Efficiently Approximating the Worst-Case Deadline Failure Probability under EDF

Georg von der Brüggen[1], Nico Piatkowski[2], Kuan-Hsun Chen[3],
Jian-Jia Chen[3], Katharina Morik[3], and Björn B. Brandenburg[1]
[1]Max Planck Institute for Software Systems (MPI-SWS), Germany, [2]Fraunhofer IAIS, Germany, [3]TU Dortmund University, Germany

***Abstract*—Probabilistic timing guarantees enable a tradeoff between system safety and hardware costs in embedded real-time systems. A key metric for assessing whether timing requirements can be satisfied with sufficiently high probability is the *worst-case deadline failure probability* (WCDFP). This paper studies the WCDFP under *earliest-deadline first* (EDF) scheduling for tasks with several probabilistic execution modes (e.g., a low-needs "typical" mode and a resource-intensive "exceptional" mode). Under EDF, no known approach can bound the WCDFP for practically sized workloads since the time complexity of prior approaches is exponential in the number of jobs.**

This paper examines the structure of the EDF WCDFP problem and establishes a safe, efficiently computable over-approximation by restricting the analysis to a set of specific intervals and providing a criterion to stop the derivation early without risking under-approximation. The analysis first assumes independent jobs and is then extended to handle dependencies (i.e., acyclic task chains). An evaluation shows that (i) even if 99.9999% of the jobs must meet their deadlines, a significantly higher utilization is possible than in the deterministic case, (ii) the analysis is scalable to 30 tasks with more than 10^{60} jobs in the hyperperiod, and (iii) assuming independence in the presence of dependent tasks can severely under-estimate the WCDFP.

I. Introduction

Real-time systems are classified as *hard* or *soft* based on the strictness of their timing requirements. Whereas hard real-time systems place utmost importance on timeliness—results must not just be functionally correct but also *always* delivered within the specified timing constraints—soft real-time systems are robust to occasional "timing glitches." Hence, they enable a much more favorable tradeoff between system safety and hardware costs if it is possible to quantify and bound the risk of violation of their timing constraints.

In fact, even certain safety-critical workloads do not lend themselves to a classic hard real-time approach. If a task has multiple distinct execution modes—such as a low-needs "typical" mode and an infrequent resource-intensive "exceptional" mode—both the maximum execution times and relative frequency of occurrence of those modes may differ substantially.

One well-known example are software-based fault-tolerance techniques [13], [17]–[19], [22] with temporal redundancy to (partially) re-execute a faulty job, which are usually applied when the probability that a fault occurs (and thus has to be corrected) is low. Nonetheless, faults may cluster with non-zero probability and hence re-execution of jobs may occur repeatedly in close temporal vicinity. Therefore, a hard-real-time analysis would have to make extremely pessimistic assumptions, oblivious to the diminishing probability of "freak events" (i.e., very rare, exceptional execution conditions).

In contrast, *probabilistic* timing analyses that *quantify* the probability of deadline misses can naturally exploit that extreme scenarios tend to occur only rarely. By allowing for an *a priori* assessment of whether the probability of temporal failure exceeds specified levels of acceptable residual risk (e.g., as specified by industry safety standards [1], [14]), probabilistic guarantees allow for an economical tradeoff between hardware costs and system safety: in many practically relevant systems, the probability of temporal failure does not actually have to be zero; it just has to be sufficiently small.

A central metric in the context of probabilistic timing guarantees is the *worst-case deadline failure probability* (WCDFP), which intuitively corresponds to the probability of the *first* deadline miss in any task. It allows lower-bounding the *mean time to (temporal) failure* of a system, and if jobs missing their deadlines are immediately aborted (i.e., there is no backlog of tardy jobs), it directly corresponds to the expected *deadline-miss rate*. Even if tardy jobs are *not* aborted, which can cause difficulties due to ripple effects, the WCDFP plays an important role in the derivation of the deadline-miss rate [9].

We focus on the calculation of the WCDFP under *earliest-deadline first* (EDF) scheduling. The most intuitive prior solutions are based on *job-level convolution* [2], [4], [16] and traverse a list of jobs according to their release time (for static-priority) or their finishing time (for EDF). They have an exponential time complexity with respect to the number of jobs and are thus not applicable even for a moderate number of jobs [7], [21]. Hence, transferring them to periodic or sporadic tasks under EDF is theoretically possible but not promising.

The *task-level convolution* [21] and the analytical *Chernoff bound method* [7] are generally applicable to larger task sets under static-priority scheduling. They, however, do not readily transfer to EDF due to a fundamental difference in analysis: under static-priority scheduling with constrained deadlines, only one job (and hence one period) of the task under analysis has to be evaluated, which means a pseudo-polynomial number of time points (with respect to the number of tasks) must be checked. In contrast, under EDF, it must be ensured that the schedulability condition holds for *all* time points from the moment when the system starts. As shown in this paper, if done naïvely, this leads to evaluating a number of intervals exponential in the number of tasks. Hence, the number of intervals that must be evaluated must be culled substantially.

Contributions. We study the WCDFP of constrained-deadline tasks under preemptive EDF scheduling in a unipro-

cessor environment and make the following contributions.

- We explain how the WCDFP of all tasks can be over-approximated assuming that each task has a set of distinct execution modes and that each task instance's mode is indicated by an independent random variable. We first focus on periodic tasks with synchronous releases, and then extend our analysis to sporadic tasks and periodic tasks with arbitrary phases (Sections IV to VI).
- In particular, we show how the number of considered intervals can be substantially reduced, exploiting a condition based on the probability that the processor idles. This condition allows stopping the calculation without under-approximating the WCDFP, thereby rendering the analysis feasible for task sets of nontrivial size (Section VI-C).
- We relax the independence assumption, allowing the execution mode of jobs of some tasks to be dependent on the execution mode of their predecessor(s) in a set of acyclic task chains, and detail how the WCDFP can still be upper-bounded in this scenario (Section VII).
- We evaluated randomly generated workloads and found that a sizable utilization gain is possible, even if only a very low WCDFP is acceptable. In most settings, we observed a gain of at least 12% system utilization for an acceptable WCDFP of 10^{-6} (i.e., at least 99.9999% of all jobs meet their deadline). Importantly, the analysis runtimes remained practical on consumer hardware (ranging from ≈0:26 to ≈5:36 hours for sets of 30 tasks).
- We further found that assuming independence when execution modes are actually dependent poses a significant risk of WCDFP under-estimation (Section VIII).

Section II introduces the system model and prior results on EDF. The precise problem statement is given in Section III. A short summary of related work can be found in Section IX.

II. Model, Notation, and Preliminaries

We consider a set of n independent periodic or sporadic tasks $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ scheduled on a uniprocessor. Each task $\tau_i = (\vec{C_i}, D_i, T_i, \psi_i)$ releases an unbounded number of instances called jobs. We let $J_{i,j}$ denote the j^{th} job of task τ_i.

Each task τ_i has h execution modes $\vec{C_i} = \langle c_{i,1}, \ldots, c_{i,h} \rangle$, where each mode is a tuple $c_{i,\ell} = (C_{i,\ell}, \mathbb{P}_i(\ell))$ consisting of a mode-dependent *worst-case execution time* (WCET) $C_{i,\ell}$ and a *mode probability* $\mathbb{P}_i(\ell)$ (for $\ell \in \{1, \ldots, h\}$). Each job executes exactly one of the modes: $\sum_{\ell=1}^{h} \mathbb{P}_i(\ell) = 1$. Note that the term "WCET" is used w.r.t. individual modes and not the maximum execution time in any mode. We assume a uniform number of modes h for notational convenience. Dummy modes with probability 0 can be created to fulfill this assumption.

The arrival times of all jobs are determined by an *arrival sequence* α, which is a function that maps each job $J_{i,j}$ to its *release time* (or *arrival time*) $a_{i,j} = \alpha(J_{i,j})$. An arrival sequence is *legal* if the jobs of every periodic (resp., sporadic) task τ_i are released exactly (resp., at least) T_i time units apart, where T_i is the *period* (resp., *inter-arrival time*) of the task. The first release of a periodic task is given by its phase ψ_i. (The first release of a sporadic task is unconstrained.) The hyperperiod H of Γ is the least common multiple (LCM) of the periods (or inter-arrival times) of all tasks in Γ.

A job $J_{i,j}$ of τ_i released at time $a_{i,j}$ and executed in the ℓ^{th} execution mode must receive up to $C_{i,\ell}$ units of processor service before its *absolute deadline* $d_{i,j} = a_{i,j} + D_i$. We assume constrained deadlines (i.e., $D_i \leq T_i$ for each task τ_i). The response time of $J_{i,j}$ is $R_{i,j} = f_{i,j} - a_{i,j}$, where $f_{i,j}$ is its *finishing time*. A task set is *schedulable* under a given scheduling policy if all jobs of all tasks meet their deadline (i.e., $R_{i,j} \leq D_i$ for each job $J_{i,j}$) in any legal arrival sequence.

We assume dense time. A list of notation is provided in Table I. In a slight abuse of notation, given a vector $\vec{X}$, we write $x \in \vec{X}$ to indicate that x is a component of $\vec{X}$. Furthermore, we denote the *indicator function* as $[\![\text{expression}]\!]_{\mathbb{1}}$, which is 1 if and only if the expression is true and 0 otherwise.

Probabilistic independence. We assume that execution mode probabilities are independent. In particular, the execution mode probability of a newly arriving job is independent of the execution mode of any other job. Hence, the execution mode of $J_{i,j}$ is determined solely by a random variable $\boldsymbol{X}_{i,j}$ that assumes value $\ell \in \{1, ..., h\}$ with probability $\mathbb{P}_i(\ell)$. We denote the WCET of $J_{i,j}$ as $C_i(\boldsymbol{X}_{i,j})$. This assumption is relaxed in Section VII, where we allow acyclic dependencies (i.e., task chains) that affect a bounded number of subsequent jobs.

Scheduling policy. We consider the preemptive *earliest-deadline first* (EDF) scheduling policy: at each point in time, the incomplete job (if any) with the earliest absolute deadline is scheduled. Hence, the processor idles only when no job is ready to be executed (i.e., EDF is work-conserving). Under EDF, ties in absolute deadline are broken arbitrarily but consistently. Hence, the analysis of a specific job assumes that all jobs with the same deadline have higher priority. The set of jobs that have to be finished before a job $J_{i,j}$ (i.e., the possibly interfering jobs with higher priority) is given by $HP(J_{i,j}) = \{J_{x,y} \mid \tau_x \in \Gamma \wedge d_{x,y} \leq d_{i,j}\} \setminus J_{i,j}$. A central concept in the analysis of EDF is the notion of a busy interval.

Definition 1: *A time t is a **quiet time** if all jobs that arrived before t are completed by time t. An interval $[t_1, t_2)$ is a **busy interval** if t_1 and t_2 are both quiet times, some job is released at time t_1, and no time $t \in (t_1, t_2)$ is a quiet time.*

Only jobs in the same busy interval interfere with each other since EDF is work-conserving. Hence, we reset each task's job count at the start of a busy interval for notational convenience.

