2015 IEEE 18th International Symposium on Real-Time Distributed Computing

Flexible Framework for Statistical Schedulability Analysis of Probabilistic Sporadic Tasks

Abdeldjalil Boudjadar
Queen's University, Canada
jalil@cs.queensu.ca

Jin Hyun Kim
INRIA/IRISA Rennes, France
jin-hyun.kim@inria.fr

Alexandre David, Kim G. Larsen
Marius Mikučionis, Ulrik Nyman, Arne Skou
Aalborg University, Denmark
{adavid,kgl,marius,ulrik,ask}@cs.aau.dk

Insup Lee, Linh Thi Xuan Phan
University of Pennsylvania, USA
{lee,linhphan}@cis.upenn.edu

***Abstract*—The analysis of probabilistic schedulability explores all possible combinations of the probabilities of task attributes, which can easily lead to exponential computation time [24]. In this paper, we present a flexible schedulability analysis framework for periodic and sporadic tasks having probabilistic attributes where the computation time scales linearly in the size of analyzed systems. The framework is given in terms of a set of Parameterized Stopwatch Automata (PSA) models, which leads to a large degree of flexibility. Probability distributions for response time are generated using statistical model checking (UPPAAL SMC) while the overall schedulability can be checked using symbolic model checking (UPPAAL). We also define** PoMD **(*percentage of missed deadlines*) as a measure of the probabilistic schedulability of systems. To evaluate our approach, we compare the time used for computing response times and the analysis results using similar task models to that of a related analytical approach.**

I. Introduction

Limited resources are a strong factor in the system setting in many embedded software application fields. Engineers are interested, not only, in whether or not the system always meets its requirements, but also how it behaves with insufficient resources. Supplying a system with less resources than it requires may lead to a degradation of the quality of service. A certain level of degradation may be acceptable in a given setting and we thus consider it important to answer questions regarding schedulability with estimates of the quality instead of just providing a yes/no answer. Another argument for using probability based methods is that the classical safe analysis methods are very pessimistic.

In many areas such as real-time communication protocols, randomized distributed algorithms and dynamic power management, the timing attributes can vary greatly. Thus, it is advantageous to describe not only the worst case scenarios but also the probability of all potential scenarios. In this way, the timing attributes of tasks follow probability distributions. Different analytical approaches to schedulability under probabilistic conditions have been defined [24], [13], [14]. We pursue the analysis of probabilistic tasks in a setting of a model-based schedulability analysis framework for single core systems.

The research presented in this paper has been partially supported by EU Artemis Projects CRAFTERS and MBAT.

In [24], the authors introduce discrete probabilities for periods, execution times and deadlines of tasks. However, the framework in [24] can only handle fixed priority scheduling mechanisms. In this paper, we provide a model based framework to include any dynamic priority scheduling policy. In fact, we initiated the current work in [5] where we considered continuous probability distributions, but only for inter-arrival times, together with fixed and dynamic scheduling policies.

Contributions: In this paper, we present a flexible and scalable schedulability analysis framework for periodic and sporadic tasks having probabilistic attributes. Our framework can analyze both schedulability and response time as well as a new metric called Percentage of Missed Deadlines (PoMD) [5].

The framework is given in terms of a set of Parameterized Stopwatch Automata (PSA) models [8], which leads to a large degree of flexibility. This flexibility is obtained because each of the templates used in the framework can be changed independently. This means, for instance, that we can handle any scheduling algorithm simply by changing the scheduling template [4]. Probability distributions for response times are generated using statistical model checking (UPPAAL SMC) while the overall schedulability can be checked using symbolic model checking (UPPAAL). The analysis time of the statistical method presented in this paper scales linearly in all aspects of the analyzed systems: number of tasks, simulation time given as a number of triggerings, and the number of samples in the probability distributions.

Compared to the approach in [24], we provide a much more flexible and scalable framework. Thus, we can analyze the same types of models, but with any type of scheduling mechanism. We also show that we can estimate other properties of scheduling systems such as PoMD (*percentage of missed deadlines*). We also believe that model based approaches are easier than formula based methods to understand and update for engineers working with embedded real-time systems. Some of the advantages come from the ability to provide concrete system traces. Like the approach in [24], we do not consider the problem of obtaining samples, but assume that they are provided as input to our framework. Such samples will never be 100% exact, but it is reasonable to use statistical input to a flexible and scalable statistical method like the one presented in this paper.

Organization of this paper: Section II cites most relevant related work. Section III introduces the background of our work, for a statistical analysis method of probabilistic scheduling systems. In Section IV, we present our framework for the statistical schedulability analysis and introduce a metric that shows the schedulability in a probability perspective. Section V shows the analysis results of the response time distributions relying on the probabilistic task attributes, as well as comparisons of our work with the work in [24]. In Section VII, we conclude this paper.

II. Related Work

This work contributes to the analysis of probabilistic real-time systems using a model based framework.

In [14], Edgar and Burns argued for the use of probabilistic methods instead of only using WCET when analyzing the schedulability of an embedded system. In [1] Atlas and Bestavros provide and implement a rate monotonic scheduling approach where the task execution time is given by a probability density function. They analyze the system schedulability over superperiods. The superperiod of a task is given by the period of the next lower priority task. In [13], Diaz *et al* provide an analysis method for both fixed and dynamic priority scheduling policies where the execution times are given as probability distributions.

In [28], the authors propose a method to control the preemptive behavior of real-time sporadic task systems by the use of CPU frequency scaling. They introduced a new sporadic task model in which the task arrival may deviate, according to a discrete time probability distribution, from the minimum inter-arrival time. A similar approach is presented in [11]. The work of [24] extends the work in [13] by making all the task attributes probabilistic. The authors developed a piece of Matlab software that randomly generates a scheduling system given a number of samples for the probability distributions and a number of tasks, while the analysis can systematically be performed using the Matlab implementation of the underlying theory. However the aforementioned methodology does not handle dynamic scheduling policies. In [27], Santinelli *et al* define a framework for schedulability analysis of hierarchical scheduling systems on single core platforms. Two task parameters are characterized by discrete probabilistic functions; execution time and period. Their paper provides a theoretical framework, but not a practical implementation. Moreover, the complexity of the analysis presented in the paper is unknown. Compared to our work, [27] models similar types of systems but does not provide an implementation of the analysis.

In [7], Carnevali *et al* introduces a framework for deadline miss analysis in the context of flat fixed priority non-preemptive scheduling systems. The execution time is the only probabilistic parameter. The probability distribution is obtained using extreme value theory. If a job is not completed on time it will be immediately discarded. A simulation model for the framework has been implemented using stochastic time Petri nets. Compared to our work, the framework only considers the probability of missing a deadline, but not the response time nor the time by which a deadline is missed.

