Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems

Georg von der Brüggen
Department of Computer Science, TU Dortmund University, Germany
georg.von-der-brueggen@tu-dortmund.de
0000-0002-8137-3612

Nico Piatkowski
Department of Computer Science, TU Dortmund University, Germany
nico.piatkowski@tu-dortmund.de
0000-0002-6334-8042

Kuan-Hsun Chen
Department of Computer Science, TU Dortmund University, Germany
kuan-hsun.chen@tu-dortmund.de
0000-0002-7110-921X

Jian-Jia Chen
Department of Computer Science, TU Dortmund University, Germany
jian-jia.chen@cs.uni-dortmund.de
0000-0001-8114-9760

Katharina Morik
Department of Computer Science, TU Dortmund University, Germany
katharina.morik@tu-dortmund.de
0000-0003-1153-5986

Abstract

This paper explores the probability of deadline misses for a set of constrained-deadline sporadic soft real-time tasks on uniprocessor platforms. We explore two directions to evaluate the probability whether a job of the task under analysis can finish its execution at (or before) a testing time point t. One approach is based on analytical upper bounds that can be efficiently computed in polynomial time at the price of precision loss for each testing point, derived from the well-known Hoeffding's inequality and the well-known Bernstein's inequality. Another approach convolutes the probability efficiently over multinomial distributions, exploiting a series of state space reduction techniques, i.e., pruning without any loss of precision, and approximations via unifying equivalent classes with a bounded loss of precision. We demonstrate the effectiveness of our approaches in a series of evaluations. Distinct from the convolution-based methods in the literature, which suffer from the high computation demand and are applicable only to task sets with a few tasks, our approaches can scale reasonably without losing much precision in terms of the derived probability of deadline misses.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases deadline miss probability, multinomial-based approach, analytical bound

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2018.6

Related Version https://ls12-www.cs.tu-dortmund.de/daes/media/documents/publications/downloads/2018-brueggen-ecrts-deadline-miss-probability.pdf

30th Euromicro Conference on Real-Time Systems (ECRTS 2018).
Editor: Sebastian Altmeyer; Article No. 6; pp. 6:1–6:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This paper is supported by DFG, as part of the Collaborative Research Center SFB876 (http://sfb876.tu-dortmund.de/), project A1 and project B2.

1 Introduction

For many embedded systems, timeliness is an important feature, especially when such systems interact with physical environments. A stronger requirement of timeliness is to provide *hard* real-time guarantees, i.e., to ensure that the calculated results are not just functionally correct but also *always* delivered within given timing constraints. Such hard guarantees are necessary if any deadline miss can be catastrophic and should be avoided. By contrast, a weaker requirement of timeliness is to allow occasional deadline misses, called *soft* real-time systems. In this case the system can still function correctly as long as the deadline misses can be quantified and bounded. For example, the system may adopt fault tolerance techniques like checkpointing, redundant execution, etc. [**?**, **?**, **?**, **?**, **?**], to neglect transient faults resulting from electromagnetic interference and radiation [**?**]. Although the additional computation incurred by such methods may lead to deadline misses, the system may still provide timing guarantees even without any online adaption [**?**]. A second example are the safety standards in the industry that require low (or very low) probability of failure (e.g., due to deadline misses) such as IEC-61508 [**?**] and ISO-26262 [**?**].

Probability theory is a basic language to describe probabilistic phenomenons, e.g., occasional deadline misses. It is based on the idea that most natural phenomena are either too complex to construct deterministic models or simply not fully observable but can be described in a probabilistic way. For example, we can establish probabilistic bounds on the worst-case execution times (WCETs) to model the execution of a task depending on the occurrence of soft errors and the triggered error recovery routines. This allows the system designer to provide probabilistic arguments based on the occurrence of error recovery. Otherwise, only the WCET, assuming that the recovery always takes place, has to be considered in the response time analysis, which is very pessimistic and therefore leads to overestimating the necessary system resources.

Probability of Deadline Misses: A key procedure needed for such soft real-time systems is the analysis of the probability of deadline misses for a real-time task. Now, we take a closer look of the problem by using the following example: Suppose that we have two periodic tasks τ_1 and τ_2 that release task instances, called jobs, periodically, starting from time 0. Each task $\tau_i \in \{\tau_1, \tau_2\}$ has two versions of execution times $C_{i,1}$ and $C_{i,2}$ with probability $\mathbb{P}_i(1)$ and $\mathbb{P}_i(2)$, respectively. The period of task τ_1 is 1 and the period of task τ_2 is 100. We assume that task τ_1 always has a higher priority than task τ_2 and task τ_1 can always meet its deadline under a fixed-priority preemptive scheduling strategy in a uniprocessor system.

In this example, the system reboots if a job of task τ_2 is not finished before the next job of task τ_2 is released. Therefore, the probability of deadline misses corresponds to the probability of system rebooting. Essentially, we are interested to know whether a job of τ_2, arriving at time t_a, can finish its execution before $t_a + 100$. This can be achieved by the *convolution* of the probability density functions of the jobs' execution times. An intuitive procedure is to evaluate the probability of the accumulative execution time, denoted as *workload*, of the jobs released from time t_a to $t_a + \ell - 1$ (inclusive), starting from $\ell = 1, 2, 3, \ldots, 100$. When ℓ is 1, we have 2^2 combinations of the workload of the two jobs released at time t_a. When ℓ is 2, we can have up to $2^2 \times 2 = 2^3$ combinations of the workload. It is rather obvious that we can have up to 2^{101} combinations of the workload when ℓ is 100, which is *exponential* with respect to the number of jobs that may interfere with a job of task τ_2.

Since there are only two versions of task τ_1, there are in fact only $\ell+1$ different workload combinations of the ℓ jobs released from time t_a to time $t_a + \ell - 1$. As a result, there are only $2(\ell+1)$ different workload combinations of the jobs released from time t_a to $t_a + \ell - 1$. We can evaluate all of them from $\ell = 1, 2, \ldots, 100$. However, this remains inefficient as we are only interested in the probability of the deadline miss at time $t_a + 100$. For this example, we do not actually care about the individual execution versions of the 100 jobs of task τ_1 released from t_a to $t_a + 99$. Instead, we only care about their overall workload, which can be calculated by using a binomial distribution over 100 independent random variables with the same distribution. As a result, we only have to consider 101 different workload combinations for the jobs of τ_1. Together with the job of task τ_2, there are in fact only 2×101 different workload combinations.

These approaches are different realizations of the same concept to convolute the probability density functions of the jobs' execution times. However, depending on how the convolution is performed, the complexity can differ largely.

Related Work: As explained above for uniprocessor systems, it is necessary to safely derive (an upper bound on) the probability of a desired workload constraint to analyze the probability of deadline misses or the probabilistic response time. Towards this, for periodic real-time task systems, Diaz et al. [**?**] developed a framework for calculating the deadline miss probability based on convolution. Moreover, Tanasa et al. [**?**] used the Weierstrass Approximation to approximate any arbitrary execution time distributions and applied a customized decomposition procedure to search all the possible combinations, in which the decomposition results in a list with $O(4^{|J|})$ elements where $|J|$ is the number of jobs in the interval of interest. These two results have exponential-time complexity with respect to the number of jobs in the interval of interest. Therefore, both of them suffer from the scalability with respect to the number of jobs. In the experimental results in [**?**] and [**?**], they can derive the probability of deadline misses with 7 and 25 jobs in the hyper-period, respectively.

For sporadic real-time task systems, in which two consecutive jobs of a task do not have to be released periodically, Axer et al. [**?**] proposed to evaluate the response-time distribution and iterate over the activations of job releases for non-preemptive fixed-priority scheduling. Maxim et al. [**?**] provided a probabilistic response time analysis by assuming probabilistic minimum inter-arrival as well as probabilistic worst-case execution times for the fixed-priority scheduling policy. Ben-Amor et al. [**?**] extended the probabilistic response time analysis in [**?**], considering precedence constrained tasks. All these approaches convolute the probability whenever a new job arrives in the interval of interest. Therefore, the convolution procedure is also heavily dependent on the number of jobs in the interval of interest.

Due to the high complexity, these convolution-based approaches are not scalable with respect to the number of jobs in the interval of interest and, thus, infeasible. Approximation techniques can be used to provide an upper bound on the probability. For example, re-sampling [**?**] and dynamic-programming based on user-defined granularity can be applied to reduce the time complexity. Moreover, Chen and Chen [**?**] provided a scalable approximation based on the Chernoff bounds. The evaluation results in [**?**] confirm the applicability and the scalability of such approximations, even when considering 20 tasks and more than thousand jobs in the hyper-period.

Our Contributions: We consider the problem of determining the deadline miss probability of a task under uniprocessor fixed-priority preemptive scheduling when each task has distinct execution modes that are executed with a known probability distribution. Our main contributions are:

- We provide a novel approach based on the multinomial distribution that, compared to the traditional convolution-based approach, allows to calculate the deadline miss probability with better analysis runtime and without any precision loss.
- The analysis is enhanced by a state pruning technique that significantly improves the runtime as well as the scalability without any loss of precision.
- We further improve our approach by merging equivalence classes, thus further reducing the runtime of our analysis while the introduced precision loss can be bounded in advance.
- In the evaluation, we show that our approach is applicable for significantly larger task sets than the previously known convolution-based approaches by testing it for task sets of up to 100 tasks.
- Furthermore, we provide additional analytical bounds based on the Hoeffding's [?] and Bernstein's [?] inequalities. Our evaluations show that these inequalities lead to fast results and can be used if the over-approximation is acceptable.

