Identification and Validation of Markov Models with Continuous Emission Distributions for Execution Times

Anna Friebe
Mälardalen University
Västerås, Sweden
anna.friebe@mdh.se

Alessandro V. Papadopoulos
Mälardalen University
Västerås, Sweden
alessandro.papadopoulos@mdh.se

Thomas Nolte
Mälardalen University
Västerås, Sweden
thomas.nolte@mdh.se

***Abstract*—It has been shown that in some robotic applications, where the execution times cannot be assumed to be independent and identically distributed, a Markov Chain with discrete emission distributions can be an appropriate model. In this paper we investigate whether execution times can be modeled as a Markov Chain with continuous Gaussian emission distributions. The main advantage of this approach is that the concept of distance is naturally incorporated. We propose a framework based on Hidden Markov Model (HMM) methods that 1) identifies the number of states in the Markov Model from observations and fits the Markov Model to observations, and 2) validates the proposed model with respect to observations. Specifically, we apply a tree-based cross-validation approach to automatically find a suitable number of states in the Markov model. The estimated models are validated against observations, using a data consistency approach based on log likelihood distributions under the proposed model. The framework is evaluated using two test cases executed on a Raspberry Pi Model 3B+ single-board computer running Arch Linux ARM patched with PREEMPT_RT. The first is a simple test program where execution times intentionally vary according to a Markov model, and the second is a video decompression using the `ffmpeg` program. The results show that in these cases the framework identifies Markov Chains with Gaussian emission distributions that are valid models with respect to the observations.**

***Index Terms*—Real-time systems, Markov Chain Model, Probabilistic Timing Analysis**

I. Introduction

In real-time systems requirements on timing properties must be considered, in addition to functional requirements, i.e., it is of importance to have the correct behavior *at the appropriate time*. Real-time requirements range from safety critical timing requirements of *hard real-time systems* found in applications of aeronautics, automotive and medical device systems to *soft real-time systems*, e.g., common in multimedia applications. Failure to meet hard real-time requirements may result in a disaster and/or loss of human life whereas failure to meet a soft real-time requirement can cause a deterioration of the Quality of Service (QoS) [1] such as in a video playback or affect the Quality of Control (QoC) [2] of a robot's motion planning.

It is a challenge to enable sufficiently accurate timing analysis of today's complex systems. Multicore processors [3] and mixed-criticality systems [4], [5], as well as fog and edge computing capabilities [6]–[8], require new methods for ensuring sound timing estimates along with functional integrity and limited over-provisioning of computational resources and bandwidth for communication. Practical timing analysis methods that consider the entire timing distribution, as opposed to only the tail of the distribution, can allow for system design without excessive over-provisioning. Taking the entire distribution into account is of particular interest primarily in the case of soft real-time applications where requirements on QoS or QoC are considered.

Frías et al. have shown that computation times of a computer vision application in a robotic system can be described as a Markov Model [9], [10]. Inspired by the work of Frías et al., in this paper we investigate the following research question: How can the execution time distribution of a task be faithfully modeled in a probabilistic framework? In particular:

1) Can execution time distributions be suitably modeled as a Markov Chain, where each state is associated with a Gaussian emission distribution?
2) How can one estimate the model parameters from timing measurements of the task's jobs?

Our main motivation for exploring *continuous emission distributions* is that they naturally include a concept of distance. Two execution time measurements that are similar are more likely to originate from the same state, compared to measurements of different magnitude. In the standard methods for Markov models with discrete emission distributions, each execution time value is treated as a label and the distance information is lost. By using continuous distributions, a model likely to provide a reasonable estimate from a smaller amount of observations. Although each execution time is a discrete number of clock cycles, realistic tasks on today's processors often result in a large number of possible values, that can be closely approximated by a continuous distribution. In order to

This work was supported by the Swedish Research Council (VR) via the project "Practical Probabilistic Timing Analysis of Real-Time Systems (PARIS)".

978-1-7281-4403-0/20/$31.00 ©2020 IEEE

develop methods that can be of practical use in schedulability analysis, we estimate model parameters from observations.

In this paper, we present an automated framework that estimates and validates an execution time distribution model from observations. The proposed model is a Markov Model with a Gaussian emission distribution associated with each state. More precisely, we propose an HMM, as we observe the execution times, but the states cannot be directly observed. Firstly, in **step 1** we identify the number of states for the HMM, and fit the model to the observations. A *tree-based cross-validation approach* [11] is adopted for identifying the number of states. We estimate the parameters for the Gaussian distributions and the transition matrix by applying the *Expectation-Maximization algorithm* [12], initialized with values resulting from the tree-based cross-validation. In **step 2** we validate the estimated model using observations. Here, we adopt a *data consistency approach* [13], and derive methods for application of this approach to Markov Chains using outputs from the *Forward-backward algorithm*.