Other statistical modeling tools, such as PRISM [17] and PARAM [16], have also been applied to model the domain of probabilistic scheduling systems. In [25] the authors compare three different approaches for re-sampling discrete probability distributions characterizing task execution times. Re-sampling is used to combat the complexity when performing exact schedulability analysis on probability distributions.

The current work relies on [5] where we introduced a model based framework for the schedulability analysis of real-time systems having probabilistic execution times. Contrary to the current work, the probability distributions used in [5] are continuous. We follow the approach of [24] in the aspect of making all task attributes probabilistic.

III. Background

Classical analytical methods have aimed at giving 100% guarantees about the schedulability of a system. However, when the system inputs are statistical then either certainty cannot be obtained or the pessimism of the analysis will be extreme as only the worst cases are used. The framework we present can be used to obtain the desired confidence level, given by the number of simulated traces and the length of each trace. This paper introduces a framework for analyzing a soft real-time system and providing realistic response time distributions for each of the probabilistic tasks in the system. For scheduling systems, probabilities can be used to describe the timing attributes (period, execution time, ...) of tasks. So that each task attribute may have different exclusive values, each of which is associated with a probability stating how probable it is for the task attribute to be assigned this particular value. The method presented in this paper can be used to analyze a soft real-time component being part of a larger hierarchical scheduling system, following the method described in [4].

A. Probabilistic Tasks and Analysis

Analytical approaches for the response time and schedulability analysis of tasks having a probabilistic execution time have been defined in [1], [13] among many others. An extension has been proposed in [24] by making multiple task attributes probabilistic. In this setting, the response time is generated in terms of a probability distribution that can be compared to the tasks deadline.

A random variable X is given by:

$$X = \left(\begin{array}{c} x_1, ..., x_n \\ p_1, ..., p_n \end{array} \right) \tag{1}$$

where $\{x_1, .., x_n\}$ are outcomes (samples), $P(x_i) = p_i$ is the probability of each sample x_i to be selected and $\sum_{i=1}^{n} p_i = 1$. The probability of any variable x is given by $P(x)$ if $x \in \{x_1, .., x_n\}$, otherwise $P(x) = 0$.

We adopt the same scheduling system definitions as in [24]. A scheduling system is given by a set of n synchronous tasks $\{T_1, .., T_n\}$ to be scheduled on one CPU according to a scheduling policy. Each task $T_i = (I_i, E_i, D_i)$ is given by three timing attributes: I_i is the minimum inter-arrival time, E_i is the execution time and D_i represents the task deadline. Each of the timing attributes is a random discrete variable that has different values (samples) according to a probability distribution. In the original version, for the sake of simplicity the authors consider that I_i and D_i are always the same.

The computation of the response time of a task relies on the response time of its triggerings (jobs). However, while the task periods are not regular (different values according to the probability distribution), the number of triggerings is not constant and could be infinite. The notion of *hyperperiods*, for which the task arrival patterns are repeated for all hyperperiods has been considered in [13]. This implies that the analysis can be restricted to a single hyperperiod. The concept of hyperperiod is only applicable to settings where tasks have fixed length periods.

The computation of a response time R in [24] consists of exploring all possible combinations of the values of inter-arrival time and execution time of each task. Such a computation is iteratively performed up to a pseudo-hyperperiod (simulation time) given as a parameter for the analysis. The simulation time is specified by a multiple of the longest possible inter-arrival time of the lowest priority task. Similarly, we use the concept of simulation time as a pseudo-hyperperiod. Once the response time probability distribution is computed, the authors use the concept of `Deadline Miss Probability` (DMP) to determine whether or not the system is schedulable.

A necessary, but not sufficient, condition for schedulability is that all potential values of the response time distribution R are smaller than the possible values of the deadline distribution D. If this is not the case, a single triggering can occur where the deadline cannot possibly be met because the deadline is shorter than the response time.

Formally, the response time distribution R of the j^{th} triggering (job) of task T_i that is released at time $\lambda_{i,j}$ is given by $R_{i,j} = B_i(\lambda_{i,j}) \otimes C_i \otimes \mathcal{I}_i(\lambda_{i,j})$, where $\otimes$ is the *convolution* operator [24] which computes the combined probability distribution of two independent random variables. $B_i(\lambda_{i,j})$ is the accumulated backlog of higher priority tasks released before $\lambda_{i,j}$ and still running at time $\lambda_{i,j}$, $\mathcal{I}_i(\lambda_{i,j})$ is the sum of execution times of tasks having priority over task T_i and triggered after $\lambda_{i,j}$. The execution of the j^{th} job of T_i can be preempted for $\mathcal{I}_i(\lambda_{i,j})$ time units. Each of the variables of the aforementioned equation is a probability distribution.

In this paper we propose a model-based framework for the modeling, schedulability and response time analysis of probabilistic scheduling systems. Our framework enables to model the concrete behavior of tasks in terms of UPPAAL parameterized stopwatch automata [8], while it can check the system schedulability under any scheduling policy and estimate the response time using UPPAAL SMC tool. Methods based on statistical model checking (SMC) scale logarithmically in the size of the analyzed models; moreover they are trivially parallelizable and still scale sub-linearly [20], thus easily scaling to industrial size systems.

Our models are flexible in the way that can be easily updated to fit different systems and configurations. Moreover, they also provide other insurance measures for any engineering setting like the percentage of missed deadlines (PoMD).

B. Statistical Model Checking

Statistical Model Checking (SMC) is a simulation-based analysis approach used to give a probabilistic estimate of a certain property being satisfied by a given model. SMC [6] is a widely accepted analysis technique in many research areas such as industrial applications in software engineering [2], [23] and systems biology [9].

In order to estimate the probability of a property, SMC generates a number of stochastic runs and checks the property on each of the runs. The property is checked up to a certain confidence level (using confidence coefficient δ) and with a certain maximum error limit (ϵ distance from the center). Since many natural properties are monotone, the truth at length k of a run implies truth on the entire run [19], therefore we only check runs up to a certain bound of a run. In UPPAAL SMC the length of runs can be specified either as a number of discrete transitions or as a simulation time or cost bound. In our work we use a constant time bound $timeBound$. The confidence level, error limit and run length are all user parameters in our framework.