2 Task Model, System Model, and Notation

We consider a given set of n independent periodic (or sporadic) tasks $\Gamma = \{\tau_1, \tau_2, \cdots, \tau_n\}$ in a uniprocessor system. Each task τ_i releases an infinite number of task instances, called jobs, and is defined by a tuple $((C_{i,1}, ..., C_{i,h}), D_i, T_i)$, where D_i is the relative deadline of τ_i and T_i is its minimum interarrival time. In addition, each task has a set of h distinct execution modes $\mathcal{M}$ and each mode j with $j \in \{1, ..., h\}$ is associated with a different worst-case execution time (WCET) $C_{i,j}$. We assume those execution modes to be ordered increasingly according to their WCETs, i.e., $C_{i,m} \leq C_{i,m+1}\ \forall m \in \{1, ..., h-1\}$. Furthermore, we assume that each job of τ_i is executed in one of those distinct execution modes. To fulfill its timing requirements in the j^{th} execution mode, a job of τ_i that is released at time t_a must be able to execute $C_{i,j}$ units of time before $t_a + D_i$. The next job of τ_i must be released at $t_a + T_i$ for a periodic task and for a sporadic task the next job is released at or after $t_a + T_i$. In this work, we focus on *implicit-deadline* task sets, i.e., $D_i = T_i$ for all tasks, and *constrained-deadline* task sets, i.e., $D_i \leq T_i$ for all tasks. The task set is assumed to be scheduled according to a preemptive fixed-priority scheduling policy, i.e., each task has a unique fixed priority, the priority cannot be changed during runtime, and the priority of each task instance is identical to the priority of the related task. At each point in time, the scheduler ensures that the job with the highest priority, among the jobs currently ready in the system, is executed. We assume that the tasks are indexed according to their priority, i.e., τ_1 has the highest and τ_n has the lowest priority. In addition, $hp(\tau_k)$ denotes the set of tasks with higher priority than τ_k and $hep(\tau_k)$ is $hp(\tau_k) \cup \{\tau_k\}$. For a task τ_i in $hp(\tau_k)$, $\rho_{i,t}$ is the maximum number of jobs that are released in an interval $[0, t)$, also called the interval of interest, and therefore interfere with task τ_k, i.e., the number of jobs released in the interval $[0, t)$ under the critical instance of τ_k. Furthermore, $\rho_{k,t}$ is the number of jobs of task τ_k in the analysis window. This notation implicitly assumes that the time window analyzed for τ_k starts at 0 for notational brevity. $\mathbb{P}_i(j)$ denotes the probability that a job of task τ_i is executed in mode j with related WCET $C_{i,j}$ and we assume that each job is executed in exactly one of these distinct execution modes, i.e., $\sum_{j=1}^{h} \mathbb{P}_i(j) = 1$. In addition, we assume that these probabilities are independent from each other according to the following definition:

▶ **Definition 1** (Independent Random Variables). *Two random variables are (probabilistically) independent if the realization of one does not have any impact on the probability of the other.*

Especially, for a newly arriving job the probability of the execution modes is independent from the execution mode of the jobs currently in the system or of previous jobs. We aim

Task-related Quantities		
$\tau_i = ((C_{i,1}, ..., C_{i,h}), D_i, T_i)$	Task τ_i and related WCETs $(C_{i,1}, ..., C_{i,h})$, deadline D_i, and period T_i	Sec. 2
$(C_{i,1}, ..., C_{i,h})$	WCET of the h different execution modes of τ_i	Sec. 2
$\mathbb{P}_i(j)$	Probability that a job of $\tau_i s$ is executed in mode j with related WCET $C_{i,j}$	Sec. 2
$\mathcal{M}$	Set of the possible execution modes (assumed identical for all tasks). $\lvert\mathcal{M}\rvert = h$	Sec. 2
$hp(\tau_k)$ and $hep(\tau_k)$	Tasks with higher priority than τ_k (higher and equal priority, respectively)	Sec. 2
$\rho_{i,t} = \lceil t/T_i \rceil$	Maximum number of jobs of τ_i released in an interval $[0, t)$ under the critical instant	Sec. 2
$J(t) = \sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil$	Total number of jobs released in the interval $[0, t)$	Sec. 5.1
S_t	Maximum accumulated workload over an interval of length t	Sec. 3.1
Probabilistic Quantities		
Φ_k	Probability of deadline miss for task τ_k	Sec. 3.1
$\mathbb{P}(S_t > t)$	Probability of overload for an interval of length t	Sec. 3.1
$\bar{X}$	Arithmetic mean of a random variable X	Sec. 4
$\mathbb{E}[X]$	Expected value of a random variable X	Sec. 4
$\mathbb{V}[X]$	Variance of a random variable X	Sec. 4
$\boldsymbol{X}(t)$	Random variable representing the possible execution modes of all jobs in $[0, t)$	Sec. 5.1
$\mathcal{X}(t)$	The state space of $\boldsymbol{X}(t)$ with $\mathcal{X}(t) = \mathcal{M}^{J(t)}$ since all jobs are considered	Sec. 5.1
$\boldsymbol{x} \in \mathcal{X}(t)$	One concrete variable assignment for $\boldsymbol{X}(t)$ over $[0, t)$	Sec. 5.1
$\mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x})$	Probability that the state space $\boldsymbol{X}(t)$ has the concrete variable assignment $\boldsymbol{x}$	Sec. 5.1
$\boldsymbol{X}_i(t)$	Subset of random variables in $\boldsymbol{X}(t))$ that relate to τ_i	Sec. 5.2
$C_i(\boldsymbol{X}_{i,j}(t))$	WCET for the j^{th} job of τ_i based on its random execution mode $\boldsymbol{X}_{i,j}(t)$	Sec. 5.1
Combinatorial Quantities		
$\mathbb{1}_{\{\text{expression}\}}$	Indicator function, i.e., evaluates to 1 iff the expression is true, and 0 otherwise	Sec. 5.1
$\sigma(\boldsymbol{x})$	A permutation of $\boldsymbol{x}$	Sec. 5.1
$\mathbb{S}_n$	Set of all permutations of length n	Sec. 5.1
$[\![\boldsymbol{x}]\!]$	Equivalence class of $\boldsymbol{x}$, i.e., all $\boldsymbol{x}' \in \mathcal{X}(t)$ that can be permuted into $\boldsymbol{x}$	Sec. 5.1

Table 1 Important notation used in this work. Please note that not all explanations in this table are precise. The precise notations can be found in the Section indicated in the table.

at relaxing the independence assumptions on tasks and jobs in future work by employing techniques from the field of probabilistic graphical models [**?**, **?**].

A list of our notation together with a brief explanation can be found in Table 1.

3 Motivation, Problem Definition, and State-of-the-Art

In this section, we will motivate the importance of the considered problem, i.e., the calculation of the probability of deadline misses, and formally define it. Afterwards, the state-of-the-art techniques are introduced, namely the traditional convolution-based approach by Maxim and Cucu-Grosjean [**?**] as well as the approach by Chen and Chen [**?**] that uses Chernoff bounds and the moment-generating function. We use the term *traditional convolution-based approach* when referring to the approach by Maxim and Cucu-Grosjean to avoid confusion, since our novel approach based on multinomial distributions also uses convolution.

3.1 Motivation and Problem Definition

One main assumption when considering real-time systems is that a deadline miss, i.e., a job that does not finish its execution before its deadline, will be disastrous and thus the WCET of each task is always considered during the analysis. Nevertheless, if a job has multiple distinct execution schemes, the WCETs of those schemes may differ largely. One example are software-based fault-recovery techniques as they rely on (at least partially) re-executing the faulty task instance. However, when such techniques are applied, the probability that a fault occurs and thus has to be corrected is very low; otherwise hardware-based faulty-recovery techniques would be applied. If such re-execution may happen multiple times, the resulting execution schemes have an increased related WCET while the probability decreases drastically. Therefore, considering solely the execution scheme with the largest WCET at design time would lead to largely over-designing the system resources. Furthermore, many real-time systems can tolerate a small number of deadline misses at runtime as long as these deadline misses do not happen too frequently. Hence, being able to predict the probability of a deadline miss is an important property when designing real-time systems. We will consider the probability of deadline misses for a single task here which is defined as follows:

▶ **Definition 2** (Probability of Deadline Misses). *Let $R_{k,j}$ be the response time of the j^{th} job of τ_k. The probability of deadline misses (DMP) of task τ_k, denoted by Φ_k, is an upper bound on the probability that a job of τ_k is not finished before its (relative) deadline D_k, i.e.,*

$$\Phi_k = \max_j \{\mathbb{P}(R_{k,j} > D_k)\}, \qquad j = 1, 2, 3, ... \tag{1}$$

It was shown in [?] that the DMP of a job is maximized when τ_k is released at its critical instant, i.e., together with a job of all higher priority tasks and all consecutive jobs of those higher priority tasks are released as early as possible. This implicitly assumes that no previous job has an overrun that interferes with the analyzed job. Hence, *time-demand analysis* (TDA) [?] can be applied to determine the worst-case response time of a task when the execution time of each job is known. TDA is an exact schedulability test for constrained and implicit deadline task sets with pseudo-polynomial runtime that, under the assumption that the schedulability of all higher priority tasks is already ensured, determines the schedulability of task τ_k by finding a point in time t where the total workload generated by tasks in $hep(\tau_k)$ is smaller than t. To be more precise: τ_k is schedulable if and only if

$$\exists\, t \text{ with } 0 < t \leq D_k \qquad \text{such that} \qquad S_t = C_k + \sum_{\tau_i \in hp(\tau_k)} \left\lceil \frac{t}{T_i} \right\rceil C_i \leq t \tag{2}$$

Thus, if $D_k \leq T_k$, task τ_k is schedulable if the statement $S_t \leq t$ is true. When probabilistic WCETs are considered, the WCET will obtain a value in $(C_{i,1}, ..., C_{i,h})$ with a certain probability $\mathbb{P}_i(j)$ for each job of each task τ_i. Therefore, for a given t we are not looking for a binary decision anymore. Instead, we are interested in the probability that the accumulated workload S_t over an interval of length t is at most t. The probability that τ_k cannot finish in this interval is denoted accordingly with $\mathbb{P}(S_t > t)$. We call the situation where S_t is larger than t an *overload* for an interval of length t and hence $\mathbb{P}(S_t > t)$ is the overload probability at time t. According to the previously introduced notation, $\rho_{i,t} = \lceil t/T_i \rceil$ for each task τ_i in $hp(\tau_k)$ and $\rho_{k,t} = 1$, i.e., only the first job of τ_k is considered here. Since TDA only needs to hold for one t with $0 < t \leq D_k$ to ensure that τ_k is schedulable, the probability that the test fails is upper bounded by the minimum probability among all time points at which the test could fail. Therefore, the probability of a deadline miss Φ_k can be upper bounded by

$$\Phi_k = \min_{0 < t \leq D_k} \mathbb{P}(S_t > t) \tag{3}$$

The number of points considered in Eq. (2) and therefore in Eq. (3) can be reduced by only considering the *points of interest*, i.e., D_k and the releases of higher priority tasks. Nevertheless, in the worst case this still leads to a pseudo-polynomial number of points. Since the minimum value among all these points is taken, an upper bound will still be obtained when only a subset of those points is considered. Two approaches to calculate Φ_k are known from the literature and are summarized in the following subsections.