Deterministic schedulability. In the classical, deterministic setting [3], where $C_i = \max_\ell \{C_{i,\ell}\}$ is the overall WCET of task τ_i, schedulability under EDF can be decided by analyzing the cumulative processor demand of the task set. The demand of τ_i in an interval $[t_1, t_2]$ (i.e., the maximum total processing time that jobs of τ_i require in $[t_1, t_2]$) is given by the *demand bound function* $dbf_i(t_1, t_2) = N_i(t_1, t_2) \cdot C_i$ [3], where

$$N_i(t_1, t_2) = \max\left\{0, \left\lfloor \frac{t_2 + T_i - D_i - \psi_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \psi_i}{T_i} \right\rceil \right\} \quad (1)$$

is the maximum number of jobs in $[t_1, t_2]$. For a given interval $[t_1, t_2]$, Eq. (1) considers only jobs with arrival time $a_{i,j} \geq t_1$ and absolute deadline $d_{i,j} \leq t_2$. For the special case of

TABLE I. SUMMARY OF NOTATION

$\tau_i = (\vec{C}_i, D_i, T_i, \psi_i)$	Task τ_i with WCET distribution $\vec{C}_i$, deadline D_i, period T_i, and phase ψ_i	Sec. II
$c_{i,j} = (C_{i,j}, \mathbb{P}_i(j)) \in \vec{C}_i$	WCET $C_{i,j}$ of the j^{th} execution mode with related probability $\mathbb{P}_i(j)$	Sec. II
$N_i(t_1, t_2)$ and $N_i(I)$	Maximum number of jobs of τ_i released in an interval $[t_1, t_2]$ (an interval I)	Sec. II
$S_{[t_1,t_2]}$ and S_I	Accumulated probabilistic demand released in an interval $[t_1, t_2]$ (an interval I)	Sec. IV
$\Phi_{k,j}$	Upper bounded probability that job $J_{k,j}$ of τ_k is the first to miss the deadline after idle time	Sec. III
Φ_k and Φ_Γ	Upper bounded probability for first deadline miss for τ_k and Γ after idle time	Sec. III
$\mathbb{P}(S_I > \lvert I \rvert)$	Probability of overload for the interval I with length $\lvert I \rvert$	Sec. IV
$\mathbb{P}_{busy}(I)$	Probability that the processor does not idle in interval I	Sec. VI
$\vec{V}$	Vector of possible probabilistic demands	Sec. IV
$v_i = (v_i^d, v_i^p) \in \vec{V}$	Possible demand value in $\vec{V}$ with demand v_i^d and probability v_i^p	Sec. IV
$\boldsymbol{X}(I)$	Random variable representing the possible execution modes of all jobs in interval I	Sec. IV
$\mathcal{X}(I)$	The state space of $\boldsymbol{X}(I)$	Sec. IV
$\vec{\boldsymbol{x}} \in \mathcal{X}(I)$	A possible variable assignment for $\boldsymbol{X}(I)$	Sec. IV
$\mathbb{P}(\boldsymbol{X}(I) = \vec{\boldsymbol{x}})$	Probability that $\boldsymbol{X}(I)$ has the specific variable assignment $\vec{\boldsymbol{x}}$	Sec. IV
$\boldsymbol{X}_{i,j}(I)$	The random variable related to the j^{th} job of τ_i in I	Sec. IV
$C_i(\boldsymbol{X}_{i,j}(I))$	WCET for the j^{th} job of τ_i in I based on its random execution mode $\boldsymbol{X}_{i,j}(I)$	Sec. IV
$\vec{\boldsymbol{x}}_{i,j}$	value of $\boldsymbol{X}_{i,j}(I)$ in assignment $\vec{\boldsymbol{x}}$	Sec. IV

$\psi_i = 0$ this reduces to $dbf_i(t) = \max\left\{0, \left\lfloor \frac{t-D_i}{T_i} \right\rfloor + 1\right\} \cdot C_i$ for the interval $[0, t]$. Let L be the first quiet time after all tasks are synchronously released at time 0, which is the longest possible busy interval. Baruah et al. [3] proved that a task set Γ is schedulable (in the deterministic sense) under preemptive EDF *if and only if*: **(i)** the processor is not over-utilized (i.e., $\sum_i^n \frac{C_i}{T_i} \leq 1$), and **(ii)** for any time interval of length t, the total processor demand $dbf(t)$ does not exceed t, which means that

$$dbf(t) = \sum_{i=1}^{n} dbf_i(t) \leq t \qquad \forall t : 0 < t \leq L. \tag{2}$$

It is sufficient to examine Eq. (2) for each absolute deadline of a job, since the demand bound function changes only at these time points, which we call *points of interest*. Given a point of interest t, the corresponding interval $[0, t]$ is called the *interval of interest*.

III. PROBLEM DEFINITION

Ultimately, we seek to bound the probability that any task in the system is the first to miss a deadline after the system resets (i.e., after the last quiet time), for *any* legal arrival sequence. To this end, we first focus on an arbitrary but fixed job $J_{k,\ell}$ of a task τ_k in the context of a fixed arrival sequence α.

Definition 2: *The* ***deadline failure probability*** *(DFP) of $J_{k,\ell}$ for a given arrival sequence α, denoted $\Phi_{k,\ell}$, is the probability that $J_{k,\ell}$ is the first job to miss its deadline in its busy interval if jobs arrive according to α, and is given by*

$$\Phi_{k,\ell} = \mathbb{P}(R_{k,\ell} > D_k \text{ and } \quad \forall J_{i,j} \in HP(J_{k,\ell}) : R_{i,j} \leq D_i). \tag{3}$$

In other words, $\Phi_{k,\ell}$ is the probability that $J_{k,\ell}$ misses its deadline while all jobs in $HP(J_{k,\ell})$ meet their respective deadlines. Based on Definition 2, the **task DFP** Φ_k of τ_k (w.r.t. α) is simply $\max_\ell\{\Phi_{k,\ell}\}$, and the **system DFP** Φ_Γ (w.r.t. α) is $\max_{1\leq k\leq n}\{\Phi_k\}$. Finally, we denote as the **worst-case DFP** (WCDFP) of a task (resp., system) the largest-possible task (resp., system) DFP. In other words, the WCDFP bounds the task (system) DFP *for any legal arrival sequence*.

To safely determine the WCDFP, since we cannot simply calculate the DFP for all jobs and take the maximum across all possible arrival sequences (i.e., apply the definition), we instead must bound the DFP by considering a specific worst-case situation, using a concept similar to a critical instant.

Recall that, under preemptive EDF, a job $J_{i,j}$ misses its deadline when the overall demand of $J_{i,j}$ and higher-priority jobs in an interval $[t_s, d_{i,j}]$ with $t_s \leq a_{i,j}$ exceeds the interval length, which is called a (demand) overload w.r.t. the interval. In the deterministic setting, whether an interval can be overloaded is a boolean proposition and it suffices to check Inequality (2) for all intervals of interest. If none can be overloaded, the task set is schedulable [3]. If however an interval that can be overloaded is found, then the analysis is stopped immediately as the task set is certainly not schedulable.

Additional challenges arise in the probabilistic setting: **(i)** Whether an interval is overloaded is no longer a simple boolean property. Rather, we must determine the probability that a given interval becomes overloaded. **(ii)** In the deterministic case, the analysis continues until either an overloaded interval is found or the busy interval ends. In contrast, probabilistic analysis is typically applied in situations where the system is schedulable in the case that all jobs execute in their mode with the least WCET, which implies that there is no interval with overload probability 100%. At the same time, the system utilization usually exceeds 100% when all jobs execute in their mode with the largest WCET, which implies that there is no longest busy interval. Finding an appropriate analysis termination condition is thus nontrivial. **(iii)** The high computational complexity of calculating the overload probability in a given interval renders an exhaustive search of the busy interval prohibitively expensive. **(iv)** A worst-case scenario that maximizes the DFP for a given job must be established for the probabilistic setting to avoid having to enumerate all possible arrival sequences.

To address (i), Sections IV and V examine how to calculate the overload probability of a given interval. To tackle (ii)–(iv),

Section VI explores what intervals have to be considered and how the number of intervals can be reduced without risking under-approximation. In both steps, we start with periodic tasks and extend to sporadic tasks subsequently.

IV. Overload Probability of a Given Interval

We start by examining the probability that one specific job $J_{k,\ell}$ misses its deadline $d_{k,\ell}$ for a fixed arrival sequence α under the simplifying assumption that each job executes for its entire mode-dependent WCET, which is safe since preemptive EDF is timing-anomaly free (i.e., a job requiring less service does not increase the likelihood of a deadline miss).

The work that must be completed in $I = [t_1, t_2]$ is the demand (denoted as $S_{[t_1,t_2]}$ or S_I) of jobs both (i) released after t_1 and (ii) with an absolute deadline no later than t_2. Hence, a deadline miss occurs at $t_2 = d_{k,\ell}$ only if there is an interval $I = [t_1, t_2]$ where $J_{k,\ell}$ and jobs in $HP(J_{k,\ell})$ exhibit cumulative demand exceeding the interval length $|I| = t_2 - t_1$. We refer to the situation that the demand exceeds the interval length as *overload*. Accordingly $\mathbb{P}(S_{[t_1,t_2]} > t_2 - t_1) = \mathbb{P}(S_I > |I|)$ is the *overload probability* of the interval $I = [t_1, t_2]$.

Analogously to the probabilistic WCET $C_i(\boldsymbol{X}_{i,j})$ of a job $J_{i,j}$, the cumulative **probabilistic demand** in an interval I is a random variable whose probability mass is given by a vector $\vec{V} = \langle v_1, v_2, \ldots, \rangle$, where each component $v_x = (v_x^d, v_x^p) \in \vec{V}$ is a pair of a possible cumulative demand $v_x^d = S_I$ with probability v_x^p, such that $\sum_x v_x^p = 1$.