In theory, the maximum number of runs n required to achieve the needed confidence level δ and precision ϵ can be derived from Hoeffding's inequality $Pr(|\bar{p} - p| > \epsilon) \leq 2e^{-2n\epsilon^2}$ [18], which says that the probability of the wrong result (when the estimated $\bar{p}$ probability differs from the real probability p by more than ϵ) is no greater than $2e^{-2n\epsilon^2}$. The probability of the wrong result is called the *level of significance* $\alpha = 1 - \delta$, and hence $n \geq -ln(^{\alpha}/_{2})/(2\epsilon^2)$ runs is enough. Hoeffding's inequality implies that the number of runs is sub-linear in terms of confidence and quadratic in terms of precision. Moreover, the complexity does not depend on the structure of the model, but merely on the simulation performance. Therefore, it is not prohibitively expensive to get a very high degree of confidence even on the models which are prohibitively difficult to solve analytically. In practice, we can exploit the fact that our samples follow binomial distribution and hence the probability estimation is even more efficient by using sequential methods [15], which adapt to the actual probability value and the confidence interval is computed by more precise methods [10].

Besides the statistical check of property satisfaction, UPPAAL SMC can evaluate the modeled process performance by estimating the mean value of an expression over the model variables. In this case we cannot assume any distribution, hence the value estimation is based on the Central Limit Theorem which says that the distribution of means of sufficiently large samples follows Normal distribution, while the small sample means follow Student's t-distribution [26]. Therefore the confidence interval with level δ and significance $\alpha = 1-\delta$ is estimated using mean and t-distribution with standard error:

$$\frac{\Sigma_i^n x_i}{n} \pm t_{\alpha/2,n-1}\sqrt{\frac{\Sigma_i^n x_i^2 - (\Sigma_i^n x_i)^2/n}{n(n-1)}}$$

where x_i are the measured samples and $t_{\alpha/2,n-1}$ is the $\alpha/2$-quantile of t-distribution with $(n-1)$ degrees of freedom.

In order to estimate the mean of maximum (minimum) value over the run of an expression V, the following syntax is used: E[time<=TimeBound; RunCount] (max: V)

The size of the estimated interval depends on the variance of the measured samples, therefore there is no generic way to limit the error and hence the user has to specify the number of runs in the query (RunCount) while α is still the level of significance and confidence level is $\delta = 1-\alpha$. The confidence interval can be made arbitrary tight by increasing the number of runs. All experiments shown in this paper are performed as SMC estimation queries with 1,000 traces (RunCount) and a simulation time (TimeBound) of 100,000 time units. In order to have valid results we performed experiments where we analyze the same system with a varying amount of traces and simulation time. When reaching more than 1,000 traces and a simulation time of more than 100,000 time units we see that the estimated values are stable.

IV. Flexible Framework for Probabilistic Scheduling Systems

In this section, we present a statistical method to analyze the schedulability of sporadic tasks where the timing attributes are probabilistic. Following probability distributions, the schedulability may not be qualitative but also be quantitative. So that the probabilistic schedulability is said to be a probabilistic guarantee for the system schedulability. We study the probabilistic schedulability in terms of two metrics: 1) the response time probability distributions; 2) percentage of missed deadlines PoMD.

A. Probabilistic Schedulability

In contrast to the classical techniques of schedulability analysis, we do not only consider if a system is schedulable or not but we provide the *Percentage of Missed Deadlines* (PoMD) as a way to measure how schedulable a system is. PoMD can be computed for either a task or a complete embedded system. It must be measured or simulated over a sufficiently large time bounded run and a sufficiently large number of runs in order to obtain usable values. A run π of a system is an infinite sequence:

$$\pi = s_0(t_0,e_0)s_1(t_1,e_1)\dots s_n(t_n,e_n)\dots$$

where s_i is a global state giving information about the state of each task (e.g. idle, ready, running, blocked) and resource (e.g. idle, occupied) at stage i; s_0 is the initial state; e_i indicate events (triggering, completing or preempting tasks) taking place with t_i time-units separating e_{i-1} and e_i resulting in a transition from state s_i to s_{i+1}. We denote by Runs the set of runs of a system. For a run π and a time-bound $t \in \mathbb{R}_{\geq 0}$ we may define (in an obvious manner) the functions:

- $\mathsf{Miss}_t(T_i,\pi) \in \mathbb{N}$ is the total number of missed deadlines for task T_i up to time t;
- $\mathsf{Trig}_t(T_i,\pi) \in \mathbb{N}$ is the total number of triggerings of task T_i up to time t.

*Definition 1 (Percentage of Missed Deadlines (*PoMD*)):* The PoMD of an entity X for a run π is given by:

$$\mathsf{PoMD}^X(\pi) = (\limsup_{t\to\infty} \frac{\mathsf{Miss}_t(X,\pi)}{\mathsf{Trig}_t(X,\pi)}) \times 100\%$$

The entity X could be a task or a system. Now, the probabilistic arrival patterns of a set of tasks S give rise to a unique probability measure $\mathbb{P}_{\mathcal{S}}$ over (Runs, $\mathcal{B}$)[1] as such PoMD^T is a random variable. In order to estimate the expected value of PoMD^X, ePoMD^X, we generate a set Π of random (according to the stochastic semantics of S) and independent runs and calculate the mean using the following formula:

$$\mathsf{ePoMD}^X(\Pi) = \frac{\sum_{\pi\in\Pi}\mathsf{PoMD}^X(\pi)}{|\Pi|}$$

In fact, we estimate the ePoMD at the system level by simulating the complete system and summing up all triggering events and deadline misses. Our concept of PoMD is similar to the concept Deadline Miss Ratio (DMR) from [22].

B. Modeling of Scheduling Systems

The PSA models of our framework are; a sporadic task model, a task's attributes generator model depending on the attributes distributions, scheduling policy models and a CPU resource model. Moreover, two analyzer models are added for the analysis of response time and PoMD.

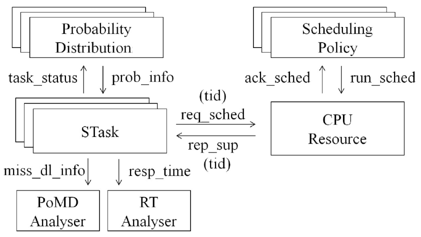

Fig. 1. Overview of PSA models composition

Fig. 1 gives and overview of the composition and communication between our PSA models. The execution of task process, **STask**, follows the probabilistic execution attributes *prob_info*. The execution attributes of a task are instantly fed by **Probability Distribution**, a probabilistic attribute generator, at each arrival point; and the beginning of a task execution is captured by an event *startSTask*. As soon as a task process starts an execution (job), it requests a CPU assignment to a resource manager process, **CPU Resource**, then the resource manager executes a scheduling function **Scheduling Policy** based on a specific scheduling policy. During this process, the two analyzers, **PoMD Analyzer** and **RT Analyzer**, track

[1]Here $\mathcal{B}$ is the standard Σ-algebra over Runs generated from a standard cylinder construction. For more see e.g. [12].