In some cases it is easier to determine $\mathbb{P}(S_t \geq t)$ instead of $\mathbb{P}(S_t > t)$, especially when analytical bounds are used (see Sec. 3.3 and Sec. 4). Since $\mathbb{P}(S_t \geq t) \geq \mathbb{P}(S_t > t)$ by definition, these values can be used directly when looking for an upper bound of $\mathbb{P}(S_t > t)$.

3.2 Traditional Convolution-Based Approaches

Each task is defined by a vector of the possible WCETs and the related probabilities, e.g., $\begin{pmatrix} 3 & 5 \\ 0.9 & 0.1 \end{pmatrix}$ where 3 and 5 are the WCETs and 0.9 and 0.1 are the related probabilities. The notation we use is similar to the one used by Maxim and Cucu-Grosjean in [?]. The convolution of two such vectors is denoted by $\otimes$ and results in a new vector. To determine this new vector, each element of the first vector is combined with each element of the second vector by 1) multiplying the related probabilities, and 2) summing up the related WCETs.

▶ **Example 3** (Convolution). $\begin{pmatrix} 3 & 5 \\ 0.9 & 0.1 \end{pmatrix} \otimes \begin{pmatrix} 5 & 6 \\ 0.8 & 0.2 \end{pmatrix} = \begin{pmatrix} 8 & 9 & 10 & 11 \\ 0.72 & 0.18 & 0.09 & 0.01 \end{pmatrix}$

Note that the summation of the probabilities is 1 for each of these vectors. The general idea of the traditional convolution-based approach [?] is the direct enumeration of the WCET state space[1] and the related probabilities. To this end, it considers the jobs in non-decreasing order of their arrival times. For each arriving job, the current system state, represented by a vector of possible states, i.e., possible total WCETs and related probability, is convoluted with the arriving job. This results in a new vector of possible states, representing the state space after the arrival of the job. After all jobs released before a certain time point are convoluted, the probability that the workload is smaller than the next arrival time of a job is calculated. Afterwards, the jobs arriving at that time are convoluted with the current states, and the probability for the next arrival time is checked etc. This process is repeated until $t = D_k$ is reached. A small example explaining the approach considering two tasks can be found in Figure 1. The first jobs of τ_1 and τ_2 are both convoluted with the initial state and the four resulting states are each convoluted with the second release of τ_1 at $t = 8$. Obviously, when all jobs that are released up to any point in time are convoluted, states that result in the same execution time can be combined by adding up the related probability, e.g., the states with WCET 13 and 14, respectively, in Figure 1.

On one hand, applying the traditional convolution-based approach can easily lead to a state explosion where the number of states is exponential in the number of jobs. On the other hand, it calculates the exact probabilities for each t in the interval of interest in one iteration. To tackle the problem of state explosion, Maxim and Cucu-Grosjean introduced a re-sampling approach to reduce the number of states to a given threshold and thus to reduce the runtime while only slightly decreasing the precision as shown in [?].

3.3 Chernoff-Bound-Based Approaches

Chen and Chen [?] use the *moment generating function* (*mgf*) in combination with the *Chernoff bound* to over-estimate the deadline miss probability. We only briefly introduce the techniques here, i.e., describe how they can be used in our setting. Details can be found in, e.g., [?]. The *mgf* of a random variable is an alternative way to specify its probability distribution. For the specific case of the WCET distribution of a task τ_i the *mgf* is $\text{mgf}_i(s) = \sum_{j=1}^{h} \exp(C_{i,j} \cdot s) \cdot \mathbb{P}_i(j)$ where *exp* is the exponential function, i.e., $\exp(x) = e^x$, and $s > 0$ is a given real number.

[1] Please note that the approach in [?] does not only consider probabilistic WCETs but also probabilistic periods. Since we only consider probabilistic WCETs here, the approach is summarized accordingly.

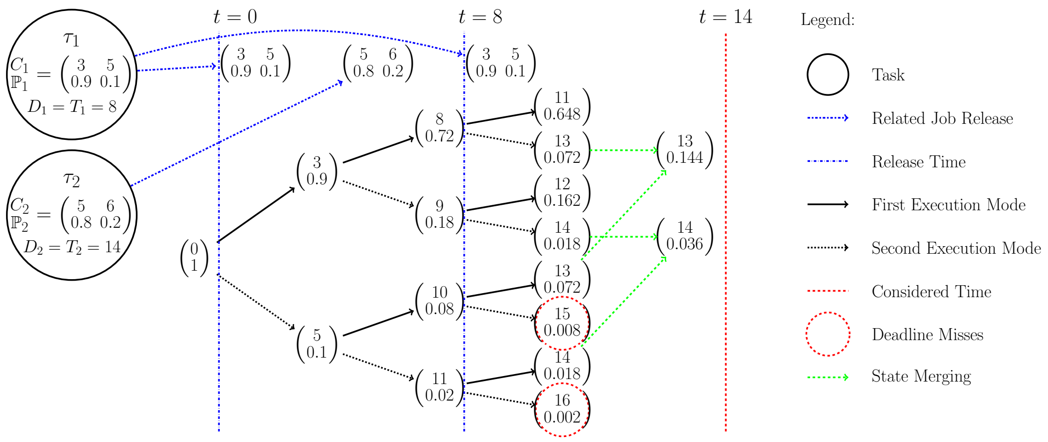

Figure 1 An example for the traditional convolution-based approach. Assume that $\mathbb{P}(S_{14} > 14)$ should be determined for two tasks τ_1 and τ_2. The initial state is convoluted with the two jobs released at $t = 0$ and the second job of τ_1 released at $t = 8$. Then, $\mathbb{P}(S_{14})$ is determined by summing up the probabilities of the states related to a workload larger than 14 (red dotted circle), leading to $\mathbb{P}(S_{14} > 14) = 0.01$. Note that states with the same execution time can be merged (dashed green arrows). This usually happens when the related paths are permutations of each other, e.g., both paths to 13 have one execution of $C_{1,1}$ and one of $C_{1,2}$.

The Chernoff bound can be exploited to over-approximate the probability that a random variable exceeds a given value. This statement is summarized in the following lemma:

▶ **Lemma 4** (Lemma 1 from Chen and Chen [?]). *Suppose that S_t is the sum of the execution times of the $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ jobs in $hep(\tau_k)$ at time t. In this case*

$$\mathbb{P}(S_t \geq t) \leq min_{s>0} \left(\frac{\prod_{\tau_i \in hep(\tau_k)} (mgf_i(s))^{\rho_{i,t}}}{\exp(s \cdot t)} \right) \tag{4}$$

The *Chernoff bound* is in general pessimistic and there is no guarantee for the quality of the approximation, even if the optimal value for s is known, i.e., the value that minimizes the right-hand side in Eq. (4). However, as the condition always holds, an upper bound can be obtained by taking the minimum over any number of s values. In contrast to the convolution-based approach, the evaluation of the right hand side of Eq. (4) is linear to the number of jobs in the interval of interest.

4 Analytical Upper Bounds

Concentration inequalities have various applications in machine-learning, statistics, and discrete-mathematics. Here, we show how some of them can be used to derive analytical bounds on $\mathbb{P}(S_t \geq t)$ which are easier to compute than the Chernoff bounds. Specifically, we will apply the Hoeffding's inequality [?] and Bernstein's inequality [?].

The *Hoeffding's inequality* derives the targeted probability that the sum of independent random variables exceeds a given value. For completeness, we present the original theorem here:

▶ **Theorem 5** (Theorem 2 from [?]). *Suppose that we are given M independent random variables, i.e., $X_1, X_2, \dots, X_M$. Let $S = \sum_{i=1}^{M} X_i$, $\bar{X} = S/M$ and $\mu = \mathbb{E}[\bar{X}] = \mathbb{E}[S/M]$. If*

$a_i \leq X_i \leq b_i$, $i = 1, 2, \ldots, M$, *then for* $s > 0$,

$$\mathbb{P}(\bar{X} - \mu \geq s) \leq \exp\left(-\frac{2M^2 s^2}{\sum_{i=1}^{M} (b_i - a_i)^2}\right) \tag{5}$$

Let $s' = sM$, *i.e,* $s = s'/M$. *Hoeffding's inequality can also be stated with respect to* S*:*

$$\mathbb{P}(S - \mathbb{E}[S] \geq s') \leq \exp\left(-\frac{2{s'}^2}{\sum_{i=1}^{M} (b_i - a_i)^2}\right) \tag{6}$$

By adopting Theorem 5, we can derive the probability that the sum of the execution times of the jobs in $hep(\tau_k)$ from time 0 to time t is no less than t:

▶ **Theorem 6.** *Let* a_i *be* $C_{i,1}$ *and* b_i *be* $C_{i,h}$. *Suppose that* S_t *is the sum of the execution times of the* $\rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ *jobs in* $hep(\tau_k)$ *released from time* 0 *to time* t. *Then,*

$$\mathbb{P}(S_t \geq t) \leq \begin{cases} \exp\left(-\frac{2(t-\mathbb{E}[S_t])^2}{\sum_{\tau_i \in hep(\tau_k)} (b_i - a_i)^2 \rho_{i,t}}\right) & \text{if } t - \mathbb{E}[S_t] > 0 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

where $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ *and* $\mathbb{E}[S_t] = \sum_{\tau_i \in hep(\tau_k)} (\sum_{j=1}^{h} C_{i,j} \mathbb{P}_i(j)) \cdot \rho_{i,t}$.