Each job $J_{i,j}$ executing in I has a WCET $C_i(\boldsymbol{X}_{i,j}(I))$ dependent on a random variable $\boldsymbol{X}_{i,j}(I)$ that determines its execution mode. Let $\boldsymbol{X}(I)$ be the set of all these random variables, let $\mathcal{X}(I)$ be all possible variable assignments of $\boldsymbol{X}(I)$, let $\vec{\boldsymbol{x}} \in \mathcal{X}(I)$ be one specific assignment, and let $\vec{\boldsymbol{x}}_{i,j}$ denote the value of $\boldsymbol{X}_{i,j}(I)$ in assignment $\vec{\boldsymbol{x}}$. Based on the number of jobs $N_i(I)$ of each task in I, the probabilistic demand is simply the sum of the probabilistic WCETs:

$$S_I(\vec{x}) = \sum_{\tau_i \in \Gamma} \sum_{j=1}^{N_i(I)} C_i(\vec{\boldsymbol{x}}_{i,j}), \tag{4}$$

where $\sum_{j=1}^{0} C_i(\vec{\boldsymbol{x}}_{i,j}) \mathrel{\widehat{=}} 0$ for notational convenience. Hence, considering all possible assignments $\vec{\boldsymbol{x}} \in \mathcal{X}(I)$, the overload probability of I and thus the probability that $J_{k,\ell}$ misses its deadline at time t_2 due to an overload in I is:

$$\mathbb{P}(S_I(\boldsymbol{X}(I)) > |I|) = \sum_{\vec{\boldsymbol{x}} \in \mathcal{X}(I)} \mathbb{P}(\boldsymbol{X}(I) = \vec{\boldsymbol{x}}) \cdot [\![S_I(\vec{\boldsymbol{x}}) > |I|]\!]_{\mathbb{1}}. \tag{5}$$

Since the execution modes of the jobs are assumed to be independent, the joint probability mass $\mathbb{P}(\boldsymbol{X}(I))$ is the product of the individual mode probabilities of all jobs. For each job, the probability distribution is identical to the probability distribution of its task, and therefore the probability that a specific assignment $\vec{\boldsymbol{x}}$ occurs is

$$\mathbb{P}(\boldsymbol{X}(I) = \vec{\boldsymbol{x}}) = \prod_{\tau_i \in \Gamma} \prod_{j=1}^{N_i(I)} \mathbb{P}_i(\vec{\boldsymbol{x}}_{i,j}), \tag{6}$$

where $\prod_{j=1}^{0} \mathbb{P}_i(\vec{\boldsymbol{x}}_{i,j}) \mathrel{\widehat{=}} 1$ for notational convenience.

Note that Eq. (6) implicitly depends on the arrival times of the jobs in I and hence must be interpreted in the context of a given arrival sequence α. In the case of periodic tasks, the number of releases $N_i(I)$ in I can be calculated with Eq. (1). For the case of sporadic tasks, we must find an approximation.

Our analysis exploits that "adding" jobs to an interval under consideration I does not risk under-approximation. To state this precisely, we formalize a notion of "increased demand."

Definition 3: *A probabilistic demand $\vec{V}$ is **increased** relative to a probabilistic demand $\vec{U}$ if*

$$\forall r \geq 0 : \quad \sum_{v_x \in \vec{V}} v_x^p \cdot [\![v_x^d > r]\!]_{\mathbb{1}} \geq \sum_{u_x \in \vec{U}} u_x^p \cdot [\![u_x^d > r]\!]_{\mathbb{1}}. \tag{7}$$

In other words, the tail distribution of $\vec{V}$ must exceed that of $\vec{U}$ for any demand threshold r (to be clear, irrespective of whether $\vec{V}$ and $\vec{U}$ reflect the same or different arrival sequences or intervals). Any change that increases the probabilistic demand in an interval I thus ensures monotonicity of the probability of an overload in I, independently of the length of the interval $|I|$ or the number of jobs executing in it.

Lemma 1: *Consider any interval I and two arrival sequences α and α', and let $\vec{V}$ and $\vec{U}$ denote the probabilistic demand in I under α' and α, respectively. If no task releases fewer jobs during I in α' than in α, then $\vec{V}$ is increased w.r.t. $\vec{U}$.*

Proof: First, suppose α and α' differ in only a single job $J_{a,b}$ that α' releases during I in addition to all the jobs also released by α during I, and assume $d_{a,b} \leq t_2$. (If no such job exists, then the set of jobs contributing demand during I is identical under α' and α and the claim follows trivially.) Due to **(i)** the assumed independence of execution modes and the facts that **(ii)** no task releases fewer jobs during I in α' than in α and **(iii)** the cumulative demand $S_{[t_1,t_2]}$ is a simple sum of execution costs and hence permutation-invariant, each $u_x \in \vec{U}$ maps to h corresponding elements $v_x^\ell \in \vec{V}$ (for $\ell \in \{1, \ldots, h\}$), where each $v_x^\ell = (v_x^{d,\ell}, v_x^{p,\ell})$ is given by $v_x^{d,\ell} = u_x^d + C_{a,\ell}$ and $v_x^{p,\ell} = u_x^{p,\ell} \cdot \mathbb{P}_a(\ell)$. Thus, for any demand threshold r and any $u_x \in \vec{U}$, if $u_x^d > r$, then also $v_x^{d,\ell} > r$ for every ℓ. Furthermore, since task τ_a's execution mode probabilities sum to 1, we have $\sum_{\ell=1}^{h} v_x^{p,\ell} = \sum_{\ell=1}^{h} u_x^p \cdot \mathbb{P}_a(\ell) = u_x^p \cdot \sum_{\ell=1}^{h} \mathbb{P}_a(\ell) = u_x^p$. Inequality (7) thus holds. The complete proof follows by repeating the above argument for each added job. ■

V. Calculation of the Overload Probability

Until now, we have analyzed the overload probability of a given interval I in general terms, but did not yet explain how to calculate it in practice. To this end, it is necessary to bound the probabilistic demand in I. While no approach specific to EDF has been proposed to date, the same problem — approximating the cumulative probabilistic demand of a set of jobs executing in a given interval — arises also in the probabilistic analysis of static-priority scheduling, for which relevant prior work exists. In the following, we discuss how to transfer two applicable techniques to the EDF setting.

In particular, two approaches based on the *Chernoff bound* [7] and *task-level convolution* [21] are known to scale

best. At a high level, in the case of static-priority scheduling, both methods iterate over a certain set of intervals, calculate the related overload probabilities, and then report the minimum among those values as the WCDFP. Exactly which intervals must be considered and exactly how the calculated probability values are further processed differs a lot in the EDF and static-priority settings; we discuss these issues in the next section. The actual calculation of the overload probability for a given interval, however, is independent of the underlying scheduling policy and specific arrival patterns. In fact, the only relevant parameters are the length of the interval and the number of jobs that have to be considered for each task. It is hence relatively straightforward to adjust these approaches for our purposes. In the interest of completeness, we summarize both approaches, reformulating lemmas and formulas to match setting and notation, and sketch the necessary changes to the static-priority setting considered in the original papers.

Chernoff bound. The approach by Chen and Chen [7] upper-bounds the overload probability by means of the *moment generating function* (*mgf*) and the *Chernoff bound*. Recall that the *mgf* specifies the probability distribution of a random variable, here the WCET of a job of τ_i, as $\text{mgf}_i(s) = \sum_{j=1}^{h} \exp(C_{i,j} \cdot s) \cdot \mathbb{P}_i(j)$, where *exp* is the exponential function (i.e., $\exp(x) = e^x$) and $s > 0$ is a given real number. Combined with the Chernoff bound, this characterization allows bounding the overload probability as follows.

Lemma 2 (based on Lemma 1 in [7]): *If S_I is the sum of the execution times of the jobs in Γ that are released in and must be finished in I, then*

$$\mathbb{P}(S_I \geq |I|) \leq \min_{s>0} \left(\frac{\prod_{\tau_i \in \Gamma} (\text{mgf}_i(s))^{N_i(I)}}{\exp(s \cdot |I|)} \right). \quad (8)$$

Lemma 2 transfers the considered interval from $[0, t)$ (which is sufficient in the static-priority case) to any I. Under static-priority scheduling, when analyzing task τ_k, only tasks with higher or equal priority than τ_k must be considered. Under EDF, each $\tau_i \in \Gamma$ can contribute to the demand in I. The number of jobs $N_i(I)$ of τ_i contributing demand to I is calculated considering all jobs that are released in and must be finished in I, which can be bounded using Eq. (1). Note that Eq. (8) is valid for any s as the Chernoff bound always provides an over-approximation. Chen et al. [8] discuss how to efficiently find an s with good precision. However, even for the s that minimizes the right-hand side of Eq. (8), the approximation error can become arbitrarily large.

Task-level convolution. The main idea behind the task-level convolution approach by von der Brüggen et al. [21] is that, if probabilities are independent, then the demand of a task τ_i in a specific interval I depends only on the number of jobs executing in each mode (but not on their order). Hence, if $N_i(I)$ jobs are considered in I, the set of all possible scenarios can be partitioned into equivalence classes depending on the number of jobs that execute in each mode. Each equivalence class satisfies $\sum_{j=1}^{h} \ell_{i,j} = N_i(I)$, where $\ell_{i,j} \in \mathbb{N}_0$ denotes the number of jobs of τ_i in I in the j^{th} execution mode. This can be seen as drawing $N_i(I)$ times from an urn with h balls (with replacement). Hence, basic combinatorial reasoning leads to a characterization based on multinomial distributions.

The demand in each scenario in such an equivalence class is given by $\sum_{j=1}^{h} \ell_{i,j} \cdot C_{i,j}$, and the probability that any of the scenarios in the equivalence class occurs can be calculated as

$$\frac{N_i(I)!}{\ell_{i,1}!\ell_{i,2}!...\ell_{i,h}!} \mathbb{P}_i(1)^{\ell_{i,1}} \cdot \mathbb{P}_i(2)^{\ell_{i,2}} \cdot ... \cdot \mathbb{P}_i(h)^{\ell_{i,h}}, \quad (9)$$

where $\frac{N_i(I)!}{\ell_{i,1}!\ell_{i,2}!...\ell_{i,h}!}$ is the number of scenarios in the equivalence class, which all lead to the same demand (and hence the same overload probability). For a task τ_i with $N_i(I)$ jobs and h modes, $\binom{N_i(I)+h-1}{h-1}$ such equivalence classes exist, where $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ is the binomial coefficient. This approach can be extended to consider equivalence classes for all tasks simultaneously; that is, each class is canonically represented by a tuple $\ell \in \otimes_{\tau_i \in \Gamma} \{1, 2, \ldots, N_i(I)\}^h$, indicating for all tasks the number of jobs in each execution mode.

Lemma 3 (based on Lemma 11 in [21]): *Let $\mathcal{L}$ be the set of all possible equivalence classes: $\mathcal{L} \subseteq \otimes_{\tau_i \in \Gamma} \{0, 1, 2, \ldots, N_i(I)\}^h$ such that, for each $\ell \in \mathcal{L}$ and each $\tau_i \in \Gamma$, it holds that $\sum_{j=1}^{h} \ell_{i,j} = N_i(I)$, where $\ell_{i,j}$ is the number of jobs of τ_i in the j-th execution mode in class ℓ. Then, $\mathbb{P}(S_I \geq |I|) =$*

$$\sum_{\ell \in \mathcal{L}} \prod_{\tau_i \in \Gamma} \frac{N_i(I)! \cdot \prod_{j=1}^{h} \mathbb{P}_i(j)^{\ell_{i,j}}}{\prod_{x=1}^{h} \ell_{i,x}!} \cdot [\![\mathrm{S_I}(\ell) \geq |\mathrm{I}|]\!]_{\mathbb{1}}, \quad (10)$$

where $S_I(\ell)$ is the demand in each scenario in class ℓ.

The main differences to Lemma 11 in [21] are that all tasks in Γ can contribute to the demand and that $N_i(I)$ is the number of jobs that are released in and must finish in I.