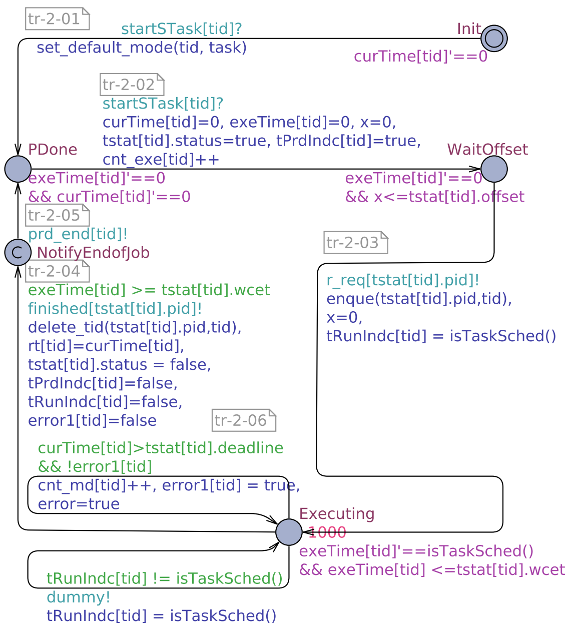
tr-2-01
startSTask[tid]?
set_default_mode(tid, task)
Init
curTime[tid]'==0
tr-2-02
startSTask[tid]?
curTime[tid]=0, exeTime[tid]=0, x=0,
tstat[tid].status=true, tPrdIndc[tid]=true,
cnt_exe[tid]++
PDone
WaitOffset
exeTime[tid]'==0
&& curTime[tid]'==0
exeTime[tid]'==0
&& x<=tstat[tid].offset
tr-2-05
prd_end[tid]!
NotifyEndofJob
tr-2-03
tr-2-04
exeTime[tid] >= tstat[tid].wcet
finished[tstat[tid].pid]!
delete_tid(tstat[tid].pid,tid),
rt[tid]=curTime[tid],
tstat[tid].status = false,
tPrdIndc[tid]=false,
tRunIndc[tid]=false,
error1[tid]=false
r_req[tstat[tid].pid]!
enque(tstat[tid].pid,tid),
x=0,
tRunIndc[tid] = isTaskSched()
tr-2-06
curTime[tid]>tstat[tid].deadline
&& !error1[tid]
cnt_md[tid]++, error1[tid] = true,
error=true
Executing
1000
exeTime[tid]'==isTaskSched()
&& exeTime[tid] <=tstat[tid].wcet
tRunIndc[tid] != isTaskSched()
dummy!
tRunIndc[tid] = isTaskSched()

Fig. 2. PSA model of sporadic tasks

timing information of each running task to compute PoMD and response time (RT).

In our PSA model, a clock can be used as a stopwatch that stops and resumes according to some conditions. Also, a clock can be used to store a double-typed value. When a clock is used as a double-typed variable, the progress rate of the clock is set to 0, and thus the clock behaves as a regular variable.

Fig. 2 shows the PSA task model as a UPPAAL template. Locations are represented by mauve colored circles, and the initial location has a double circle. The location names are written in red and locations can also have invariants. Edges are black arrows from source locations to destination locations. On an edge we have guards, synchronization events and updates. Exponential rates are used to give an exponential probability distribution for the waiting time in this location. Intuition; a high number indicates that there is a high probability of leaving very fast. The figure also contains tags that are not part of the model, but are used solely to make easier the description of the models elements.

The behavior of the task model, Fig. 2, is as follows:

- Init (location): A task process at this location waits for event startSTask[tid] fired by a probability attribute generator. Then, it sets the default execution attributes to actual task's execution attributes in tstat[tid] using user-defined function, set_default_mode() and moves to PDone.
- PDone (location): A task process leaves this location when it receives the startSTask and starts a new job. During transition to location WaitOffset, it resets all clocks and add one to cnt_exe[tid] for future PoMD computation.
- WaitOffset (location): A task process delays for the offset time. When moving to location Executing, it requires a CPU allocation from the resource manager pid through the event r_req[pid].
- Executing (location): A task process in this location performs the actual execution using a CPU. The execution may stop if the CPU is not available to the task. Two stopwatches, curTime[tid] and exeTime[tid], are running at this location. curTime[tid] measures the actual running time of a task while exeTime[tid] measures the running time of a task when the task can use a CPU. The function isTaskSched() checks the availability of CPU. Thus exeTime[tid] runs only if the function returns 1, otherwise it stops. But curTime[tid] keeps running, regardless of function isTaskSched(). When the current time reaches the deadline, the transition tr-2-06 must be taken to make the log of the error situation at a shared global variable error and increases the missed deadline counter cnt_md[tid] for PoMD computation. If the required execution time is fulfilled, the transition tr-2-04 outgoing from location Executing is taken with updating the clock rt[tid] with the running time curTime[tid] of a task for response time (RT) computation.

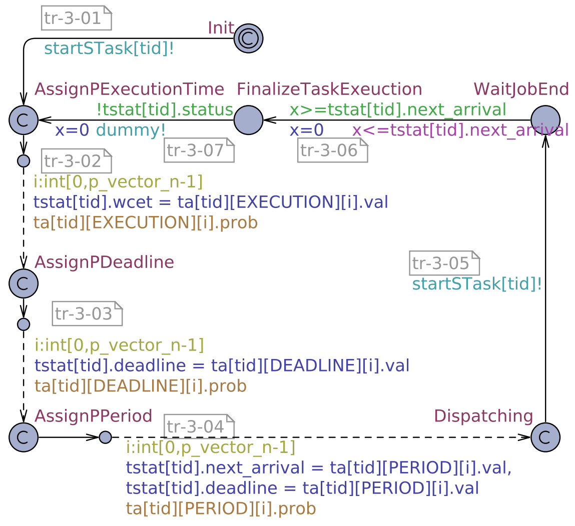

Fig. 3. PSA model for the assignment of probabilistic task attributes

Fig. 3 shows the PSA model of a probabilistic attribute generator, which assigns probabilistic execution attributes to actual task attributes. The PSA process behaves as follows:

- Init (location): The process leaves this location immediately and instantiates a task tid by sending the event startSTask[tid] on transition tr-3-01.
- tr-3-02, tr-3-03, tr-3-04 (transition): The process assigns probabilistic execution attributes, e.g. ta[tid].[EXECUTION][i].val, to the associated task execution attributes, e.g tstat[tid].wcet, according to discrete probability distributions, e.g. ta[tid].[EXECUTION][i].prob.
- tr-3-05 (transitions): The process initiates a task's new job by triggering the event startTTask[tid].
- WaitJobEnd (location): The process at this location waits for the job end, i.e. x>=tstat[tid].next_arrival, then it joins

the location FinalizeTaskExecution.