Proof. Since the execution time of a job of task τ_i is an independent random variable, there are in total $\rho_{i,t}$ independent random variables with the same distribution function upper bounded by $C_{i,h}$ and lower bounded by $C_{i,1}$ for each $\tau_i \in hep(\tau_k)$. With Eq. (6) and $s' = t - \mathbb{E}[S_t]$, we directly get:

$$\mathbb{P}(S_t \geq t) = \mathbb{P}(S_t - \mathbb{E}[S_t] \geq t - \mathbb{E}[S_t]) \leq \exp\left(-\frac{2(t - \mathbb{E}[S_t])^2}{\sum_{\tau_i \in hep(\tau_k)} (b_i - a_i)^2 \rho_{i,t}}\right) \tag{8}$$

when $s' > 0$. Otherwise, i.e., when $s' \leq 0$, we use the safe bound $\mathbb{P}(S_t \geq t) \leq 1$. ◀

The Chernoff bound and the related inequality by Hoeffding and Azuma can be generalized by the *Bernstein's inequality*. The original corollary is also stated here:

▶ **Theorem 7** (Corollary 7.31 from [?])**.** *Suppose that we are given* L *independent random variables, i.e.,* $X_1, X_2, \ldots, X_L$, *each with zero mean, such that* $|X_i| \leq K$ *almost surely for* $i = 1, 2, \ldots, L$ *and some constant* $K > 0$. *Let* $S = \sum_{i=1}^{L} X_i$. *Furthermore, assume that* $\mathbb{E}[X_i^2] \leq \theta_i^2$ *for a constant* $\theta_i > 0$. *Then for* $s > 0$,

$$\mathbb{P}(S \geq s) \leq \exp\left(-\frac{s^2/2}{\sum_{i=1}^{L} \theta_i^2 + Ks/3}\right) \tag{9}$$

The proof can be found in [?]. Note, however, that the result in [?] is stated for the two-sided inequality, i.e., as upper bound on $\mathbb{P}(|S| \geq s)$. Here, the one-sided result, which is a direct consequence of the proof in [?] (page 198), is tighter.

Hence, we can derive the following upper bound:

▶ **Theorem 8.** *Suppose that the sum of the execution times of all* $L = \rho_{k,t} + \sum_{\tau_i \in hp(\tau_k)} \rho_{i,t}$ *jobs is* S_t. *Let* $K = \max_{\tau_i \in hep(\tau_k)} C_{i,h} - \mathbb{E}[C_i]$ *be the centralized WCET of any job, where* $\mathbb{E}[C_i] = \sum_{j=1}^{h} \mathbb{P}_i(j) C_{i,j}$ *is the expected execution time of a job of task* τ_i. *Then,*

$$\mathbb{P}(S_t \geq t) \leq \begin{cases} \exp\left(-\frac{(t-\mathbb{E}[S_t])^2/2}{\sum_{\tau_i \in hep(\tau_k)} \mathbb{V}[C_i]\rho_{i,t} + K(t-\mathbb{E}[S_t])/3}\right) & \text{if } t - \mathbb{E}[S_t] > 0 \\ 1 & \text{otherwise} \end{cases} \tag{10}$$

for any $t > 0$, where $\rho_{i,t} = \left\lceil \frac{t}{T_i} \right\rceil$ and $\mathbb{E}[S_t] = \sum_{\tau_i \in hep(\tau_k)} (\sum_{j=1}^{h} C_{i,j}\mathbb{P}_i(j))\rho_{i,t}$.

Proof. Since for each task $\tau_i \in hep(\tau_k)$ the execution time of a job of task τ_i is an independent random variable, there are in total $\rho_{i,t}$ independent random variables with the same distribution function. Suppose that C_l is a random variable representing the execution time of a job of task τ_i and let $Y_l = C_l - \mathbb{E}[C_i] = C_l - \sum_{j=1}^{h} C_{i,j}\mathbb{P}_i(j)$ denote its centralized execution time. Since the expected execution time of a job is fully determined by its corresponding task, we have $\mathbb{E}[C_l] = \mathbb{E}[C_i]$.

Hereinafter, we explain why we adopt $\mathbb{V}[C_i]$ instead of θ_i^2 as known from Theorem 7. Consider Eq. (9) with $S = \sum_{l=1}^{M} Y_l$. The exact variance $\mathbb{V}[Y_l] = \mathbb{E}[Y_l^2] - \mathbb{E}[Y_l]^2 = \mathbb{E}[Y_l^2]$ is unknown and hence some loose upper bound θ^2 must be considered in most applications of Bernstein's inequality, like stated in Theorem 7. Here, the probabilities of the different execution modes are given numerically, i.e., $\mathbb{P}_i(j)$ for $C_{i,j}$. Hence, for an arbitrary but fixed task τ_i with h different execution modes, this results in

$$\mathbb{V}[Y_l] = \sum_{j=1}^{h} \mathbb{P}_i(j) \left(C_{i,j} - \mathbb{E}[C_i]\right)^2 = \sum_{j=1}^{h} \mathbb{P}_i(j) \left(C_{i,j}^2 - 2C_{i,j}\mathbb{E}[C_i] + \mathbb{E}[C_i]^2\right)$$

$$= \sum_{j=1}^{h} \mathbb{P}_i(j)C_{i,j}^2 - \sum_{j=1}^{h} \mathbb{P}_i(j)2C_{i,j}\mathbb{E}[C_i] + \sum_{j=1}^{h} \mathbb{P}_i(j)\mathbb{E}[C_i]^2 = \mathbb{E}[C_i^2] - \mathbb{E}[C_i]^2 = \mathbb{V}[C_i] \quad (11)$$

i.e., $\mathbb{V}[Y_l] = \mathbb{V}[C_i]$, which can be computed exactly in time $\mathcal{O}(h)$. Instead of imposing an upper bound θ^2, we can invoke the tightest version of Theorem 7 by using the exact variance.

Since $\mathbb{E}[Y_l] = 0$ and $\forall 1 \leq l \leq M : Y_l \leq K$, we can invoke Theorem 7 with $s = t - \mathbb{E}[S_t]$. When $s \leq 0$, we use a safe bound $\mathbb{P}(S_t \geq t) \leq 1$. When $s > 0$, Eq. (9) can be rewritten as

$$\mathbb{P}\left(\sum_{l=1}^{M} Y_l \geq t - \mathbb{E}[S_t]\right) \leq \exp\left(-\frac{(t - \mathbb{E}[S_t])^2/2}{\sum_{l=1}^{M} \mathbb{V}[Y_l] + K(t - \mathbb{E}[S_t])/3}\right) \quad (12)$$

Finally, observing that $\sum_{l=1}^{M} Y_l = S_t - \mathbb{E}[S_t]$ and $\sum_{l=1}^{M} \mathbb{V}[Y_l] = \sum_{\tau_i \in hep(\tau_k)} \mathbb{V}[C_i]\rho_{i,t}$ (from Eq. (11)) completes the proof. ◀

5 The Multinomial-Based Approach

In the traditional convolution-based approach [?], the underlying random variable represents the execution mode of each single job. First, we take a closer look on the related state space and show that the complexity of this approach depends on the specific definition of these random variables. Afterwards, we explain how this state space can be transformed into an equivalent space that describes the states on a task-based level by proving the invariance when considering equivalence classes for each task. As a result, we introduce our novel approach that is based on the multinomial distribution. The section is concluded with a short discussion regarding the complexity of our approach compared to the traditional convolution-based approach presented in Section 3.2.

5.1 The State Space of the Traditional Convolution-Based Approach

In this approach [?], $\boldsymbol{X}(t)$ is the set of the random variables representing the individual jobs released in the interval $[0, t)$ in the order of their arrival times. Note that the notion of $\boldsymbol{X}(t)$ instead of $\boldsymbol{X}$ is necessary, since the underlying state space and thus the underlying set of random variables are dependent on the considered t. Let $J(t)$ be the number of jobs released in $[0, t)$ under the critical instance of τ_k. Hence, $\boldsymbol{X}(t)$ represents a set of $J(t)$ independent random variables representing the execution modes of the individual tasks, i.e., $\boldsymbol{X}(t)$ is the Cartesian product over those $J(t)$ variables. To understand how the computation can be simplified, it is necessary to explicitly consider the random variables $\boldsymbol{X}(t)$ as well as the dependence between $\boldsymbol{X}(t)$ and the quantities S_t and C_i. To simplify notation, let us assume that all jobs have a common set of h execution modes $\mathcal{M}$, i.e., $|\mathcal{M}| = h$.[2] Thus, the state space of the random variable $\boldsymbol{X}(t)$ is $\mathcal{X}(t) = \mathcal{M}^{J(t)}$. A concrete assignment of these variables is denoted $\boldsymbol{x} \in \mathcal{X}(t)$, and the portion of $\boldsymbol{x}$ that corresponds to the jobs of task τ_i is denoted $\boldsymbol{x}_i$. Each task τ_i releases $\rho_{i,t} = \lceil t/T_i \rceil$ jobs, and thus $J(t) = \sum_{\tau_i \in hep(\tau_k)} \lceil t/T_i \rceil$. Hence, $\lceil t/T_i \rceil$ of the $J(t)$ random variables in $\boldsymbol{X}(t)$ are related to the task τ_i. Since the execution time of the j^{th} job of task τ_i depends on the related random variable $\boldsymbol{X}_{i,j}(t)$ we denote it $C_i(\boldsymbol{X}_{i,j}(t))$. Linking the total workload S_t to the random variables, from Eq. (2) we get:

$$S_t = S_t(\boldsymbol{X}(t)) = C_k(\boldsymbol{X}_{k,1}(t)) + \sum_{\tau_i \in hp(\tau_k)} \sum_{j=1}^{\rho_{i,t}} C_i(\boldsymbol{X}_{i,j}(t)) \tag{13}$$

Based on this, we denote the exact expression for the probability of a overload at time t as

$$\mathbb{P}(S_t(\boldsymbol{X}(t)) > t) = \sum_{\boldsymbol{x} \in \mathcal{X}(t)} \mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) \mathbb{1}_{\{S_t(\boldsymbol{x}) > t\}} \tag{14}$$

Here, $\mathbb{1}_{\{\text{expression}\}}$ is the *indicator function* which evaluates to 1 if and only if the expression is true, and to 0 otherwise. Since the execution modes of the jobs are assumed to be independent, the joint probability mass $\mathbb{P}(\boldsymbol{X}(t))$ factorizes over the jobs. The probability of each execution mode per job is fully determined by its corresponding task, and hence

$$\mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) = \prod_{\tau_i \in hp(\tau_k)} \prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j}(t)) \tag{15}$$

Each factor $\mathbb{P}_i(x)$ is the probability mass of any job of task τ_i, being in some state $x \in \mathcal{M}$. Note that Eq. (14) is exactly the quantity computed by the traditional convolution-based approach [?]. Hence, its stems from the state space $\mathcal{X}(t) = \mathcal{M}^{J(t)}$ that is exponential in the total number of jobs. Nevertheless, we leverage the independence of job modes to compute $\mathbb{P}(S_t(\boldsymbol{X}(t))) \geq t)$ over a different state space, which is the key insight of our method.