Eq. (10) is computed as follows [21]. For each task τ_i, the equivalence classes with w.r.t. probability and demand of $N_i(I)$ jobs of τ_i are calculated. These $\binom{N_i(I)+h-1}{h-1}$ pairs of probability and demand are represented as a vector (e.g., $\left\langle \begin{smallmatrix} 6 & 9 \\ 0.7 & 0.3 \end{smallmatrix} \right\rangle$ would indicate a possible demand of 6 or 9 with related probabilities 0.7 and 0.3, respectively). Initially, there are n such vectors, one for each task, which are then convolved in a pairwise fashion until only one vector remains [21]. After all vectors have been convolved, the probabilities of all elements of the final vector with demand exceeding $|I|$ are summed up to determine $\mathbb{P}(S_I \geq |I|)$.

To increase runtime efficiency, von der Brüggen et al. [21] introduced two optimization techniques, which we both applied in the evaluation. *State pruning* keeps track of the minimum and maximum demand the remaining tasks (i.e., tasks whose demand vector has not been convolved yet) can jointly contribute. Consider the following example vector: $\left\langle \begin{smallmatrix} 12 & 13 & 14 & 15 \\ 0.72 & 0.18 & 0.09 & 0.01 \end{smallmatrix} \right\rangle$. Let these minimum and maximum values be 5 and 7, respectively, and the interval that is considered have length $|I| = 19$. Now, the vector entry with demand 12 can be discarded, since it cannot result in the total demand exceeding 19 even when adding the maximum remaining demand of 7. Conversely, the entry with demand 15 can be discarded from further consideration as well, since even the addition of the the minimum remaining demand will result in demand exceeding 19. In the latter case, the probability of

0.01 has to be added to the overload probability.

The second runtime optimization is *state merging*, which reduces the vector's size at the cost of some added pessimism. For instance, to reduce the above example vector to length two, the entry related to demand 12 would be kept and the remaining three entries would be merged by summing up their probabilities (i.e., 0.28) and taking the maximum demand (i.e., 15), leading to the shortened vector $\left\langle \begin{smallmatrix} 12 & 15 \\ 0.72 & 0.21 \end{smallmatrix} \right\rangle$. We refer the interested reader to [21] for a more detailed discussion.

VI. Bounding the WCDFP

Equipped with a way to calculate the overload probability of a given interval, we now focus on bounding the WCDFP of a given task. We first consider synchronous periodic releases, examine how an exact calculation would work, and discuss inherent complexity issues. Thereafter, we provide a more scalable over-approximation that also extends to sporadic tasks.

A. Synchronous Periodic Tasks

We first bound the number of intervals of interest.

Lemma 4: *For each task in a set of synchronously released periodic constrained-deadline tasks, a job that exhibits the task DFP exists within the first hyperperiod.*

Proof: By contradiction. Suppose a job $J_{i,k}$ with $a_{i,k} \geq H \cdot m$ and $m \in \mathbb{N}$ exists (i.e., $J_{i,k}$ is released in the $m+1^{th}$ hyperperiod) that has a DFP $\Phi_{i,k} > \Phi_{i,j}$ for a all jobs $J_{i,j}$ with $a_{i,j} < H$. Since by definition of DFP we analyze the probability that a job is the *first* to miss the deadline in its busy interval, for each job $J_{i,k}$, only scenarios in which no previous job in the same busy interval misses its respective deadline contribute to $J_{i,k}$'s DFP $\Phi_{i,k}$. Hence, to determine $\Phi_{i,k}$, we only have to consider busy intervals wherein all higher-priority jobs finish before their respective deadlines. As a result, only jobs released after time $H \cdot m$ contribute to $\Phi_{i,k}$, since any job J released before $H \cdot m$ would have to execute after $H \cdot m$ to contribute, which contradicts the assumption that J meets its deadline. For periodic tasks, the release pattern in the interval $[0, H)$ is identical to the release pattern in the interval $[m{\cdot}H, (m{+}1){\cdot}H)$. Since the probabilistic distributions are independent and identical, $\Phi_{i,k}$ of $J_{i,k}$ released at $a_{i,k}$ is identical to $\Phi_{i,k-(m \cdot H/T_i)}$ of $J_{i,k-(m \cdot H/T_i)}$ released at $a_{i,k-(m \cdot H/T_i)} < H$, which contradicts the assumption that $\Phi_{i,k} > \Phi_{i,j}$ for a job $J_{i,j}$ with $a_{i,j} < H$. ■

Lemma 4 allows us to upper-bound the DFP $\Phi_{i,j}$ for any job $J_{i,j}$, and hence the task DFP of τ_i, by considering only jobs in the first hyperperiod.

Lemma 5: *Let $J_{i,j}$ be released in $[0, H)$. Let $A(0, d_{i,j} - D_i)$ be the set of arrival times of all jobs in $HP(J_{i,j})$ and of $J_{i,j}$ itself. The failure probability of $J_{i,j}$ is upper-bounded by*

$$\Phi^*_{i,j} = \sum_{t_s \in A(0, d_{i,j} - D_i)} \mathbb{P}(S_{[t_s, d_{i,j}]} > d_{i,j} - t_s). \qquad (11)$$

Proof: For $J_{i,j}$ to miss its deadline, an overloaded busy interval $[t_s, d_{i,j})$ with $t_s \in A(0, d_{i,j} - D_j)$ must exist, and $J_{i,j}$ may miss its deadline due to an overload in any of these intervals. Hence, the sum of the overload probabilities of all such intervals upper-bounds the probability of $J_{i,j}$ being the first job to miss its deadline. ■

The number of intervals considered in Eq. (11) can be reduced if the length of the longest busy interval is less than H. Note that Eq. (11) is an over-approximation, since it only checks whether an interval is overloaded but not whether a specific assignment of the probabilistic variables already results in an overload in any subintervals. Hence, if such assignments exist, then they contribute to $\Phi^*_{i,j}$ multiple times, leading to a safe but imprecise over-approximation.

Unfortunately, bounding the WCDFP based on Lemma 5 alone is computationally intractable for large numbers of jobs in the hyperperiod. To see why, let the number of jobs in the hyperperiod be $|H|$, let $J_1, J_2, J_3, ..., J_{|H|}$ be the jobs in H ordered according to their absolute deadline, and let $a_1, a_2, \ldots, a_{|H|}$ be their arrival times. By Lemma 5, the probability that $J_{|H|}$ misses its deadline is bounded by the sum of the probabilities calculated for all intervals. For $J_{|H|}$, we thus would have to consider the intervals starting at times $a_1, a_2, \ldots, a_{|H|}$ (i.e., $O(|H|)$ intervals in total). Performing the same procedure for $J_{|H|-1}$, $J_{|H|-2}$, etc., however, leads to considering $O(|H|^2)$ intervals overall, which is computationally intractable since the number of jobs in the hyperperiod is generally exponential in the number of tasks.

B. Over-Approximation and Generalization

Since using Lemma 5 to upper-bound the WCDFP is impractical complexity-wise, we instead provide a more scalable over-approximation, which is also applicable to sporadic tasks. We first analyze the task DFP of an individual sporadic task τ_k in any given (possibly non-periodic) arrival sequence by relating it to a periodic worst-case scenario. Thereafter, we show that the scenario holds for all tasks in general regardless of the given initial arrival sequence, which allows us to bound the WCDFP even for sporadic tasks.

More specifically, in the following, we consider the specific job $J_{k,\ell}$ with deadline $d_{k,\ell}$ that exhibits the task DFP Φ_k for τ_k in a given arrival sequence α. For simplicity, we discard from consideration any job $\tau_{i,j}$ with a deadline $d_{i,j} > d_{k,\ell}$, since such jobs do not impact $\Phi_{k,\ell}$ and hence also not Φ_k.

Our analysis proceeds by a sequence of step-wise "transformations" of the considered scenario. We say an arrival sequence α is *transformed* into another arrival sequence α' if α and α' differ only regarding the arrival time of one job (i.e., a transformation changes the arrival time $a_{i,j}$ of some job $\tau_{i,j}$ to $a'_{i,j}$). Let I be the interval $[a_{i,j}, d_{k,\ell})$ before and let I' be the interval $[a'_{i,j}, d_{k,\ell})$ after transformation. A transformation is *safe* (with respect to Φ_k) if it does not decrease the DFP of $J_{k,\ell}$ and hence does not decrease the DFP of τ_k either.

We require some additional notation to reason about the safety of the employed transformations. Let $\boldsymbol{X}(\alpha)$ be the set of random variables for all jobs in α, let $\widehat{\mathcal{X}(\alpha)}$ be all assignments of $\boldsymbol{X}(\alpha)$ that result in a deadline failure for $J_{k,\ell}$, and let $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$ be one of these assignments. Let $A(0, d_{k,\ell} - D_k)$ be the release times of jobs in $[0, d_{k,\ell} - D_k]$ under the arrival sequence α (i.e., the possible start times of busy intervals).

Lemma 6: *The transformation $a'_{i,j} = a_{i,j} + \delta$ for $\delta > 0$ is safe if $d_{i,j} + \delta \leq d_{k,\ell}$ (i.e, any job $J_{i,j}$ can be released later as long as its deadline is not pushed past $J_{k,\ell}$'s deadline).*

Proof: We must show for all $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$ that there still exists an interval under α' where $\vec{x}$ leads to a deadline failure.

Consider any $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$ and let $I = [t, d_{k,\ell}]$ be an interval that is overloaded for $\vec{x}$. I is either: **(i)** the interval starting with the release of $J_{i,j}$, or **(ii)** an interval starting with the release of a different job. In case (ii), $I = I'$ and the number of jobs that must be serviced in I' is either increased due to $J_{i,j}$ if $a_{i,j} < t$ and $a'_{i,j} \geq t$, or otherwise remains the same. Hence, the demand in I under α' is not less than under α.

In case (i), $|I'| = |I| - \delta$ since $a_{i,j} = t$. If no job is released in $[t, t + \delta)$, an overload of $\vec{x}$ in I implies an overload in I' since the jobs in these intervals are the same and $|I'| < |I|$.

Otherwise, let t^* be the earliest release in $[t, t + \delta)$ in α, let $I^* = [t^*, d_{k,\ell})$, and let $J_{p,q}$ be a job released at time t^*. If $t^* = t$, then $I^* = I$ and $\vec{x}$ results in an overload in I^*. If $t^* > t$, the demand of $\vec{x}$ in I' is less than in I, since the cost of $J_{p,q}$ no longer counts as demand in I'. However, since I^* is not changed by the transformation, $J_{i,j}$ is added to I^*, thus the demand in I^* after the transformation is no less than the demand in I before the transformation. Finally, since $|I^*| < |I|$ (since $t^* > t$) and because $\vec{x}$ results in an overload of I before the transformation, we know that $\vec{x}$ results in an overload of I^* after the transformation. ■

We call a task τ_i *aligned* with $J_{k,\ell}$ in a *prefix interval* $I = [t, d_{k,\ell}]$ if there exists a job $J_{i,j}$ with the same deadline as $J_{k,\ell}$, and τ_i is released periodically prior to $J_{i,j}$ in I.