- FinalizeTaskExecution (location): In the case where a task misses the deadline and also the next arrival, the start of a new job must be delayed. Thus the process residing at this location checks whether the task completes the execution. If the task is still running, i.e !tstat[tid].status, the process postpones the release of a new job until the end of the current job.

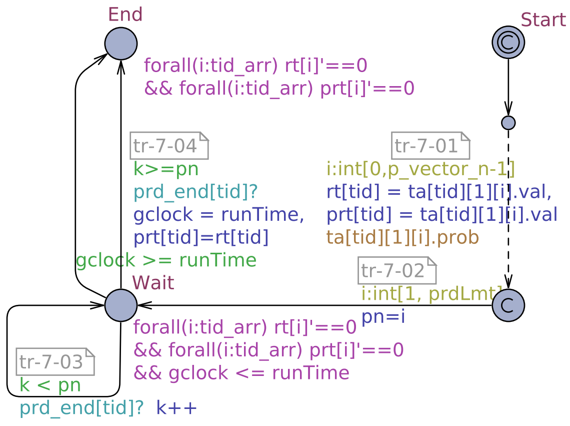

Fig. 4. PSA model of the response time analyzer

Fig. 4 and Fig. 5 show two PSA templates used for the analysis of RT and PoMD, respectively. The RT analyzer in Fig. 4 uses the following variables:

- gclock (clock): A global clock.
- prt[tid] (clock): A stopwatch to store the response time rt[tid] at the end of a single run of a task. It is used to generate the RT distribution.
- pn (int): The random periods of a task to be performed for stochastic task execution.
- prd_end[tid] (channel): The RT analyzer is reported via this channel about the end of each period of a task.

The RT analyzer in Fig. 4 behaves as follows:

- tr-7-01 (transition): The process initializes rt[tid] with the execution time of task tid according to the probability distribution.
- tr-7-02 (transition): The process stochastically selects the number of periods but less than the maximum number of periods for a task to execute for the statical analysis of the response time.
- Wait (location): The process waits the end of an execution period reported via the channel prd_end[tid]. If the number of periods is fulfilled, it finalizes a single run of a task by assigning the final response time rt[tid] to prt[tid].

The PoMD analyzer in Fig. 5 calculates a PoMD at the end of a single run of a task. It behaves as follows:

- tr-8-01 (transition): If the process reaches the end of a run, it moves to location CalPoMD and starts the calculation of PoMD for each task using the equation on the transition tr-8-04 (transition).

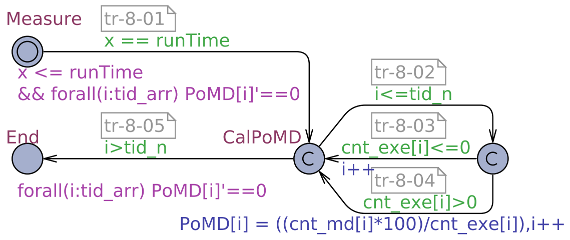

Fig. 5. PSA model of PoMD analyzer

PoMD analyzer takes one loop over the committed location, in the right hand side, for each of the tasks (i <= tid_n) in the system. The transition tr-8-04 calculates the PoMD for one task at a time. Transition tr-8-03 is only used if no deadline was missed by the current task, in order to avoid division by zero. When all PoMDs have been calculated (i > tid_n) the transition tr-8-05 is triggered moving to the End location.

V. Probabilistic Analysis and Comparison

In this section, we present our analysis method and compare our results, given in terms of response time distributions and the time used to compute the response time distributions, to the results obtained using the method from [24].

In the different graphs and tables presented in this section results are named after the tools used to obtain them, thus UPPAAL for the method of this paper and Matlab for the method presented in [24]. Each instance that we analyze is identified by two numbers. The first number indicates the number of tasks that are in the task set, the second number indicates how many multiples of the maximal inter-arrival time (abbreviated $maxATime$) for the tasks that are analyzed.

For all the experiments we present, in the first part we have fixed the number of samples in the probability distributions for the execution time, inter-arrival time and deadline to four. Three different task sets were randomly generated using the random task set generator provided by the Matlab implementation. The exact same task sets were then analyzed with the two different methods.

A. Our Analysis Method

Based on the models presented in the previous section, we explore the potential executions of the system using Statistical Model Checking (UPPAAL SMC). We also use Symbolic Model Checking (UPPAAL) to check the schedulability of the systems. The computation times for these results are presented at the end of this section.

UPPAAL SMC allows different types of queries for analyzing statistical properties of a system. In the current analysis we use the following estimation query:

$$E[<= runTime; runCnt](max : prt[tid]) \qquad (2)$$

$runTime$ is the time bound of each single run (trace), $runCnt$ is the number of runs generated in order to perform the statistical analysis, and $prt[tid]$ is the clock variable tracking the response time of the task tid for each triggering (job). Each generated trace has the same maximum number of triggerings.

UPPAAL SMC stochastically selects one of the prt values generated over the same task trace, and then performs a statistical analysis over the selected prt values of all ($runCnt$) traces. So that the query computes the response time distribution of the given task tid over the set of traces $runCnt$. In our setting, max does not have any real impact, since the $prt[tid]$ is assigned only one value during the individual trace. But it is still needed to respect the UPPAAL SMC query syntax. We use a simple auxiliary PSA process to compute the PoMD.

All the models and analysis results are available at http://people.cs.aau.dk/~ulrik/submissions/340472/ISORC2015.zip.

B. Comparison of the Response Time Analysis

In the following, we first compare the computation times of the two methods. In the next subsection we present and compare the probability distributions that the methods produce for the response time distributions. All the graphs and the raw data obtained from the experiments are also included in the above mentioned zip-file.

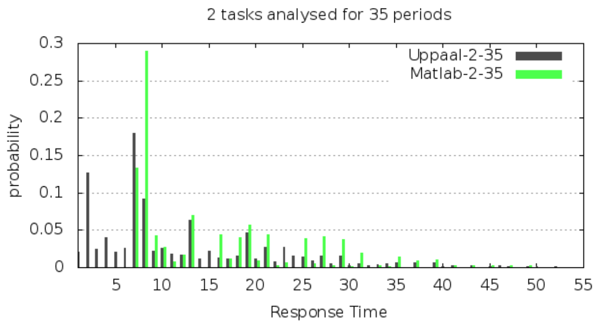

Fig. 6. Probability distributions of response time for a set of 2 tasks.