5.2 Invariance and Equivalence Classes

In Eq. (15), for any fixed task τ_i, the expression $\prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j})$ is determined by the number of jobs for each state in $\mathcal{M}$. As an example, consider an arbitrary task τ_i with two distinct execution states, i.e., $\mathcal{M} = \{C_{i,1}, C_{i,2}\}$, and suppose that $\boldsymbol{x}_i = (C_{i,1}, C_{i,2}, C_{i,1}, C_{i,2})$, $\boldsymbol{x}'_i = (C_{i,1}, C_{i,1}, C_{i,2}, C_{i,2})$, and $\boldsymbol{x}''_i = (C_{i,2}, C_{i,1}, C_{i,1}, C_{i,2})$. The resulting probability is identical in all three cases, i.e., $\mathbb{P}_i(\boldsymbol{x}_i) = \mathbb{P}_i(\boldsymbol{x}'_i) = \mathbb{P}_i(\boldsymbol{x}''_i)$. We formalize this property subsequently.

▶ **Lemma 9** (Probability Permutation Invariance). *Let τ_i be a task with a set of distinct execution modes $\mathcal{M}$, let $\rho_{i,t}$ be the number of jobs of τ_i released up to time t, and let*

[2] If a task has less than h (or even only one) execution modes, dummy modes with probability 0 can ensure this condition. Alternatively, $\mathcal{M}_i$ and h_i can be defined based on the execution modes of τ_i.

$\boldsymbol{x}_i \in \mathcal{M}^{\rho_{i,t}}$ be the random vector that represents the execution mode of all jobs which belong to task τ_i. The probability mass $\mathbb{P}_i$ is permutation invariant with respect to $\boldsymbol{x}_i$, i.e.,

$$\forall\ \boldsymbol{x}_i \in \mathcal{M}^{\rho_{i,t}} : \forall \sigma \in \mathbb{S}_{\rho_{i,t}} : \mathbb{P}_i(\boldsymbol{x}_i) = \mathbb{P}_i(\sigma(\boldsymbol{x}_i)) \tag{16}$$

where $\mathbb{S}_n$ contains all permutations of n objects.

Proof. The lemma follows directly from the independence of job-wise execution modes, thus $\mathbb{P}_i(\boldsymbol{x}_i) = \prod_{j=1}^{\rho_{i,t}} \mathbb{P}_i(\boldsymbol{x}_{i,j})$, and from the commutativity of the multiplication. ◀

Up to now, we considered just a single task τ_i, but the lemma indeed holds *for all* tasks simultaneously. Recall that the random modes of all tasks are represented by $\boldsymbol{X}(t)$. Let $\boldsymbol{X}_i(t)$ represent the random modes of the jobs of task τ_i, i.e., $\boldsymbol{X}_i(t)$ is the subset of random variables in $\boldsymbol{X}(t)$ that relate to the random modes of τ_i. Applying the permutation invariance to each $\boldsymbol{X}_i(t)$, we derive a partition on $\mathcal{X}(t)$ into equivalence classes.

▶ **Definition 10** (Execution Mode Equivalence Classes). For any $\boldsymbol{x} \in \mathcal{X}(t)$, its equivalence class $[\![\boldsymbol{x}]\!]$ with respect to permutation invariance is given by

$$[\![\boldsymbol{x}]\!] = \{\boldsymbol{x}' \in \mathcal{X}(t) \mid \forall \tau_i \in hep(\tau_k) : \exists \sigma \in \mathbb{S}_{\rho_{i,t}} : \boldsymbol{x}_i = \sigma(\boldsymbol{x}'_i)\} \tag{17}$$

Based on this definition, the statement $\forall \boldsymbol{x}' \in [\![\boldsymbol{x}]\!] : \mathbb{P}(\boldsymbol{x}) = \mathbb{P}(\boldsymbol{x}')$ is a straightforward corollary of Lemma 9. The equivalence relation in Lemma 10 is established by an equivalent occurrence of execution modes for each task. Hence, each equivalence class has a canonical representative, given by a tuple $\boldsymbol{\ell} \in \otimes_{\tau_i \in hep(\tau_k)}\{1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$, which for each task contains the number of jobs for all execution modes. For convenience we use $[\![\boldsymbol{\ell}]\!]$ to address the set of all $\boldsymbol{x}$ in the same equivalence class and rephrase Eq. (14) accordingly.

▶ **Lemma 11** (Class-based Overload Probability). *For any set of execution modes $\mathcal{M}$, let $\mathcal{L}(t) = \otimes_{\tau_i \in hep(\tau_k)}\{0, 1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$. Then,*

$$\mathbb{P}(S_t(\boldsymbol{X}(t)) \geq t) = \sum_{\boldsymbol{\ell} \in \mathcal{L}(t)} \prod_{\tau_i \in hep(\tau_k)} \frac{\rho_{i,t}! \prod_{j=1}^{|\mathcal{M}|} \mathbb{P}_i(j)^{\boldsymbol{\ell}_{i,j}}}{\prod_{x \in \mathcal{M}} \boldsymbol{\ell}_{i,x}!} \mathbb{1}_{\{S_t([\![\boldsymbol{\ell}]\!]) \geq t\}} \tag{18}$$

where $\boldsymbol{\ell}_{i,j}$ denotes the number of jobs of task τ_i which are in the j-th execution mode, and $S_t([\![\boldsymbol{\ell}]\!])$ denotes the execution time for some arbitrary $\boldsymbol{x} \in [\![\boldsymbol{\ell}]\!]$.

Proof. For all members of the class $[\![\boldsymbol{x}]\!]$, each task has the same number of jobs which are in the same state. Iterating over the set $\mathcal{L}(t) = \bigotimes_{\tau_i \in hep(\tau_k)}\{0, 1, 2, \ldots, \rho_{i,t}\}^{|\mathcal{M}|}$ corresponds to iterating over all such count vectors, which is in turn the same as iterating over all equivalence classes $[\![\boldsymbol{x}]\!]$. Each class $[\![\boldsymbol{\ell}]\!]$ contains all state permutations for all jobs of each task. For each task τ_i, this is equivalent to the well-known combinatorial problem of counting the number of ways how $\rho_{i,t}$ objects can be placed into $|\mathcal{M}|$ bins, given by the corresponding multinomial coefficient. Combining those for all tasks, we get

$$|[\![\boldsymbol{\ell}]\!]| = \prod_{\tau_i \in hep(\tau_k)} \binom{\rho_{i,t}}{\boldsymbol{\ell}_{i,1}\ \boldsymbol{\ell}_{i,2}\ \ldots\ \boldsymbol{\ell}_{i,|\mathcal{M}|}} = \prod_{\tau_i \in hep(\tau_k)} \frac{\rho_{i,t}!}{\prod_{x \in \mathcal{M}} \boldsymbol{\ell}_{i,x}!} \tag{19}$$

Combining these facts, we get

$$\sum_{\boldsymbol{x} \in \mathcal{X}(t)} \mathbb{P}(\boldsymbol{X}(t) = \boldsymbol{x}) = \sum_{\boldsymbol{\ell} \in \mathcal{L}(t)} |[\![\boldsymbol{\ell}]\!]| \mathbb{P}(\boldsymbol{X}(t) = [\![\boldsymbol{\ell}]\!]) \tag{20}$$

Observing that $\mathbb{P}(\boldsymbol{X}(t) = [\![\boldsymbol{\ell}]\!]) = \prod_{j=1}^{|\mathcal{M}|} \mathbb{P}_i(j)^{\boldsymbol{\ell}_{i,j}}$ implies the lemma. ◀

5.3 Detailing the Multinomial Approach

Now, we can combine the findings of Section 5.1 and Section 5.2 into an algorithm for calculating $\mathbb{P}(S_t > t)$, i.e., the probability of an overload for a length t, more efficiently. For simplicity of presentation, we will also refer to the overload probability *at time* t and the state space *at time* t, implicitly assuming that both the probability and the state space is calculated considering the interval $[0, t)$ with respect to the critical instant of τ_k. The traditional convolution-based approach determines this probability by successively calculating the probability for all other points of interest in the interval $[0, t)$. Nevertheless, the probability for t is evaluated based on the resulting states after all jobs in $[0, t)$ are convoluted. With respect to t, the intermediate states are not considered.