Lemma 7: *Each non-alined task τ_i can be aligned with $J_{k,\ell}$ in I by means of a sequence of safe transformations.*

Proof: Let $J_{i,j}$ have the largest absolute deadline among jobs of τ_i with deadline less than $d_{k,\ell}$, and consider the jobs $J_{i,1}, J_{i,2}, \ldots, J_{i,j-1}, J_{i,j}$ in order of decreasing deadlines. First, we set $a'_{i,j} = d_{k,\ell} - D_i > a_{i,j}$, which is safe according to Lemma 6. Next, we set $a'_{i,j-1} = d_{k,\ell} - D_i - T_i > a_{i,j-1}$, which is similarly safe and ensures a legal periodic inter-arrival time separation of $J_{i,j-1}$ and $J_{i,j}$. Repeating the process, we set $a'_{i,j-2} = d_{k,\ell} - D_i - 2T_i > a_{i,j-2}$, etc., until τ_i is aligned with $J_{k,\ell}$ in I under the resulting arrival sequence.[1] ■

Based on Lemma 7, for a given I, all tasks can be aligned in a series of safe transformations. These transformations are required only once (i.e., for the longest interval I that results in an overload for $J_{k,\ell}$ for any $\vec{v}$), since the intervals overlap and tasks are aligned to the same time $d_{k,\ell}$ in all intervals, which we note with the following lemma.

Lemma 8: *If all tasks are aligned with $J_{k,\ell}$ in I, they are aligned w.r.t. any subinterval $I' = [t', d_{k,\ell}] \subseteq I$ as well.*

Lemma 1 implies that postponing releases can only reduce the probabilistic demand, and hence we can safely over-approximate the DFP of $J_{k,\ell}$ in a prefix interval I by considering the case where all tasks are aligned to $J_{k,\ell}$.

[1]For simplicity, we assume w.l.o.g. sufficient prior jobs of τ_i to cover all of I. Otherwise, by Lemma 1 we can shift jobs released after $d_{k,\ell}$ into I.

Lemma 9: *For each prefix interval $I = [t, d_{k,\ell}]$, the overload probability is maximized when all tasks are aligned to $d_{k,\ell}$.*

Proof: For each τ_i, the number of jobs that must be serviced in I, and hence contribute demand towards a possible overload, is maximized when τ_i is aligned to $J_{k,\ell}$. The claim hence follows from Lemma 1 and the fact that postponing the release times of jobs already accounted for in a prefix interval may only reduce the number of demand-inducing jobs by moving their deadline or arrival time out of the interval. ■

In the following, let α^* be the periodic arrival sequence obtained from aligning all tasks to $J_{k,\ell}$ in I.

Lemma 10: *Let $A(0, d_{k,\ell} - D_k)$ be the set of all arrival times times in $[0, d_{k,\ell} - D_k]$ in the arrival sequence α^*. Then the DFP $\Phi_{k,\ell}$ of $J_{k,\ell}$ is bounded as follows.*

$$\Phi_{k,\ell} \leq \sum_{t_s \in A(0, d_{k,\ell} - D_k)} \mathbb{P}(S_{[t_s, d_{k,\ell}]} > d_{k,\ell} - t_s) \qquad (12)$$

Proof: Lemmas 6–9 ensure that all intervals overloaded in α for any $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$ are safely transformed into aligned intervals with arrival sequence α^*. This means that, for each $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$, there exists a prefix interval w.r.t. $J_{k,\ell}$ that is overloaded also under α^*. Overload probabilities of any interval I where $|I| < D_k$ can be ignored, since for a job of τ_k to miss its deadline, the busy prefix interval's length must be at least D_k; hence $t_s \leq d_{k,l} - D_k$. The sum of the overload probabilities of all prefix intervals starting at points in $A(0, d_{k,\ell} - D_k)$ thus accounts for each $\vec{x} \in \widehat{\mathcal{X}(\alpha)}$ at least once. ■

While α^* is periodic, we still cannot compute the right-hand side of Inequality (12) *a priori* based on Eqs. (5) and (6) because $d_{k,\ell}$ is generally unbounded and we hence have no way to enumerate $A(0, d_{k,\ell} - D_k)$. To limit the analysis space to a finite set of prefix intervals, we first observe from reasoning analogously as in the proof of Lemma 4 that busy intervals of length exceeding H have no impact on $J_{k,\ell}$'s DFP. Hence, independent of the actual deadline of $J_{k,\ell}$, the longest relevant prefix interval is bounded by the hyperperiod.

Lemma 11: *Inequality* (12) *can be rewritten as*

$$\Phi_{k,\ell} \leq \sum_{t_s \in A(d_{k,\ell} - H, d_{k,\ell} - D_k)} \mathbb{P}(S_{[t_s, d_{k,\ell}]} > d_{k,\ell} - t_s), \quad (13)$$

when all tasks are aligned with $J_{k,\ell}$.

Since the proof of Lemma 11 follows similar steps as the proof of Lemma 4, it is omitted due to space limitations.

The right-hand side of Inequality (13) is still not statically computable since it continues to depend on $d_{k,\ell}$, which is unknown *a priori*. However, the transformations that allow us to calculate $\Phi_{k,\ell}$ based on Lemmas 10 and 11 start from an arbitrary given arrival sequence α for task set Γ, but always end up in the same worst-case scenario: all tasks are aligned to $d_{k,\ell}$ in a periodic arrival sequence that is legal according to the parameters of Γ. Furthermore, the longest interval that must be considered starts at $d_{k,\ell} - H$, independently of the job $J_{k,\ell}$. Since we always arrive at the same worst-case scenario, we can shift the timeline to eliminate the dependency on $d_{k,\ell}$.

Theorem 1: *For task τ_k in a given task set Γ, the task DFP Φ_k is bounded from above by*

$$\Phi_k^* = \sum_{t_s \in A(0, H-D_k)} \mathbb{P}(S_{[t_s,H]} > H - t_s), \tag{14}$$

where $A(0, H - D_k)$ is the set of all arrival times in $[0, H - D_k]$ when all tasks are aligned with the job $J_{k,H/T_k}$ with deadline $d_{k,H/T_k} = H$.

Proof: Let α^* be the periodic arrival sequence wherein all tasks are aligned to H. For the job $J_{k,\ell}$ that exhibits Φ_k under any given α, the arrival sequence can be transformed into α' that is aligned with $d_{k,\ell}$ for all tasks in a series of safe transformations according to the arguments in this subsection. By changing the time base of the system (i.e., by shifting "time zero" and adjusting all arrival times and deadlines in α' by $\Delta = H - d_{k,\ell}$), we obtain the stated bound. ■

Since the lemmas in this subsection and Theorem 1 hold for any $\tau_k \in \Gamma$, we arrive at the following conclusion.

Corollary 1: *The worst-case arrival sequence α^* is identical for all $\tau_k \in \Gamma$.*

The arrival sequence α^* is obtained by releasing each task in $\tau_i \in \Gamma$ at time $\phi_i^* = T_i - D_i$ and periodically thereafter. Based on Theorem 1 and Corollary 1, we can calculate the task DFP for each task simultaneously by enumerating the intervals $[t_s, H)$ for all $t_s \in A(0, H)$ and summing up the relevant calculated overload probabilities on a per-task basis for each task τ_k (for $t_s < H - D_k$). Hence, we can upper-bound the WCDFP for all tasks by considering $O(|H|)$ intervals in total.

C. Runtime Improvement

Until now, we have examined how the WCDFP under EDF can generally be calculated and how to over-approximate this probability by restricting the number of intervals under consideration to $O(|H|)$. However, since the number of jobs in the hyperperiod is generally exponential with respect to the number of tasks, and as the evaluation of each interval is computationally expensive, we still must reduce the number of intervals considered to scale to practical workload sizes with a larger number of tasks and/or long hyperperiods. Such a culling of considered intervals is possible due to the work-conserving nature of EDF, as noted by the following property.

Lemma 12: *For the arrival sequence α^* in Theorem 1 and a given interval $[t_1, H]$ with $t_1 \in A(0, H - D_k)$, the portion $\Phi_k^{t_s<t_1}$ of the DFP Φ_k that is contributed by intervals $[t_s, H]$ with $t_s \in A(0, H - D_k)$ and $t_s < t_1$ is upper-bounded by*

$$\Phi_k^{t_s<t_1} \leq \mathbb{P}_{busy}([t_1, H]), \tag{15}$$

where $\mathbb{P}_{busy}([t_1, H])$ is the probability that the processor does not idle in the interval $[t_1, H]$.

Proof: By Definition 2 and the definition of α^*, the DFP Φ_k is the probability that the first deadline miss occurs at time H. For a deadline miss to occur at time H, a busy interval must start some time before H and not end by H. If a busy interval starts at $t_s < t_1$ and does not end by H, then the processor is necessarily busy throughout $[t_1, H]$, as otherwise a quiet time would occur before H and the busy interval would end prematurely. Hence, the total probability of a busy interval starting at any time $t_s < t_1$ and resulting in a deadline miss at time H is upper-bounded by $\mathbb{P}_{busy}([t_1, H])$, since this pessimistically assumes a deadline miss whenever the processor does not idle in $[t_1, H]$ (i.e., a necessary condition for a deadline miss is interpreted as a sufficient condition). ■

Lemma 12 effectively allows reducing the number of considered intervals with an additional bounded error. The idea is to stop the calculation when $\mathbb{P}_{busy}(I)$ of the currently considered interval $I = [t_x, H)$ sinks below a given threshold value. Specifically, let Φ_i^+ denote the current estimate of Φ_i^* when encountering t_x in the computation of Eq. 14 (i.e., Φ_i^+ is the partial sum accumulated so far). The given threshold for breaking out of the calculation may either be a fixed absolute threshold, or a threshold relative to the current $\max_i\{\Phi_i^+\}$ probability estimate. Once either threshold is exceeded, we set $\Phi_i^* = \Phi_{\tau_i}^+ + \mathbb{P}_{busy}(I)$ for each task τ_i in the set.

The final piece of the puzzle is a bound on $\mathbb{P}_{busy}(I)$, which we obtain by observing that, in addition to the $N_i(t_x, H)$ jobs that must finish in $I = [t_x, H)$, each task may have released one additional job prior to t_x that can also be serviced in I. This can be seen as considering the demand in a fictitious interval I^+ of the same length as I wherein the number of jobs of each task that must be completed in I^+ exceeds the number of jobs that must be completed in I by one. If this demand over I^+ is larger than $|I|$, then the processor does not necessarily idle at some time in I.

$$\mathbb{P}_{busy}(I) \leq \sum_{\vec{\boldsymbol{x}} \in \mathcal{X}(I^+)} \mathbb{P}(\boldsymbol{X}(I^+) = \vec{\boldsymbol{x}}) \cdot [\![S_{I^+}(\vec{\boldsymbol{x}}) > |I|]\!]_{\mathbb{1}} \tag{16}$$

VII. Dependent Task Modes

So far, we have assumed that the execution modes of jobs and hence the related random variables are independent. This assumption, however, is not always realistic. As an example, assume a periodic trigger (e.g., a sensor or task) that, for certain rare inputs, triggers more complex calculations in a set of related tasks. In this scenario, each of these tasks has a "typical" execution mode and rare "exceptional" modes with increased execution time, but the exceptional executions happen in the same time interval in a correlated fashion since they are caused by the same periodic trigger. In the following, we explain why it is important to account for such scenarios and how our approach can be extended accordingly.

