Journal of Field Robotics*, vol. 23, no. 3–4, pp. 165–184, March–April 2006.

[3] T. M. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *International Journal of Robotics Research*, vol. 26, no. 2, pp. 141–166, February 2007.

[4] 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

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] M. Völp, M. Roitzsch, and H. Härtig, “Towards an interpretation of mixed criticality for optimistic scheduling,” in *Work-in-Progress Session of the 21st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2015.

[19] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, “Mixed-criticality real-time scheduling for multicore systems,” in *10th International Conference on Computer and Information Technology (CIT)*. IEEE, June 2010, pp. 1864–1871.

[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] 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.

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

[23] M. Völp, “What if we would degrade LO tasks in mixed-criticality systems?” in *Work-in-Progress Session of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2014.

[24] 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.

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

[26] 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.

[27] A. Z. O. Salmon, P. M. G. del Foyo, and J. R. Silva, “Scheduling real-time systems with periodic tasks using a model-checking approach,” in *12th IEEE International Conference on Industrial Informatics (INDIN)*, July 2014, pp. 73–78.

[28] G. Behrmann, A. David, K. G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks, “UPPAAL 4.0,” in *3rd International Conference on Quantitative Evaluation of Systems (QEST)*, September 2006, pp. 125–126.

[29] G. Igna, “Performance analysis of real-time task systems using timed automata,” Ph.D. dissertation, Radboud Universiteit Nijmegen, Nijmegen, Netherlands, January 2013.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.