We utilize this insight to calculate the vector representing the possible states at time t more efficiently. Lemma 9 shows that the overload probability of a state for a concrete variable assignment $\boldsymbol{x} \in \mathcal{X}(t)$ is identical to the probability of all permutations of $\boldsymbol{x}$, i.e., the related equivalence class. This allows us to consider the jobs in $J(t)$ in any order. We further know from Lemma 11 that all assignments that are part of the same equivalence class result in the same value for S_t. Considering only one task τ_i, those assignments differ regarding the order in which the execution modes happen but not with respect to the total number of executions in a given mode. However, if the jobs are convoluted in the non-decreasing order of their arrival times, this leads to a large number of unnecessary states that will be merged in the end. For example, in Figure 1 the state space can be reduced if the second job of τ_1 would be convoluted before the job of τ_2 is convoluted, since the resulting merged state space after the convolution of the two jobs of τ_1 only has 3 states that represent the number of executions in each mode. Therefore, to reduce the state space as much as possible, we consider the jobs ordered according to the tasks they are related to, i.e., first all $\rho_{1,t}$ jobs of τ_1 are considered, then all $\rho_{2,t}$ jobs of τ_2, etc. However, if the jobs are just reordered and then convoluted, this still leads to a large number states that are merged later on.

Regardless, the number of states is already significantly lower than in the traditional convolution-based approach. Fortunately, if the number of jobs for a task is known, all possible combinations and the related probabilities can be calculated directly using the multinomial distribution. To be more precise, assume a given task τ_i as well as a given number of releases $\rho_{i,t}$ in an interval of length t and let $\boldsymbol{\ell}_{i,j}$ be the number executions in mode $j \in \{1, ..., h\}$. We know that $\boldsymbol{\ell}_{i,j} \in \{0, 1, ..., \rho_{i,t}\}$ and $\sum_{j=1}^{h} \boldsymbol{\ell}_{i,j} = \rho_{i,t}$, leading to $\binom{\rho_{i,t}+h-1}{h-1}$ possible combinations of $\boldsymbol{\ell}_{i,1}, ..., \boldsymbol{\ell}_{i,h}$ where $\binom{a}{b} = \frac{a!}{b!(a-b)!}$ is the binomial coefficient. For each combination, we can calculate the related probability as

$$\frac{\rho_{i,t}!}{\boldsymbol{\ell}_{i,1}!\boldsymbol{\ell}_{i,2}!...\boldsymbol{\ell}_{i,h}!}\mathbb{P}_i(1)^{\boldsymbol{\ell}_{i,1}} \cdot \mathbb{P}_i(2)^{\boldsymbol{\ell}_{i,2}} \cdot ... \cdot \mathbb{P}_i(h)^{\boldsymbol{\ell}_{i,h}} \tag{21}$$

where $\frac{\rho_{i,t}!}{\boldsymbol{\ell}_{i,1}!\boldsymbol{\ell}_{i,2}!...\boldsymbol{\ell}_{i,h}!}$ determines the number of possible paths for the related equivalence classes and $\mathbb{P}_i(1)^{\boldsymbol{\ell}_{i,1}} \cdot \mathbb{P}_i(2)^{\boldsymbol{\ell}_{i,2}} \cdot ... \cdot \mathbb{P}_i(h)^{\boldsymbol{\ell}_{i,h}}$ is the probability of one of these paths. The total workload of the $\rho_{i,t}$ jobs of τ_i is calculated for each of these combinations based on the related values of $\boldsymbol{\ell}_{i,1}$ to $\boldsymbol{\ell}_{i,h}$. The $\binom{\rho_{i,t}+h-1}{h-1}$ states represent the equivalence classes of τ_i and the related probabilities. After calculating these representatives for each task, the overload probability can be calculated by convoluting them and adding up the overload probabilities of the resulting state space. A concrete example for our approach, assuming that each task has two possible execution modes, is given in Figure 2. Details on how some equations can be simplified in this case can be found in the related full version [**?**]. Note that based on Lemma 9 the states representing the tasks can be convoluted in any order.

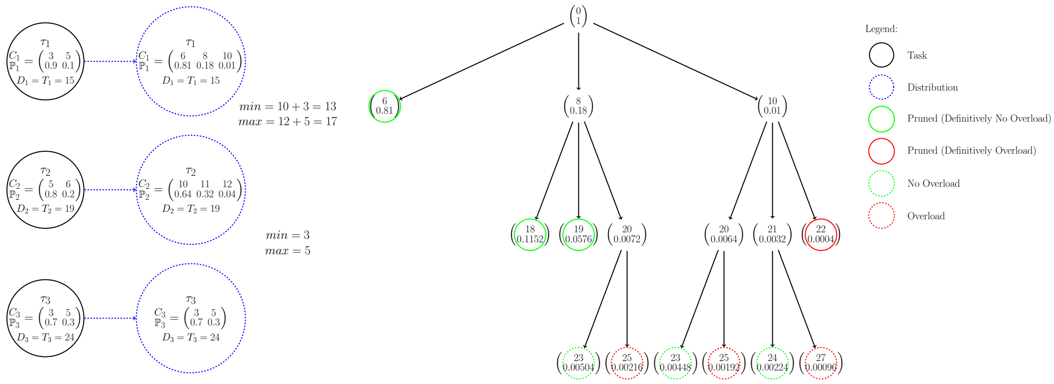

Figure 2 The multinomial approach convoluting 3 tasks with two modes. The number of children depends on the number of jobs of the related task. Note that nodes can be ignored in further steps if they never lead to an overload (green solid circles) or if they always lead to an overload (red solid circle). In the end, the overload probability at $t = 24$ is calculated by summing up the related probabilities (dashed and solid red) which leads to deadline miss probability of 0.00574.

In fact, considering t, the job-based state space of the traditional convolution-based approach has been transferred into a task-based space state with identical properties regarding the overload probability. To visualize the different approaches, the traditional convolution-based approach constructs a binary tree based on the jobs (see Figure 1) where each layer represents the state of the system after the related job is convoluted. The multinomial-based approach on the other hand constructs a tree based on the tasks (see Figure 2) which means that the number of children on each level depends on the number of jobs the related task releases. If the nodes on the $J(t)^{th}$ level of the binary tree are merged as show in Figure 1, the number of states on that level is identical to the number of states on the k^{th} level of the tree resulting from our approach. While the state space of our reformulation is still large, it opens up opportunities for pruning strategies and other state reduction strategies which are not suitable for the traditional approach. These strategies will be explained in Section 6.

5.4 Complexity Discussion and Comparison

When considering the complexity of the multinomial-based approach for τ_k over an interval $[0, t)$ (an interval of length t that ends at time t for notational brevity) under the critical instance of τ_k, both the number of tasks that are contributing to the workload in the interval, i.e., $\rho_{i,t}$ for the higher priority tasks, and the total number of jobs in the interval $J(t)$ have to be considered. The number of multinomial coefficients depends on $\rho_{i,t}$ and the number of possible execution states h for each task and can be calculated as $\binom{\rho_{i,t}+h-1}{h-1}$. This is also called the h-simplex of the $\rho_{i,t}^{th}$ component. The convolution of these states over all tasks leads to a total number of states of $\prod_{i=1}^{k} \binom{\rho_{i,t}+h-1}{h-1}$.

The classical convolution-based approach considers each job individually with h possible outcomes and, therefore, leads to $h^{J(t)}$ states, i.e., it is exponential in the number of jobs. Hence, without state merging, it is not feasible for input sets with a sensible cardinality. However, the convolution-based approach in the process also calculates the deadline miss probability at all possible points of interest in the interval, i.e., at each point in time a job is released. Furthermore, states can be merged when they have the same related workload, e.g., states resulting from a permutation of the same number of abnormal executions of a given task. Lemma 9 directly implies that when convolution is used in combination with merging states, the final number of states for the convolution-based approach at time t is identical to the number of states created by the multinomial-distribution-based approach (assuming that all states created by our approach lead to pairwise different workloads). However, while our approach creates only necessary states, the traditional convolution-based approach not only creates unnecessary states but also requires additional overhead for state merging after each step. Therefore, when considering a single point in time our approach is significantly faster than the traditional convolution-based approach with task merging. On the other hand, since our approach needs to consider all points of interest individually, if the number of such points increases due to the number of tasks the traditional convolution-based approach should be favoured. However, we were not able to observe this behaviour in our evaluation since both our multinomial-based approach as well as the traditional convolution-based approach with state merging only rarely were able to provide results for task sets with a cardinality of 10. Hence, for our approach runtime optimizations are provided in the next section. Note that this differs depending on the actual setting and that the period range is the most important parameter since it relates to the number of jobs.

6 Runtime Improvement

Here we introduce two strategies to improve the runtime efficiency. The first one prunes the state space, i.e., discards states directly if the impact on the overload probability can be determined without considering the remaining tasks, detailed in Section 6.1. This reduces the runtime without sacrificing any precision. The second technique combines execution mode equivalence classes with very low probability when creating the task representations to reduce the size of the state space beforehand, explained in Section 6.2. While this leads to an increase of the resulting overload probabilities, this error can be bounded for each task under consideration and therefore also with respect to the total error of the derived overload probability. Note that both techniques can be combined, which is done in the evaluation.

6.1 Pruning the State Space

Our multinomial-based approach calculates the probabilities for each interval individually, a property we already used when we transferred the state space from a job-based to a task-based state space. For convenience, assume that in our multinomial-based approach the representatives of the tasks are convoluted according to the task index. Recall that the state space can be seen as a rooted tree where each node on the j^{th} row represents a possible state after the convolution of the first j tasks and that we are only interested in the nodes on the k^{th} (and last) layer, i.e., the states after all task representations are convoluted. Such a tree is displayed in the example in Figure 2. The general concept of pruning is to remove a state R if the resulting subtree, i.e., the subtree with root R, has no further impact on the evaluation on the k^{th} layer, i.e., either *all* states on the k^{th} layer in the subtree with root R evaluate to an overload or for *all* states on the k^{th} layer in the subtree with root R the resulting workload is less than the interval length. In the first case, the state is discarded and the related probability is added to the overload probability considering t. In the second case, the state is directly discarded. This is done by checking the boundary conditions. To this end, for each task we determine the minimum and maximum execution time it can contribute to the total workload up to time t, respectively, which can be easily done while calculating the vectors that represent the task. On the i^{th} layer, the minimum and maximum workload that can be contributed by the remaining tasks, denoted as $C_{\min_i}$ and $C_{\max_i}$, is the sum of the minimum and maximum values related to the remaining tasks. Let $\mathbb{P}(\text{discard})$ be a variable accounting for the overload probability of discarded states, initialized with 0. For each state Q created by the convolution of τ_i with the previous state space let $C(Q)$ be the related total workload. We check the two following conditions:

1. $C(Q) + C_{\max_i} \leq t$: In this case the subtree rooted at Q only leads to states that will not lead to an overload at time t, since the branch related to the maximum cumulative workload in this subtree does not lead to an overload. Therefore, Q can directly be discarded. In the example in Figure 2 those states are marked with a solid green circle.
2. $C(Q) + C_{\min_i} > t$: All paths in the subtree rooted at Q result in an overload at time t, since the branch related to the minimum cumulative workload in this subtree leads to an overload. Hence, Q can directly be discarded and $\mathbb{P}(\text{discard})$ is increased by the probability of Q. In Figure 2 those states are marked with a solid red circle.