Assuming independent random variables instead of correlated exceptional execution can result in an unbounded error, as sketched next. Let $\tau_1 = \tau_2 = ((0.5, 1 + \varepsilon), 2, 2)$ be two synchronously released tasks, let 0.5 and $1+\varepsilon$ be the execution times in typical and exceptional mode, respectively, where the execution mode of τ_2 depends on the execution mode of τ_1, let τ_1 be in exceptional mode with probability 0.5, and let $\varepsilon > 0$. Thus the probability that τ_2 is in exceptional mode is also 0.5. This task set overloads the interval $[0, 2]$ with probability 0.5, as either both are in typical or both are in exceptional mode. However, if the random variables for τ_1 and τ_2 are considered to be independent, the calculated probability

is 0.5^2. This probability can be further reduced to 0.5^{m-1} by dividing τ_2's parameters by m so that it releases $m > 1$ jobs in $[0, 2]$. Since $\frac{|0.5-0.5^{m-1}|}{0.5^{m-1}}$ approaches infinity for increasing m, the relative error can be arbitrarily large.

Revised system model. We assume a set of dependent tasks Γ^D in addition to the independent tasks Γ. Each task $\tau_i \in \Gamma^D$ is defined by $\tau_i = (\vec{C}_i, D_i, T_i, \Gamma^i)$, where $\vec{C}_i = \langle C_{i,1}, \ldots, C_{i,h} \rangle$ with $C_{i,k-1} \leq C_{i,k}$ for $k \in \{2, ..., h\}$ are the execution times of τ_i in its h distinct modes and Γ^i details the set of tasks that τ_i depends on. Each element of Γ^i is a tuple (τ_j, o_j^i, L_j^i), where τ_j is a task (either from Γ or Γ^D) that triggers exceptional behaviour in τ_i, $o_j^i \in \mathbb{N}$ determines the maximum number of jobs of τ_i that one job of τ_j can affect, and L_j^i bounds the interval after the release of a job of τ_j in which jobs of τ_i can be affected. When τ_i depends on τ_j, we assume that, if τ_j is executed in an exceptional mode $k > 1$, then the next o_j^i jobs of τ_i are executed in mode k, too. We denote this as $\tau_j \prec \tau_i$ and say that τ_j triggers mode k in τ_i. If a job depends on jobs from multiple tasks and/or multiple jobs from the same task, it executes the maximum triggered mode. Each dependent task $\tau_j \in \Gamma^D$ may depend both on independent tasks and on other dependent tasks in $\Gamma^D \setminus \tau_j$. However, the dependencies in the system must form a partial order for the trigger relation $\prec$. That is, we allow acyclic (branching) dependency chains.

Resulting challenges. The general concept for the calculation of the WCDFP remains the same as detailed in the previous section: iterate over intervals of increasing length, over-approximate the overload probability for each interval, and sum up these overload probabilities. Hence, the main difference is how the overload probability for each individual interval is calculated when jobs have dependencies. Thus, in the following we focus on finding the overload probability for one specific interval I. The probabilistic demand of the tasks in Γ can be calculated with Eq. 4 as before. The new challenge is to bound the probabilistic demand of the tasks in Γ^D.

In the independent scenario, Lemma 1 ensures that we can greedily maximize the number of jobs in a given interval (i.e., release jobs periodically) to maximize the probabilistic demand. This cannot directly be extended to the dependent case, since for strictly periodic releases the number of jobs a specific task τ_i releases in any interval $L = [t_s, t_f)$ depends not only on $|L|$, but also on the values of t_s and t_f (i.e., for two intervals L and L' with $|L| = |L'|$, the number of jobs τ_i releases in L and L' is not necessarily the same). For sporadic tasks, this leads to situations where, for a given task τ_i, postponing a job release can increase the number of jobs of τ_i in L and hence the total probabilistic demand in an interval of interest I that includes L, even if the number of jobs of τ_i in I is decreased, as depicted in Fig. 1.

Therefore, precisely determining the overload probability in a given interval for dependent sporadic tasks requires solving a complex combinatorial problem, even in seemingly simple cases (e.g., exceptional execution is triggered every tenth time a periodically refreshed sensor is read). We therefore instead seek to conservatively bound the overload probability.

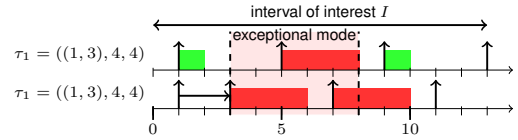

Fig. 1. The probabilistic demand of periodic releases of τ_1 (with $c_{i,1} = 1$, $c_{i,2} = 3$, and $D_i = T_i = 4$) over the interval $I = [0, 12]$ is 5 if the number of jobs in I is maximized (upper timeline), but 6 if the release at time 1 is postponed to time 3. However, in this situation the number of jobs that are released in and must finish in I is reduced from 3 to 2. There is no pattern that releases 3 jobs in I and 2 in the interval triggering exceptional mode.

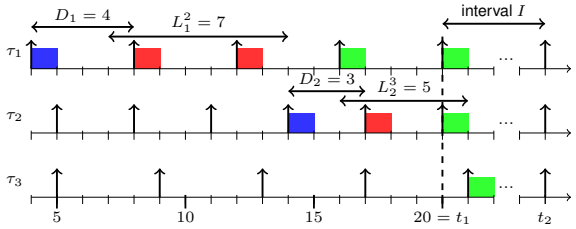

Fig. 2. Upper-bounding the number of jobs released before the interval I that can still affect jobs of τ_3 in I for the task chain $\lambda_k = \tau_1 \prec \tau_2 \prec \tau_3$.

Analysis outline. For any task τ_i (i.e., in Γ or in Γ^D), at most $N_i(I) = \left\lfloor \frac{|I|-D_i}{T_i} \right\rfloor + 1$ jobs contribute demand to $I = [t_1, t_2]$ (i.e., jobs that are released and must be finished in I). For each task $\tau_j \in \Gamma$, the number of times exceptional execution occurs is given by the concrete assignment $\vec{x} \in \mathcal{X}(I)$ of the random variables related to τ_j. Thus, when determining the demand in I, we first evaluate the tasks in Γ and then the tasks in Γ^D. The tasks in Γ^D are evaluated in order of the trigger relation $\prec$. For each task $\tau_i \in \Gamma^D$ we have to determine the number of exceptional jobs q_i^k (for $k \in \{2, ..., h\}$) in I based on the modes of the tasks $\tau_j \in \Gamma^i$ in I. For a dependent task $\tau_i \in \Gamma^D$, the execution mode of some of these jobs can depend on jobs that are released strictly before t_1. This situation is depicted in Figure 2. Our goal is to bound the number of jobs we have to consider in addition to the jobs in I.

Relevant jobs. For the task chain $\lambda_k = \tau_1 \prec \tau_2 \prec \tau_3$ in Fig. 2, jobs of τ_3 may depend on any jobs of τ_2 that are executed up to L_2^3 time units before I. This includes any job released in t_1's prefix interval of length L_2^3 and one job which is released before said interval but executed in it. Since this job is released at most D_2 units before, we can restrict the analysis of τ_2 to an interval $I_2 = [t_1 - D_2 - L_2^3, t_2]$ of length $|I_2| = D_2 + L_2^3 + |I|$. For τ_1, we have to include all jobs released in $I_1 = [t_1 - D_1 - L_1^2 - D_2 - L_2^3, t_2]$.

Hence, for each $\tau_i \in \Gamma$, we first must determine an extended interval I_i^* that contains all jobs that can directly or indirectly affect jobs in I, from which we will derive an extended set of corresponding random variables with cardinality $N_i(I_i^*) \geq N_i(I)$. To this end, we initially set $I_i^* = I$ for each task τ_i, and then traverse the dependent tasks. For each $\tau_j \in \Gamma^D$, we consider all chains ending in τ_j, since for the last task in a chain no additional jobs have to be considered. We consider each $\tau_i \in \Gamma^j$ by calculating the length of $I_j^i = I + L_i^j + D_i$ and, for each of these τ_i, recursively calculate $I_j^k = I_j^i + L_k^i + D_k$ for each of τ_i's predecessors $\tau_k \in \Gamma^i$, etc., stopping the recursion after we

reach an independent task. That is, for dependencies $\tau_1 \prec \tau_3$, $\tau_2 \prec \tau_3$, and $\tau_3 \prec \tau_4$ with τ_1, $\tau_2 \in \Gamma$, we consider $\tau_1 \prec \tau_3$, $\tau_2 \prec \tau_3$, $\tau_1 \prec \tau_3 \prec \tau_4$, and $\tau_2 \prec \tau_3 \prec \tau_4$ (i.e., all (sub-)chains starting with a dependent task). During this calculation, we update I_i^* when we find an interval I_j^i related to τ_i with $I_j^i > I_i^*$. Afterwards, we set $N_i(I_i^*) = \left\lfloor \frac{|I_i^*| - D_i}{T_i} \right\rfloor + 1$ for each task. Thus the extended set of random variables $\boldsymbol{X}(I^*)$ for each independent $\tau_i \in \Gamma$ has $N_i(I_i^*)$ related random variables and no random variables for the dependent tasks $\tau_i \in \Gamma^D$.

Overload probability. Given $\vec{x} \in \mathcal{X}(I^*)$, we count the number of times $\tau_j \in \Gamma$ is executed in mode k in I, denoted as q_j^k for $k \in \{1, ..., h\}$. The remaining step is to calculate q_i^k for $k \in \{1, ..., h\}$ and each task $\tau_i \in \Gamma^D$. Since we evaluate Γ first and the tasks in Γ^D according to the partial order $\prec$, when considering task τ_i, the number of exceptional jobs q_j^k is known for each $\tau_j \in \Gamma^i$ and each $k \in \{1, ..., h\}$.

For $\tau_i \in \Gamma^d$, we sum up the total number of jobs in each exceptional mode as $q_i^k = \sum_{\tau_j \in \Gamma^i} o_j^i \cdot q_j^k$ for $k \in \{2, ..., h\}$. That is, we over-approximate the number of exceptional jobs that can affect any job in I by greedily assuming that each exceptional job of each task $\tau_j \in \Gamma^i$ triggers a maximum number of exceptional jobs of τ_i. Afterwards, if $\sum_{k=2}^h q_i^k \leq N_i(I^*)$, we set $q_i^1 = N_i(I^*) - \sum_{k=2}^h q_i^k$ (i.e., all remaining jobs of τ_i in I^* execute in mode 1). If $\sum_{k=2}^h q_i^k > N_i(I^*)$, then we enumerated more jobs in exceptional mode than there actually are in I^* due to the over-approximation. In this case, we reduce q_i^2 (and if that is not sufficient q_i^3 and so on) to ensure $\sum_{k=1}^h q_i^k = N_i(I^*)$ (i.e., we retain the jobs with largest execution times).