As shown in Fig. 6, the two methods do not produce the same response time distributions for a set of 2 tasks analyzed for 35 $maxATime$. Again, we recall that in all of our figures the legend includes numbers such as "Uppaal-2-35", the first number "2" refers to the number of tasks and the second number "35" refers to the number of job triggerings. That legend is also the name of the specific data file used for the plot, which can be found in the referenced zip file. The first obvious difference in Fig. 6 is that the UPPAAL method produces response times below 7 while this is not the case for the Matlab method on this specific set of tasks. This can easily be explained as the Matlab method only analyzes the cases where the task under analysis has always been preempted by a higher priority task. In this task set, the shortest possible response time for the lower priority task is exactly 7. This number is in fact the sum of the shortest execution time (1) of the lower priority task and the longest execution time (6) of the task having higher priority. In contrast, our analysis technique considers also the cases where the task under analysis is not preempted by any other task. So that the shortest possible response time of a task is the same as its shortest execution time (1).

The analysis method presented in [24] only considers cases where the lowest priority task has always been preempted by a higher priority task, i.e. it is a kind of probability distribution of the response time in *almost* worst cases. For the computation of response time, the authors consider only the case where all tasks are running simultaneously, so that the lowest priority task is assumed never to be triggered alone (best scenario). Our analysis method, on the other hand, produces the probability distribution of the response time for all possible scenarios where the lowest priority task has just been triggered.

When analyzing a task set for different multiples of $maxATime$ with the same method we can see that the analysis results keep changing. The authors of [24] do not even consider analyzing more than one job ahead and do not treat this in their discussion. However their Matlab implementation facilitates calculation for more than one triggering, which we have used. Our experiments show that their calculated distributions remain quite stable after three triggerings, as can also be graphically seen from Fig. 7. From this we conclude that for a lot of cases it could be sufficient to compute the response time distributions only for a few triggerings using the Matlab implementation.

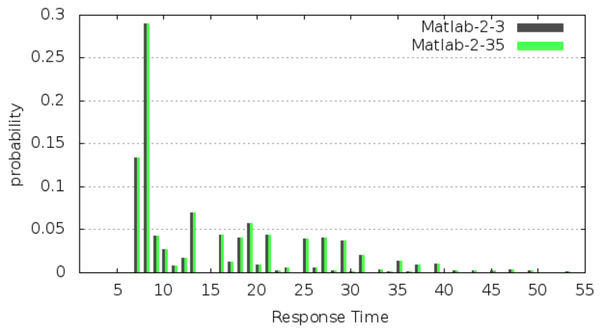

Fig. 7. Probability distributions of response time for 2 tasks analyzed with 2 different multiples of $maxATime$ using Matlab.

When analyzing the same task set with two different multiples of $maxATime$ using the UPPAAL method, the response time probabilities vary more as can be seen in Fig. 8. On the other hand, we would not normally have chosen to use such a low multiple of $maxATime$ for the analysis. This is only done to better compare with the Matlab based method.

C. Comparison of the Computation Time

In this section, we present and compare the time used for computing the response time distributions. In fact, using the Matlab implementation we randomly generate three different task sets of 2 tasks, 3 tasks and 4 tasks respectively. All task execution attributes follow probability distributions of 4 samples. The three different task sets have been analyzed using the two methods for a varying number of triggerings. All computation times in MATLAB are performed without re-sampling. The provided software package had re-sampling disabled because enabling it leads to a software bug.

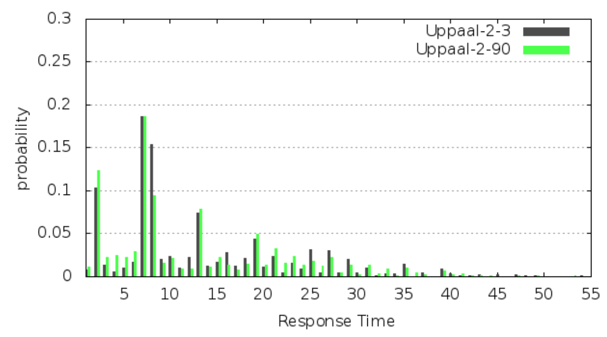

Fig. 8. Probability distributions of response time for 2 tasks analyzed with 2 different multiples of $maxATime$ using Uppaal.

TABLE I
TIME USED FOR COMPUTING THE RESPONSE TIME DISTRIBUTIONS

#$maxATime$	Matlab 2-Task / Uppaal 2-Task	Matlab 3-Task / Uppaal 3-Task	Matlab 4-Task / Uppaal 4-Task
3	0.0624	0.3432	1.4976
	1.355	3.844	4.437
7	0.3744	5.3508	21.6997
	2.963	4.005	13.392
15	3.1668	69.202	254.078
	6.733	6.976	19.237
35	40.7943	1315.5	5053.4
	14.122	19.724	38.92
90	**1152.2**	**39501.**	**168500.**
	30.603	**40.498**	**86.095**

Table I presents the computation times for all of these experiments. The first six data points of the second column (experiment for 2 tasks) of Table I are presented in Figure 9 in order to illustrate that the computation time of our method grows linearly in the number of $maxATime$ that we analyze, while the Matlab method grows exponentially with regard to the number of triggerings that are being analyzed. For the experiments with 90 $maxATime$, Table I shows huge differences between the results obtained by both analysis methods.

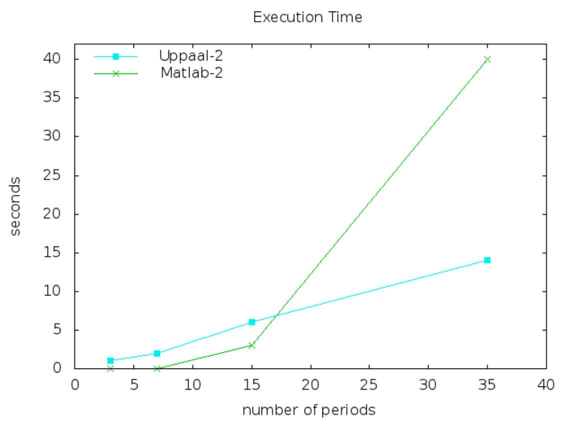

Fig. 9. Computation time for the 2-task set with different #$maxATime$.

Since we have concluded that the Matlab method stabilizes after only a few triggerings of look ahead, we here choose to analyze the computation time for different numbers of samples in the probability distributions. We run both Matlab and UPPAAL analysis for a set of 4 tasks with **7** multiples of $maxATime$, while we vary the number of samples from 2 to 5.