Obviously all created states can only fulfill one of these two conditions but not both due to $C(Q) + C_{\min_i} \leq C(Q) + C_{\max_i}$. If Q fulfills none, the state is added to the representation of $\tau_1, ..., \tau_i$. The correctness of this pruning approach follows directly from the observations that the total probability of a subtree on each level is equal to the probability of the root and from the fact that the total workload of each branch is always smaller than the maximum workload (larger than the minimum workload, respectively). A proof is therefore omitted. Note that the order in which the tasks are considered has no impact on the applicability of the pruning technique.

When considering a similar technique for the traditional convolution-based approach, one major difference is that the overload probability of all values is calculated successively. To be more precise, it considers the critical instant of τ_k at time 0 and the deadline miss probability for all intervals $[0, t)$, where t is the release time of a higher priority task. The interval $[0, D_k)$ is calculated successively and the result at time t_b depends on the result at time t_a if $t_a < t_b$. We visualize this by a rooted directed binary tree where each layer represents an arriving job and the layers are created according to the jobs arrival time, i.e., the height of the tree depends on the number of considered jobs (see Figure 1). The nodes on each layer represent the state space after the convolution of the related job. One important property of this approach is that the probability of deadline miss is calculated on each layer. Hence, pruning a state, i.e., removing a state and the branches resulting from it, can only be done if those branches have no impact on the probability on *all* following layers, i.e, a state R at time t_a can only be pruned if all branches of the subtree with root R will for all $t_b \in (t_a, D_k]$ either lead to an overload at t_b or to no overload at t_b. This cannot be determined by evaluating the overload condition for any single time point $t_b \in (t_a, D_k]$. Assume, for instance, for a $t_b \in (t_a, D_k]$ that $C(Q) + C_{\min_{t_b}} > t_b$ where $C_{\min_{t_b}}$ is the minimum workload created by jobs released in the interval $[t_a, t_b)$. Let t_{b-1} and t_{b+1} be the previous and next considered points with respect to t_b in the convolution based approach. We observe that τ_k may have no overload at t_{b-1}, if the minimum workload of the job released at t_{b-1} is smaller than $t_b - t_{b-1}$. Similar arguments can be taken to create a case with no overload at t_{b+1} and for the cases where τ_k has no overload at t_b if $C_{\max_{t_b}}$ is considered.

6.2 Union of Execution Mode Equivalence Classes

The general concept of the presented runtime improvement technique is to reduce the state space by unifying equivalence classes with low probability when creating the representation for the individual tasks. In contrast to the pruning technique, this obviously results in a loss of precision when approximating the deadline miss probability for a given point in time. However, if done carefully, the precision loss can be upper bounded by a constant. We will

# $C_{i,2}$ jobs	0	1	2	3	4	5	6	7	8	9	10
Total C_i	10	11	12	13	14	15	16	17	18	19	20
Probability	0.78	0.2	0.023	0.0016	$7.0 \cdot 10^{-05}$	$2.2 \cdot 10^{-06}$	$4.63 \cdot 10^{-08}$	$6.8 \cdot 10^{-10}$	$6.53 \cdot 10^{-12}$	$3.72 \cdot 10^{-14}$	$9.5 \cdot 10^{-17}$

# $C_{i,2}$ jobs	0	1	2	3	4	5	6 or 7		8, 9, or 10		
Total C_i	10	11	12	13	14	15	17		20		
Probability	0.78	0.2	0.023	0.0016	$7.0 \cdot 10^{-05}$	$2.2 \cdot 10^{-06}$	$4.701 \cdot 10^{-08}$		$6.564711 \cdot 10^{-12}$		

Table 2 Distribution for 10 releases of τ_i with $C_{i,1} = 1$, $C_{i,2} = 2$, $\mathbb{P}_i(1) = 0.975$, $\mathbb{P}_i(2) = 0.025$. The upper part details the distribution before and the lower part after merging equivalence classes.

introduce the concept based on the example in Table 2. Therein, we detail the release of 10 jobs in the interval of interest for a task τ_i with two execution modes that have a WCET of $C_{i,1} = 1$ and $C_{i,2} = 2$, with related probabilities $\mathbb{P}_i(1) = 0.975$ and $\mathbb{P}_i(2) = 0.025$. In the upper half, the original equivalence classes are displayed, i.e., one for each possible number of jobs (0 to 10), together with their total WCET and their (rounded) related probability. We will explain afterwards how the approach can be generalized.

The probability decreases rapidly with respect to the number of executions in the mode related to $C_{i,2}$. Such distributions are common when considering probabilistic execution times for real-time systems. The reason is that if the execution mode with larger WCET has a comparatively high probability, classical non-probabilistic worst-case response time analysis considering the larger WCET should be used to ensure timeliness for relatively common cases. Since the probability of the equivalence classes decreases, the impact of those classes on the overload probability over the given interval decreases as well. Therefore, the number of states that are created in our approach, and thus the runtime, can be reduced by unifying some of these highly unlikely equivalence classes. To guarantee a safe approximation, i.e., the resulting overload probability is only increased, we define the merge of a set of equivalence class as follows:

▶ **Definition 12** (Union of Task Equivalence Classes). *Let $\mathcal{C} = \{[\![\boldsymbol{x}_i]\!], [\![\boldsymbol{x}'_i]\!], [\![\boldsymbol{x}''_i]\!], \ldots\}$ be a set of $|\mathcal{C}| = q$ equivalence classes of task τ_i in a given interval of interest $[0, t)$. For each class $[\![\boldsymbol{x}_i]\!] \in \mathcal{C}$, let $\mathbb{P}_i([\![\boldsymbol{x}_i]\!])$ and $C_i([\![\boldsymbol{x}_i]\!])$ denote its probability and the related total worst-case execution time, respectively. Furthermore, let $[\![\boldsymbol{x}_i^{\max}]\!] \in \mathcal{C}$ be the equivalence class with the highest total WCET, i.e., $[\![\boldsymbol{x}_i^{\max}]\!] = \arg\max_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} C_i([\![\boldsymbol{x}_i]\!])$.*

When we union all classes in $\mathcal{C} = \{[\![\boldsymbol{x}_1]\!], ..., [\![\boldsymbol{x}_q]\!]\}$, the classes in $\mathcal{C}$ are replaced by a a new class $[\![\boldsymbol{x}_i^{\mathcal{C}}]\!] = \bigcup_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} [\![\boldsymbol{x}_i]\!]$ that has the following characteristics:

1. $C_i([\![\boldsymbol{x}_i^{\mathcal{C}}]\!]) = C_i([\![\boldsymbol{x}_i^{\max}]\!])$
2. $\mathbb{P}_i([\![\boldsymbol{x}_i^{\mathcal{C}}]\!]) = \sum_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} \mathbb{P}_i([\![\boldsymbol{x}_i]\!])$

As shown in Table 2, merging the equivalence classes for 6 and 7 executions of mode 2, the probability of the newly created class is the summation of their probabilities and the related WCET is the maximum among those two classes, i.e., the WCET of the class with 7 executions. We now show that merging a set of equivalence classes leads to a bounded error with respect to the overload probability.

▶ **Lemma 13** (Unifying Equivalence Classes Leads to a Bounded Maximum Error). *For task τ_i let $\mathcal{C} = \{[\![\boldsymbol{x}'_i]\!], [\![\boldsymbol{x}''_i]\!], \ldots\}$ be a set of $|\mathcal{C}| = q$ equivalence classes for the interval of interest $[0, t)$. If $\mathcal{C}$ is merged into $[\![\boldsymbol{x}_i^{\mathcal{C}}]\!]$ according to Definition 12, the probability of overload can only increase and the error is bounded by $(\sum_{[\![\boldsymbol{x}_i]\!] \in \mathcal{C}} |[\![\boldsymbol{x}_i]\!]| \mathbb{P}_i([\![\boldsymbol{x}_i]\!])) - |[\![\boldsymbol{x}_i^{\max}]\!]| \mathbb{P}_i([\![\boldsymbol{x}_i^{\max}]\!])$.*

This follows from Eq. (18), Eq. (20), and the fact that any $\mathcal{C}$ in which no class $[\![\boldsymbol{x}_i]\!]$ triggers the indicator function $\mathbb{1}_{\{S_t([\![\boldsymbol{x}]\!]) > t\}}$ does not introduce any error. Hence, if at least $[\![\boldsymbol{x}_i^{\max}]\!]$ triggers

$\mathbb{1}_{\{S_t([\![x]\!])>t\}}$ the maximum probability increase happens if all other classes did not trigger $\mathbb{1}_{\{S_t([\![x]\!])>t\}}$ before the unification but do afterwards. Since the process can be repeated for all tasks this directly leads to:

▶ **Theorem 14** (Bounded For The Overall Increase On The Overload Probability)**.** *If equivalence classes of tasks with respect to the interval $[0, t)$ are merged, the total increase of the overload probability for this interval is increased by the sum of the individual overload probability increase of the individually tasks.*

Now we can calculate the overloaded probability over $[0, t)$ with a bounded total error while reducing the states that have to be considered. Assume a value b for the allowed maximum error to be given and a set of n tasks. The maximum error is bounded by b if for each task the error is bounded by b/n. This can be achieved by ordering the related states in decreasing order of probability, traversing them in this order while summing up the probabilities of each state, and keeping all states until the summation is larger than $1 - b/n$. Afterwards the remaining states are unified into one.