After q_i^k has been calculated for each $\tau_i \in \Gamma^D$ and k, we upper-bound the demand over I for the given $\vec{x}$. To this end, we need to consider $N_i(I)$ jobs of each task $\tau_i \in \Gamma^D$. Thus, if $N_i(I^*) > N_i(I)$, we remove $N_i(I^*) - N_i(I)$ jobs, again keeping the $N_i(I)$ jobs with the largest execution times.

This procedure safely upper-bounds the demand of τ_i in I for each $\tau_i \in \Gamma^D$ since we conservatively assume that each exceptional job of each task $\tau_j \in \Gamma^i$ triggers the maximum number of exceptional jobs of τ_i, and since in the case of over-approximation we retain the jobs with the largest execution times. After we determined q_i^k for each task in Γ^D (for a given $\vec{x}$), we determine the demand in I (assuming $\vec{x}$) as

$$S_I(\vec{x}) = \sum_{\tau_i \in \Gamma} \sum_{j=1}^{N_i(I)} C_i(\vec{\boldsymbol{x}}_{i,j}) + \sum_{\tau_\ell \in \Gamma^D} \sum_{k=1}^{h} C_{\ell,k} \cdot q_\ell^k. \quad (17)$$

Finally, the overload probability of I can be upper-bounded by iterating over all $\vec{x} \in \mathcal{X}(I)$ using Eq. (5) with Eq. (17).

VIII. Evaluation

We conducted experiments to assess **(i)** whether the proposed probabilistic analysis for independent tasks allows substantially higher system utilization compared to conventional deterministic analysis (provided a low rate of deadline misses is deemed acceptable), **(ii)** how well the approximations scale to sets with tens of tasks and/or long hyperperiods, **(iii)** what percentage of the hyperperiod is covered before the analysis is stopped, and **(iv)** how dependent tasks affect the WCDFP.

Setup. We randomly generated workloads consisting of sporadic tasks with two execution modes, representing "typical" execution behavior and a rare "exceptional" mode. For a given number of tasks n and a target total *typical-mode utilization* $U_{sum} = \sum_{i=1}^n C_{i,1}/T_i$, we generated for each task τ_i a typical-mode utilization U_i using the UUniFast generator [5]. Task periods were drawn from a log-uniform distribution spanning $[10\,\text{ms}, 1000\,\text{ms}]$, as suggested by Emberson et al. [12]. We assigned implicit deadlines to all tasks (i.e., $D_i = T_i$) and set $C_{i,1} = U_i \cdot T_i$ and $C_{i,2} = r \cdot C_{i,1}$ according to a configurable *exceptional-cost multiplier* r. We set the execution-mode probabilities $\mathbb{P}_i(2) = p$ and $\mathbb{P}_i(1) = 1 - p$ for each task based on a configurable *exceptional-case probability* p. Following prior work [7]–[9], [21], our setup reflects scenarios in which modes differ significantly in execution times and high execution times occur only infrequently. If these conditions are not met, a deterministic analysis is advisable instead.

As discussed in Section V, we transferred two state-of-the-art methods for calculating the overload probability of individual intervals to the EDF setting: *task-level convolution* (TLC) [21], and the improved version [8] of the *Chernoff bound* (CB) approach [7]. We implemented both algorithms in Python 2.7 and conducted the experiments on a Lenovo Thinkpad x280 with an Intel Core i7-8550U clocked at 1.8 GHz with 16 GB RAM and 8 MB L3 cache. The calculation was stopped when an interval I with $\mathbb{P}_{busy}(I) \leq 0.1 \cdot \Phi^+$ was found, where Φ^+ is the current WCDFP estimate.

Experiment 1: Efficiency gains. To assess the achievable resource utilization, we fixed either p or r and evaluated the effect of changing the other parameter. For each combination of p and r, and each typical-mode utilization U_{sum} between 0% and 100% in steps of 2%, we examined 100 task sets with $n = 5$. We bounded each workloads's system WCDFP with TLC and tracked the fraction of task sets below six WCDFP thresholds and the deterministic baseline, as shown in Fig. 3.

All insets show substantial gains in achievable system utilization (compared to the deterministic baseline) even if more than 99.9999% of the jobs must meet their deadline. For instance, when comparing in Figs. 3(a)–(e) the points at which each curve crosses the threshold of a 75% acceptance ratio, we observe at least 12–18 percentage points higher typical-mode utilization if a nonzero WCDFP is allowed. Only in Fig. 3(f), where $r = 4.5$, is the difference less than 10 percentage points. However, the relative gain is still large, since in this rather extreme setting only task sets with less than $\approx$ 22% typical-mode utilization are deterministically schedulable. As expected, in all settings, a larger permissible WCDFP results in higher achievable resource utilization.

Figs. 3(a)–(c) show the effect of fixing $r = 2$ while varying $p \in \{0.025, 0.01, 0.001\}$. The gap to the deterministic case clearly increases as the exceptional mode becomes less likely. Figs. 3(d)–(f) show the effect of varying $r \in \{r = 1.5, 3, 4.5\}$ for a fixed $p = 0.01$. The general trends remain roughly the same as already described, with the main difference being that

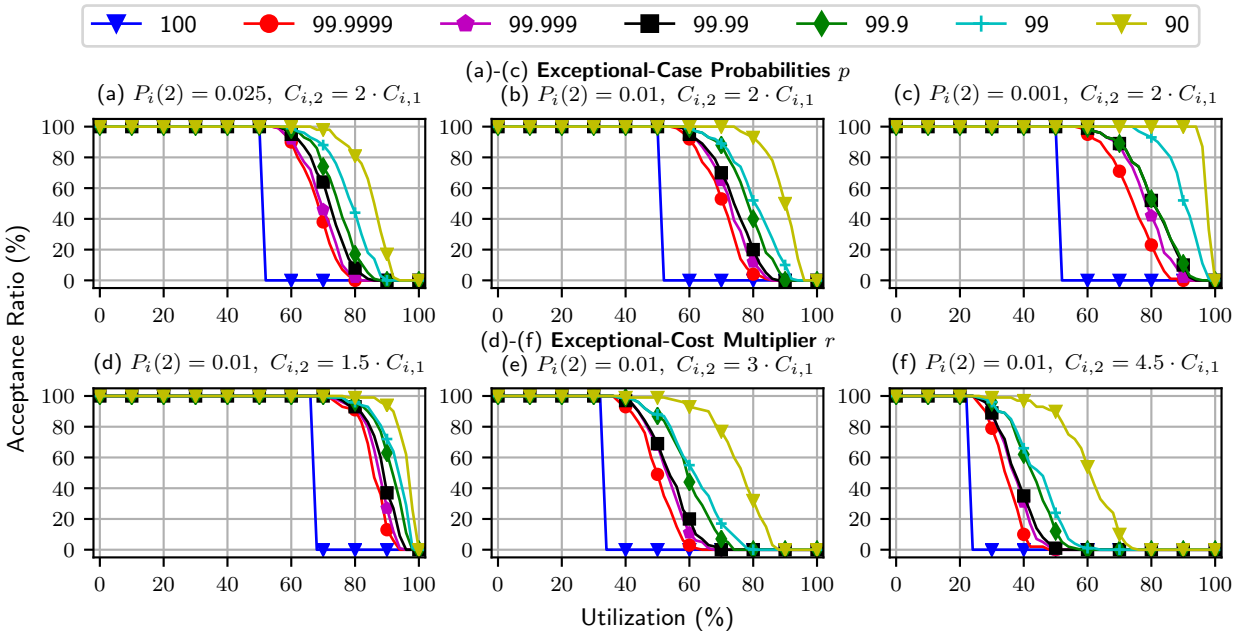

Fig. 3. Achieved acceptance ratio assuming various levels of acceptable WCDFP thresholds for varying exceptional-case probabilities p (insets (a)–(c)) and varying exceptional-cost multipliers r (insets (d)–(f)). Curves labeled 90 indicate how many of the task sets had a WCDFP below 10%, curves labeled 99 indicate how many had a WCDFP below 1%, etc. If the WCDFP is at most 10%, each job has a probability of 10% or less to miss its deadline (if jobs that overrun are aborted). Curves labeled 100 indicate classic deterministic schedulability (i.e., the WCDFP is 0) and thus represent the baseline.

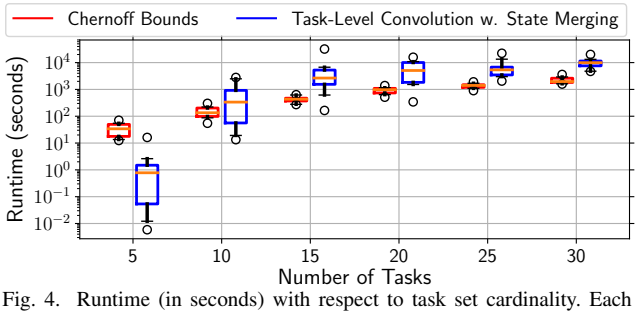

Fig. 4. Runtime (in seconds) with respect to task set cardinality. Each box shows the range between the first and the third quartile of observations, the orange line is the median, the whiskers cover the $[5; 95]$ percentile range, and the maximum and minimum observations are shown as individual points.

the absolute difference in percentage points decreases when r increases, which however does not affect the relative gains since the baseline diminishes even more rapidly.

Overall, the experiment demonstrates that the proposed probabilistic analysis allows for substantially better resource utilization if infrequent deadline misses are permissible.

Experiment 2: Scalability. To assess the runtime of the approach, we created 10 tasks sets with cardinality 5–30 in steps of 5, with $U_{sum} = 80\%$, $p = 0.025$, and $r = 2$. As can be seen in Figs. 3(a)–(c), for $r = 2$ the acceptance-ratio curves decrease rapidly around $U_{sum} = 80\%$. The choice of $U_{sum} = 80\%$ hence ensures that the calculation does not become too easy (which might be the case if the WCDFP is either much smaller or larger than the considered thresholds).

Applying task-level convolution directly (i.e., without further optimizations) did not yield timely results for more than 10 tasks, even though it previously scaled up to 100 tasks in the case of static-priority scheduling [21], since longer intervals with a larger number of jobs have to be considered in the case of EDF. Thus, we repeatedly used the *state merging* runtime optimization [21] as discussed in Section V, each time over-approximating with an added pessimism of up to 10^{-7}.

Fig. 4 shows the results using boxplots with a logarithmic scale for the y-axis. For each cardinality, TLC's runtime exhibits more variance than CB's runtime. We observed TLC's maximum runtime for a set with 15 tasks ($\approx$ 8:54 hrs). From 10 to 30 tasks, the average runtime of TLC increases by roughly a factor of 13, and by a factor of 15 in the case of CB. For 30 tasks, CB's runtime ranged from $\approx$ 0:26 hrs to $\approx$ 1:01 hrs (avg. $\approx$ 0:37 hrs) and TLC's runtime ranged from $\approx$ 1:18 hrs to $\approx$ 5:36 hrs (avg. $\approx$ 2:48 hrs).