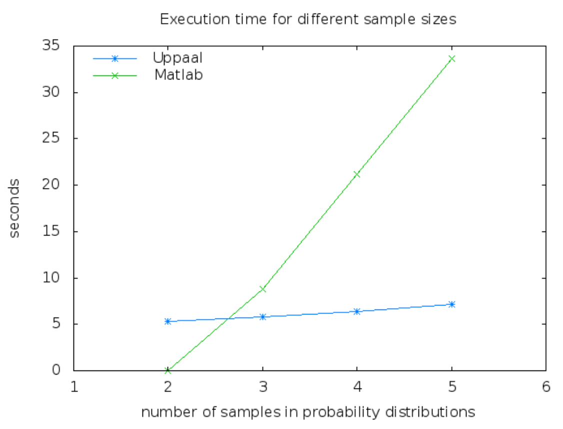

Fig. 10. Computation time for different sample sizes of the probability distributions.

Fig. 10 presents the results of such experiments, where the computation time of Matlab grows with a higher rate compared to our UPPAAL analysis in the number of samples.

As stated in the introduction, our method is very flexible and can just as easily handle dynamic scheduling policies such as EDF. Fig. 11 shows the response time probability distributions for the same task set using two different scheduling algorithms. Only one of the PSA models has to be exchanged in order to change the scheduling policy.

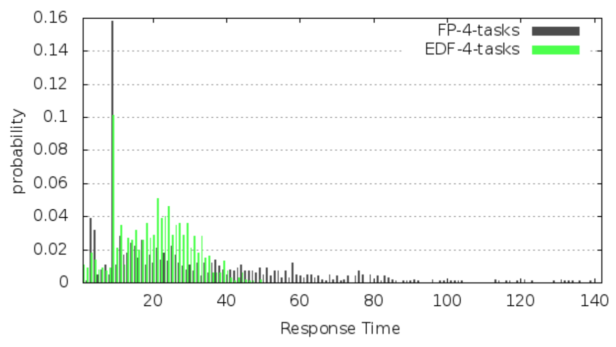

Fig. 11. Response time distributions for identical task sets with 2 different scheduling policies.

TABLE II
TIME AND MEMORY USAGE OF SMC ANALYSIS (10 TASKS, 8 SAMPLES)

runTime / runCnt	Time	Memory
100,000 / 10	11 s	35,632 KB
100,000 / 100	115 s	33,132 KB
100,000 / 1000	1,039 s	30,632 KB

Table II shows the computation time and memory usage for the analysis of 10 tasks with 8 samples. We have fixed the $runTime$ to 100,000 while we vary the number of runs ($runCnt$). This further illustrates, as stated in the introduction, the well known fact that Statistical Model Checking scales to industrial size systems [20].

D. PoMD Computation

PoMD can be used to answer the question; how schedulable is the system? It is a statistical metric that conceptually has the same meaning as DMP [24] but computed differently.

TABLE III
PoMD ANALYSIS FOR 4 TASKS WITH 4 SAMPLES

#$maxATime$	PoMD	Time
3	36.368	1.3s
7	31.388	2.48s
15	30.000	5.1s
35	29.535	11.34s
90	29.361	29.35s
1000	29.216	301.53s

Table III shows the PoMD we computed for a set of 4 tasks having 4 samples for each probability distribution, while we vary the multiple of $maxATime$ in the analysis. One can see that the PoMD becomes stable from a certain point (35 time $maxATime$) of the analysis duration.

E. Analysis Using Symbolic Model Checking

For more safety, in the case of hard real-time, it would be necessary to also check whether a given task set is schedulable or not according to a classical definition [21]. The main point of the method presented in this paper is to handle soft real-time systems. For the purpose of hard or mixed criticality systems, we can use symbolic model checking on the same models as we use for the statistical analysis [3]. Below we present the verification times we obtained using the symbolic model checker UPPAAL for different task sets. Notice that the

TABLE IV
SYMBOLIC MODEL CHECKING RESULTS (TASK SET = ((BCET,WCET), (MIN_PRD,MAX_PRD), DEADLINE))

Tasks	Task Set	Results	Time	Memory
2	((6, 6),(9,13),9)	Schedulable	0.01s	13,952KB
	((1,2),(14,18),14)			
3	((5,6),(16,17),12)	Schedulable	0.01s	22,820KB
	((2,2),(14,25),14)			
	((27,29),(1,2),20)			
4	((2,6),(20,21),20)	Schedulable	108.14s	36,692KB
	((3,4),(25,26),20)			
	((2,2),(26,34),25)			
	((1,3),(36,39),31)			
4	((2,6),(17,21),20)	Unschedulable	1.31s	45,388KB
	((3,4),(25,26),20)			
	((2,2),(26,34),25)			
	((1,3),(36,39),31)			

tasks' attributes have a range of values, i.e. bcet and wcet, and minimum and maximum inter-arrival time. UPPAAL analyzes the same models, but does it differently. The uncertainty is handled as non-determinism. As shown in Table IV each task attribute is given by lower and upper bounds. For each task attribute UPPAAL non-deterministically selects one value for each period, and analyze the state space obtained by all combinations. Thus, UPPAAL provides a qualitative answer, **yes** or **no**, for the schedulability property.

VI. DISCUSSION AND EVALUATION

In this section we compare drawbacks and advantages of our work. As other simulation based approaches, ours can provide concrete traces for specific scenarios. We find that the use of concrete traces makes the method easier to understand for software engineers. Our method can also be updated to reflect the particularities of a given platform, something that is impossible with an analytical approach that only abstracts one generic task behavior.

We have performed a simple form of sensitivity analysis [29] in Section V-D, when examining how the simulation time (hyperperiod) affects the output. Further investigation into the sensitivity of the model based on the different input parameters could be future work.

Compared to classical analytical approaches, our method has the drawback that it is a simulation based approach. Thus, there can be cases that are very unlikely but possible and that maybe do not show up in the statistical analysis. For medium sized systems, we can prove the schedulability using the same models and symbolic model checking as shown in Table IV.

Since [24] only consider fixed priorities, when computing the response time (R) distribution of a task all lower priority tasks can be ignored. For all shown examples the authors compute the response time distributions of the lowest priority tasks, but can also calculate the response time for tasks other than the lowest priority task. When analyzing a task with priority n where n is not the lowest priority any task with priority lower than n will be ignored. This also has the effect that the method cannot handle dynamic priorities.