So far we considered a setting similar to the one displayed in Table 2, i.e., the workload increases as the probability decreases. However, this is not necessarily the case, e.g., when a task has two execution modes with an equal probability or when a task has three execution modes and $C_{i,2}$ has the lowest probability. Nevertheless, in such cases the approach based on Theorem 14 can still directly be exploited since the union of equivalence classes is agnostic to the workloads and related probabilities as long as the total probability of the combined equivalence classes is less than b/n and thus the approach can directly be used. Hence, for a given task properties of the related distribution can be exploited in the process. For example, for two execution modes with identical probability the symmetry of the resulting distribution can be used if modes with a total probability of $b/2n$ at both ends are unified.

7 Evaluation

The main focus of our evaluation was to determine if our novel multinomial-based approach can provide good results in reasonable analysis runtime, especially considering the scalability with respect to the number of tasks for reasonable settings. To this end, for a given utilization U_{sum} and a number of tasks, we generated random implicit-deadline task sets with one execution mode according to the UUniFast method [**?**]. As suggested by Emberson et al. [**?**], the periods of those tasks were generated according to a log-uniform distribution with two orders of magnitude, i.e., $10ms - 1000ms$. We only considered tasks with two distinct execution modes in the evaluation, called normal and abnormal execution mode and hence $\mathcal{M} = \{N, A\}$. The normal execution mode is considered to have a (much) higher probability. The WCET in the normal mode was set according to the utilization, i.e., $C_{i,N} = U_i \cdot T_i$ and the WCET in abnormal mode was calculated as $C_{i,A} = f \cdot C_{i,N}$ for all tasks in the set.

We used a fixed setting, defined by U_{sum}, f, and $\mathbb{P}_i(A)$, tracking the resulting deadline miss probability and runtime related parameters. In each setting, the deadline miss probability for the lowest-priority task under the rate-monotonic scheduling approach was determined. In our evaluations, we considered the following approaches where the **bold** name indicates how the approach is referred to:

1. **Convolution:** The *traditional convolution-based approach* [**?**].
2. **Conv. Merge:** The *traditional convolution-based approach* [**?**] with state merging.
3. **Multinomial:** Our novel multinomial-based approach from Sec. 5.3.
4. **Pruning:** The approach in Sec. 5.3 combined with the pruning technique in Sec. 6.1.

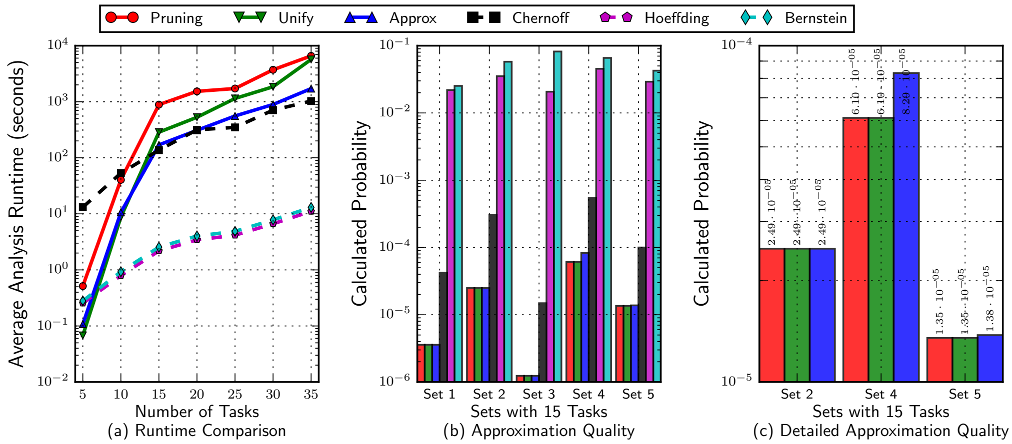

Figure 3 (a) Average runtime with respect to task set cardinality. (b) Approximation quality for 5 sets with 15 tasks. (c) Detailed approximation quality for the multinomial-based approaches.

5. **Unify:** The approach in Sec. 5.3 combined with the pruning technique in Sec. 6.1 and reducing the complexity with the union of equivalence classes presented in Sec. 6.2.
6. **Approx:** Approximation of **Pruning** by only considering the deadline of τ_k and the last releases of higher-priority tasks, inspired from the literature, e.g., [**?**, **?**, **?**, **?**].
7. **Chernoff:** The analytical approach using *Chernoff bounds* by Chen and Chen [**?**].
8. **Hoeffding:** The analytical approach using *Hoeffding's inequality* (Sec. 4).
9. **Bernstein:** The analytical approach using *Bernstein inequalities* (Sec. 4).

To allow runtime comparisons, all approaches were implemented in the same programming language, i.e., Python, and executed on the same machine, i.e., a 12 core Intel Xeon $X5650$ with 2.67 GHz and 20 GB RAM. For the analytical bounds, in contrast to the work by Chen and Chen [**?**], all releases of higher-priority tasks were considered since the bounds have a lower runtime than our novel approach.

Figure 3 shows the results for randomly generated tasks sets with a normal-mode utilization of $U_{sum} = 70$, and for all tasks $f = 2$ and $\mathbb{P}_i(A) = 0.025$ were assumed. Hence, $\mathbb{P}_i(N) = 0.975$. To analyze the scalability, the cardinality of the task sets ranged from 5 to 35 in steps of 5. In Figure 3(a) the average runtime of the analysis is displayed with respect to the cardinality. For a cardinality from 5 to 20 tasks, we evaluated 20 task sets while a cardinality from 25 to 35 tasks, due to the high runtime, 5 task sets were analyzed. For **Convolution** usually no result was delivered for a cardinality of 5, i.e., a crash due to an out of memory error occurred. Even for 3 tasks no result could be provided in some cases since, for instance, 38 jobs already leads to $2^{38} = 274877906944$ states for D_k in **Convolution**. For **Conv. Merge** and **Multinomial** a setting with 10 tasks often lead to no results. Hence, those three approaches are not displayed. However, the results for **Conv. Merge**, **Multinomial**, and **Pruning** were always identical (if **Conv Merg** and **Multinomial** derived results), showing that our pruning technique drastically decreases the runtime of the analysis and increases the scalability without any precision loss. We see that **Bernstein** and **Hoeffding** are orders of magnitude faster than the other approaches which are compatible with respect to the related runtime. The large runtime of **Chernoff** yields from finding a *good* s value in Eq. (4) which may differ for each point in time. The difference between **Approx** and **Pruning** stems from a different number of tested time points, i.e., for **Approx** this number depends on the number of tasks while for **Pruning** it is related to the number of jobs, while the calculation for one time point does not differ largely.

The statistical information of the derived deadline miss probabilities is unfortunately not meaningful. For example, for task sets with 15 tasks, the derived deadline miss probability in our evaluations under **Pruning** ranged from $3.0 \cdot 10^{-39}$ to $6.1 \cdot 10^{-5}$. Therefore, comparing the average values or other statistical means does not yield much information. In addition, comparing relative values is problematic if the probability gets low. Hence, we show a small sample of 5 task sets with roughly similar probabilities in Figure 3(b). These are the first 5 randomly generated task sets with deadline miss probability larger than 10^{-6}. This selection is only done to increase the readability of the figure. We observed in general similar relative behaviour among (nearly) all the evaluated task sets. We see that the error of **Bernstein** and **Hoeffding** is large compared to **Chernoff**, i.e., by several orders of magnitude, while the three approaches based on the multinomial distribution result in similar values, roughly one order of magnitude better than **Chernoff**. We also conducted experiments with different probabilistic distributions which in general lead to identical results.

In Figure 3(c), we compare the deadline miss probability of the three multinomial-distribution based approaches more closely. We can see that **Unify** performs very similar to **Pruning**, i.e., the error is in the magnitude of 10^{-9}. This is significantly smaller than the predefined *allowed error* of 10^{-6} for **Unify** in the experiments since: 1) execution mode equivalences classes are only merged for some of the tasks and the maximum error for each task may already be significantly smaller than 10^{-6}, and 2) the worst-case analysis in Sec. 6.2 is pessimistic. For **Approx** the error for Set 4 and Set 5 is in the magnitude of 10^{-5} and 10^{-7}, respectively, since only a subset of the points of interest is considered. In some rare cases even a larger relative difference could be observed.

Most importantly, all approaches we provide are able to deliver results even for large task sets, since the time needed to evaluate a single point in time remains still in the scale of minutes, i.e., in runs with 75 and 100 tasks one time point was evaluated on average in 621.6 and 791.1 seconds, respectively. Therefore, when a given task set needs to be analyzed, the approach can be used directly, especially since it is highly parallelizable due to the fact that different points in time can be analyzed completely individually. Hence, we suggest to first run *Hoeffding's* as well as *Bernstein's* bounds since they have a small runtime even for large task sets. If a sufficiently low deadline miss probability cannot be guaranteed from these bounds, we propose to run the multinomial-based approach with equivalence class union in parallel on multiple machines by partitioning the time points equally. We point out that it is especially helpful to use the union of equivalence classes if the periods of tasks differ largely, e.g., in automotive applications where periods often range from 1 to 1000 ms [**?**].

8 Conclusion

We provide a novel way to analyze the deadline miss probability of constrained-deadline sporadic soft real-time tasks on uniprocessor platforms where points in time are considered individually. Our main approach convolutes the equivalence classes of a task represented by the values of the multinomial distribution. The runtime of this approach can be improved by the detailed pruning technique without any precision loss. Furthermore, we present an approximation via unifying equivalent classes with a bounded loss of precision. In addition, we provide two analytical bounds based on the well-known Hoeffding's and Bernstein's inequalities which have polynomial runtime with respect to the number of considered time points. We demonstrate the effectiveness in the evaluations, specifically showing that our approaches scale reasonably even for large task sets.