Overall, this experiment demonstrates that reasonably sized workloads (up to 30 tasks) can be analyzed in manageable time ($\approx$ 1–3 hours on average) on consumer hardware. In contrast, prior methods are impractical due to scalability issues (e.g., [11], [20] evaluated only 7 and 25 jobs in the hyperperiod).

Experiment 3: Hyperperiod coverage. For both approaches, the fraction of the hyperperiod that had to be analyzed was reduced drastically by Lemma 12. The longest considered interval was always similar for both approaches at roughly 5 to 12 times the largest period, without any apparent relation to the number of tasks. Since the hyperperiod typically grows with the number of tasks, the considered fraction of the hyperperiod ranged from $\approx 10^{-17}$ (5 tasks) to $\approx 10^{-60}$ (30 tasks). This shows that considering all possible intervals would be computationally intractable, even for 5 tasks.

Experiment 4: Impact of dependencies. We evaluated 50 tasks sets with 10 tasks each, randomized as before, with $P_i = 0.01$ and $r = 2$. As the base setting, we inserted dependencies by picking $d = 5$ tasks as dependent with a total of $e = 8$ dependencies forming a partial order. For

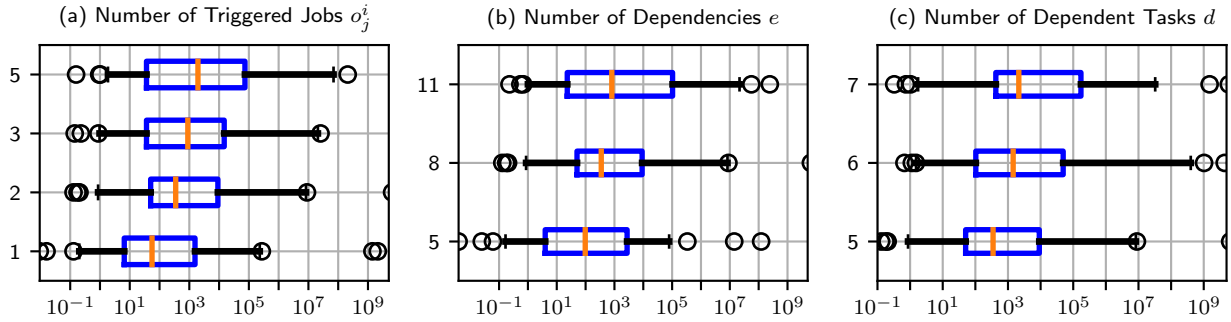

Fig. 5. The dependent WCDFP relative to the independent WCDFP (i.e., Φ_d/Φ_i) for a varying number of triggered jobs, dependencies, and dependent tasks.

each dependency $i \prec j$, we set $o_j^i = 3$ and $L_j^i = 3 \cdot T_i$. We compared the resulting dependent WCDFP Φ_d to the independent WCDFP Φ_i for the same tasks while varying d, e, and o_j^i. Fig. 5 depicts the results (using boxplots as in Fig. 4).

In Fig. 5(a), we observe that, when each exceptional job triggers one exceptional execution in the related task, Φ_d is in most cases already 1-3 orders of magnitude larger than in the independent case. The relation further increases with the number of dependent jobs. Similarly, when the number of dependencies (Fig. 5(b)) or the number of dependent tasks (Fig. 5(c)) is increased, the relative difference between Φ_d and Φ_i increases as well, since a single exceptional job will usually affect multiple dependent jobs. It is further worth noticing that $\frac{\Phi_d}{\Phi_i}$ can be extremely large due to very low Φ_i values, which in our evaluation ranged from $2 \cdot 10^{-14}$ to $2 \cdot 10^{-1}$.

For a few task sets, Φ_d was actually smaller than Φ_i. This occurred on average for ≈ 3.4 task sets per setting, with a maximum of 8 workloads in the case of $d = 5$). The reason is that TLC needed to over-approximate via state merging in the independent setting (recall Section V), whereas in the dependent setting TLC was applied directly due to the smaller number of independent tasks. In some cases, these over-approximations outweigh the increase due to dependencies.

From the experimental evidence, we conclude that the increase in the WCDFP due to acyclic dependencies is often quite substantial. Assuming independence when tasks are not independent may significantly under-estimate the WCDFP.

IX. Related Work

We focus on work most closely related to this paper. A comprehensive survey of probabilistic schedulability analyses has been recently presented by Davis and Cucu-Grosjean [10].

For periodic real-time task systems, Diaz et al. [11] developed a framework for analyzing the WCDFP based on job-level convolution. Tanasa et al. [20] presented approximations of arbitrary execution-time distributions and a customized decomposition procedure to search the space of possible combinations. The decomposition leads to a list with $O(4^{|J|})$ elements for $|J|$ jobs. These two results suffer from scalability limitations due to their exponential-time complexity with respect to the number of jobs (i.e., [11] and [20] reported results for at most 7 and 25 jobs in the hyperperiod, respectively).

For sporadic tasks under static-priority scheduling, Maxim and Cucu-Grosjean [16] provided a probabilistic response-time analysis via resampling. Axer et al. [2] considered the response-time distribution under non-preemptive static-priority scheduling. Ben-Amor et al. [4] extended [16] to analyze precedence-constrained tasks under EDF. All of them convolve the probabilistic demand whenever a new job arrives, hence the analysis runtime strongly depends on the number of jobs.

Instead of evaluating the jobs by convolving the previous state space with each newly considered job, task-level convolution by von der Brüggen et al. [21] considers the intervals of interest individually and utilizes multinomial distributions. They also propose different over-approximations of the WCDFP based on task-level convolution.

The WCDFP of preemptive static-priority uniprocessor scheduling can also be approximated by analytical techniques based on the Chernoff bound as proposed by Chen and Chen [7] and Chen et al. [8], and by Hoeffding's and Bernstein's inequalities as proposed by von der Brüggen et al. [21].

The analytical methods in [7], [8], [21] and task-level convolution [21] have been shown to scale to more than 20 tasks and more than a thousand jobs in the hyperperiod, hence we built on them in our work (recall Sections V and VIII).

Concurrently with this study, Markovic et al. [15] proposed the use of circular convolution techniques based on the Fourier Transform to substantially reduce the runtime of convolution-based methods, and Bozhko et al. [6] proposed Monte Carlo simulation as an alternative to both convolution-based methods and analytical bounds. As both Markovic et al. and Bozhko et al.'s proposals scale well to larger task sets in the static-priority case [6], [15], they may be applicable to EDF, too.

X. Conclusion

We have studied the problem of bounding the WCDFP of independent constrained-deadline periodic or sporadic task sets under preemptive uniprocessor EDF. The central contribution of this paper is the first WCDFP over-approximation with affordable analysis runtimes. Specifically, the proposed approach scales to workloads of nontrivial size with tens of tasks and thousands of jobs per hyperperiod.

Our evaluation showed reasonable runtimes for randomized tasks sets with up to 30 tasks (i.e., on average $\approx$ 0:37 hours for Chernoff bound approach and $\approx$ 2:48 hours for task-level convolution). It also underscored that accepting even a low WCDFP enables significant gains in resource utilization.

Finally, we showed how to bound the WCDFP in the presence of dependent tasks and demonstrated how dangerous an unjustified independence assumption can be.

ACKNOWLEDGEMENTS

This result is part of two projects (PropRT and TOROS) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements No. 865170 and No. 803111). This paper is supported by DFG, as part of the Collaborative Research Center SFB876, project A1. Parts of this research have been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01IS18038B).

REFERENCES

[1] IEC-61508 Edition 2.0. Functional safety of electrical / electronic / programmable electronic safety-related systems ed2.0. Technical report, International Electrotechnical Commission (IEC), 2010. URL: http://www.iec.ch/functionalsafety/standards/page2.htm.

[2] Philip Axer and Rolf Ernst. Stochastic response-time guarantee for non-preemptive, fixed-priority scheduling under errors. In *Design Automation Conference*, 2013.

[3] Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium*, 1990.

[4] Slim Ben-Amor, Dorin Maxim, and Liliana Cucu-Grosjean. Schedulability analysis of dependent probabilistic real-time tasks. In *Real-Time Networks and Systems*, 2016.

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

[6] Sergey Bozhko, Georg von der Brüggen, and Björn B. Brandenburg. Monte carlo response-time analysis. In *Real-Time Systems Symposium, 2021*.

[7] Kuan-Hsun Chen and Jian-Jia Chen. Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In *Symposium on Industrial Embedded Systems*, 2017.

[8] Kuan-Hsun Chen, Niklas Ueter, Georg von der Brüggen, and Jian-Jia Chen. Efficient computation of deadline-miss probability and potential pitfalls. In *Design, Automation & Test in Europe*, 2019.

[9] Kuan-Hsun Chen, Georg von der Brüggen, and Jian-Jia Chen. Analysis of deadline miss rates for uniprocessor fixed-priority scheduling. In *Real-Time Computing Systems and Applications*, 2018.

[10] Robert I. Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *LITES*, 6(1):03:1–03:60, 2019.

[11] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, Lucia Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium*, 2002.

[12] Paul Emberson, Roger Stafford, and Robert I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2010.

[13] Jie S. Hu, Feihui Li, Vijay Degalahal, Mahmut T. Kandemir, Narayanan Vijaykrishnan, and Mary Jane Irwin. Compiler-directed instruction duplication for soft error detection. In *Design, Automation and Test in Europe*, 2005.

[14] ISO-26262-1:2011. ISO/FDIS 26262: Road vehicles - functional safety. Technical report, International Organization for Standardization (ISO), 2000. URL: https://www.iso.org/standard/43464.html.

[15] Filip Markovic, Alessandro Vittorio Papadopoulos, and Thomas Nolte. On the convolution efficiency for probabilistic analysis of real-time systems. In *Euromicro Conference on Real-Time Systems, 2021*.

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

[17] Bogdan Nicolescu, Raoul Velazco, Matteo Sonza-Reorda, Maurizio Rebaudengo, and Massimo Violante. A software fault tolerance method for safety-critical systems: effectiveness and drawbacks. In *Integrated Circuits and Systems Design*, pages 101–106, 2002.

[18] Nahmsuk Oh, Philip P. Shirvani, and Edward J. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Trans. Reliability*, 51(1):63–75, 2002.

[19] Semeen Rehman, Muhammad Shafique, Pau Vilimelis Aceituno, Florian Kriebel, Jian-Jia Chen, and Jörg Henkel. Leveraging variable function resilience for selective software reliability on unreliable hardware. In *Design, Automation and Test in Europe*, 2013.

[20] Bogdan Tanasa, Unmesh D. Bordoloi, Petru Eles, and Zebo Peng. Probabilistic response time and joint analysis of periodic tasks. In *Euromicro Conference on Real-Time Systems*, 2015.

[21] Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently approximating the probability of deadline misses in real-time systems. In *Euromicro Conference on Real-Time Systems*, 2018.

[22] Dakai Zhu, Hakan Aydin, and Jian-Jia Chen. Optimistic reliability aware energy management for real-time tasks with probabilistic execution times. In *Real-Time Systems Symposium*, 2008.