A. Scalability

Comparing our work to [24], our framework obtains similar results in terms of response time distributions but with a much faster computation time. Based on the experiments shown in Table I and Fig. 7, the computation time of Matlab is exponential in the number of tasks and the multiple of $maxATime$ analyzed. The Matlab analysis time also grows much faster than the analysis time of UPPAAL based on the size of the initial probability samples as seen in Fig. 10. Based on results depicted in Table I, Fig. 8 and Fig. 10, the computation time of UPPAAL analysis grows linearly in all metrics. As an example, for a scheduling system of 3 tasks with 4 samples for each probability distribution of the task attributes, the computation time generated by our framework does not exceed 2% of the computation time of the response time generated by the Matlab implementation of [24]. An example of the computation time used by MATLAB for a simple system of 7 tasks with 4 samples for each probability distribution and 3500 time units (90 triggerings) as a simulation time is 47 hours.

We have successfully analyzed a system of 16 tasks having each 4 samples for the probability distributions. The analysis of such a system for 15 times $maxATime$ takes less than 8 minutes. We can thus conclude that our analysis method scales well.

VII. Conclusion

We have presented a flexible framework for the schedulability analysis of probabilistic sporadic tasks. The framework is given as a set of Parameterized Stopwatch Automata (PSA) models that can be analyzed using both Uppaal SMC and Uppaal. We have also introduced PoMD as a metric for the degree of schedulability of probabilistic scheduling systems. We compare the response time distributions we obtained to those obtained in [24]. The results are not identical, but we believe that the results we obtain are more useful when evaluating the schedulability of a real-time system. Our method considers all possible scenarios in the system while [24] only considers cases where the task under analysis has just been preempted. This gives a probability distribution of a set of *almost worst* case scenarios which we find less useful. The analysis time of our method grows linearly in all parameters of the system under analysis. This is in contrast with the method of [24] which grows exponentially in the number of tasks and the number of triggerings analyzed. We have shown the flexibility of our framework by including the analysis of a dynamic scheduling policy as well as by estimating another parameter of the system (PoMD).

References

[1] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 123–132, Dec 1998.

[2] A. Basu, S. Bensalem, M. Bozga, B. Delahaye, and A. Legay. Statistical abstraction and model-checking of large heterogeneous systems. *International Journal on Software Tools for Technology Transfer*, 14(1):53–72, 2012.

[3] A. Boudjadar, A. David, J. Kim, K. Larsen, M. Mikuionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In J. L. Fiadeiro, Z. Liu, and J. Xue, editors, *Formal Aspects of Component Software (FACS 2013)*, Lecture Notes in Computer Science, pages 61–78. Springer International Publishing, 2014.

[4] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In *Proceedings of FACS 2013*, LNCS volume 8348, P 61-78, 2013.

[5] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Degree of schedulability of mixed-criticality real-time systems with probabilistic sporadic tasks. In *Proceedings of TASE 2014*. IEEE Computer Society Press, 2014.

[6] P. E. Bulychev, A. David, K. G. Larsen, M. Mikucionis, D. B. Poulsen, A. Legay, and Z. Wang. UPPAAL-SMC: Statistical model checking for priced timed automata. In H. Wiklicky and M. Massink, editors, *QAPL*, volume 85 of *EPTCS*, pages 1–16, 2012.

[7] L. Carnevali, A. Melani, L. Santinelli, and G. Lipari. Probabilistic deadline miss analysis of real-time systems using regenerative transient analysis. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 299:299–299:308, New York, NY, USA, 2014. ACM.

[8] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[9] E. Clarke, J. Faeder, C. Langmead, L. Harris, S. Jha, and A. Legay. Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In M. Heiner and A. Uhrmacher, editors, *Computational Methods in Systems Biology*, volume 5307 of *LNCS*, pages 231–250. Springer Berlin Heidelberg, 2008.

[10] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):404–413, 1934.

[11] L. Cucu and E. Tovar. A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times. *SIGBED Review*, 3(1):7–12, 2006.

[12] A. David, K. G. Larsen, A. Legay, M. Mikucionis, D. B. Poulsen, J. van Vliet, and Z. Wang. Statistical model checking for networks of priced timed automata. In U. Fahrenberg and S. Tripakis, editors, *FORMATS*, volume 6919 of *LNCS*, pages 80–96. Springer, 2011.

[13] J. Diaz, D. Garcia, K. Kim, C.-G. Lee, L. Lo Bello, J. Lopez, S.-L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 289–300, 2002.

[14] S. Edgar and A. Burns. Statistical analysis of wcet for scheduling. In *Real-Time Systems Symposium, 2001. (RTSS 2001). Proceedings. 22nd IEEE*, pages 215–224, Dec 2001.

[15] J. Frey. Fixed-width sequential confidence intervals for a proportion. *The American Statistician*, 64(3):242–249, 2010.

[16] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PARAM: A model checker for parametric Markov models. In *Proc. 22nd International Conference on Computer Aided Verification (CAV'10)*, volume 6174 of *LNCS*, pages 660–664. Springer, 2010.

[17] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. Prism: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer Berlin Heidelberg, 2006.

[18] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[19] T. Hrault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2004.

[20] A. Legay and B. Delahaye. Statistical model checking : An overview. *CoRR*, abs/1005.1327, 2010.

[21] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[22] S. Manolache, P. Eles, and Z. Peng. Optimization of soft real-time systems with deadline miss ratio constraints. In *Real-Time and Embedded Technology and Applications Symposium, Proceedings. RTAS 2004. 10th IEEE*, pages 562–570, 2004.

[23] J. Martins, A. Platzer, and J. Leite. Statistical model checking for distributed probabilistic-control hybrid automata with smart grid applications. In S. Qin and Z. Qiu, editors, *Formal Methods and Software Engineering*, volume 6991 of *Lecture Notes in Computer Science*, pages 131–146. Springer Berlin Heidelberg, 2011.

[24] D. Maxim and L. Cucu-Grosjean. Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters. In *RTSS 2013 - IEEE Real-Time Systems Symposium*, Vancouver, Canada, 2013.

[25] D. Maxim, M. Houston, L. Santinelli, G. Bernat, R. I. Davis, and L. Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *RTNS*, pages 111–120. ACM, 2012.

[26] D. C. Montgomery. *Design and Analysis of Experiments*. June 2000.

[27] L. Santinelli, P. Yomsi, D. Maxim, and L. Cucu-Grosjean. A component-based framework for modeling and analyzing probabilistic real-time systems. In *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*, pages 1–8, Sept 2011.

[28] A. Thekkilakattil, R. Dobrin, and S. Punnekkat. Probabilistic preemption control using frequency scaling for sporadic real-time tasks. In *The 7th IEEE International SIES*, June 2012.

[29] F. Zhang, A. Burns, and S. Baruah. Sensitivity analysis for edf scheduled arbitrary deadline real-time systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2010 IEEE 16th International Conference on*, pages 61–70, Aug 2010.