A set of probabilistic techniques are selected and combined in the framework, to enable identification and validation of the HMM. The methods are applied to two test cases, a test program with a known Markov Chain behavior, and a video decompression program treated as a black box. The results are presented and discussed. Further investigation is needed to evaluate the applications where the framework and the specific techniques of each step are most suitable, and for what applications other techniques are better for one or several of the proposed steps.

The rest of the paper is structured as follows. Section II presents the related work, followed by Section III that presents the task model. Section IV presents the proposed framework. Sections V and VI discuss the experimental results, Finally, Section VII concludes the paper and highlight directions for future work.

II. Related Work

Cucu-Grosjean and Davis have recently provided thorough surveys of the literature on probabilistic methods in Timing Analysis [14], Response Time Analysis, analysis of server-based systems, Real-Time Queuing Theory, system analysis with fault modeling and Mixed Criticality Systems [15]. Cazorla et al. provide a taxonomy and a survey on the methods used in Probabilistic Worst-Case Execution Time Analysis [16]. The authors also emphasize the fact that while measurement-based approaches may allow analysis of a black-box system, the results are only reliable if the analysis data are representative with respect to the operational environment. That is, all sources of variation in execution times or latencies that are relevant for the result need to be contributing to the variation in the data. Otherwise their effects need to be upper-bounded or accounted for in other ways.

In the area of static probabilistic timing analysis, many works consider models for set-associative or fully associative caches. Quinones et al. [17] showed that for some cases with programs displaying a cache risk pattern, random replacement gives better results and lower variability compared to Least Recently Used (LRU) replacement. Altmeyer et al. [18] provide analysis considering reuse distance, associativity and contention. Analysis using random replacement caches has been extended to the multi-path case [19], [20]. Chen and Beltrame [21] perform timing analysis for single-path programs on systems with evict-on-miss random replacement caches by using an adaptive Markov model.

Measurement-Based Probabilistic Timing Analysis methods estimate the pWCET by applying statistical techniques to observations of execution time measurements. While WCET is a scalar value – the upper bound of the worst case execution time of runs – the pWCET is a probability distribution representing the upper bound on the probability of exceeding each execution time value in valid scenarios of repeated runs of the program. The theoretical basis is in Extreme Value Theory (EVT). The first work in this direction was by Burns, Edgar and Griffin [22]–[24]. Measurement-Based Probabilistic Timing Analysis was then introduced by Cucu-Grosjean et al. [25] in 2012.

Probabilistic Response Time Analysis is used to calculate the response time distribution of jobs, and in this manner estimate the probability of a deadline miss. Diaz et al. [26] presented response time analysis for a system with periodic tasks where random variables describe execution times. Here, the worst-case processor utilisation can exceed 1, since a backlog is considered at the end of the hyperperiod. They show that the backlog is a Markov chain. In [27] they also provided properties needed to achieve a safe over-approximation. More recent, the system model was extended by Kaczynski et al. [28] to also allow for systems with aperiodic tasks.

Similarly as in Measurement-Based Probabilistic Timing Analysis for pWCET estimates, EVT has also been applied in order to estimate response time distributions. The majority of the work in this line of research, Statistical Response Time Analysis, has been performed by Lu et al. [29]–[32].

Real-Time Queueing Theory is an area where queue lengths and waiting times are analyzed mathematically. Lehozcky [33] introduced the concept in 1996, building upon work on queuing theory that started in the 1950s. Doytchinov provided a mathematical formalization in [34].

Probabilistic analysis has also been applied in analysis of server-based systems. Buttazzo and Abeni introduced the Constant Bandwidth Server (CBS) [35], and probabilistic deadlines for Quality of Service guarantees [36]. The same group has considered execution times [37], [38] and interarrival times [38]–[40] modeled with probability distributions.

Frías et al. [9], [10] have published work regarding execution time models for tasks where execution times display dependencies due to slowly changing input data. They have shown that for a robotic image processing task in line following, modeling execution times as a Hidden Markov Model is appropriate. In this work, discrete emission distributions for the different states are used. The deadline miss probability under CBS/Earliest Deadline First (EDF)

is estimated for the Hidden Markov Model and compared to an assumption of independent and identically distributed random variables. The calculated probabilities are compared to experimental results with CBS/EDF as implemented in the Linux SCHED_DEADLINE scheduling policy. The experiments show that with an independent and identically distributed (i.i.d.) assumption of execution times, the probability of respecting the deadline is overestimated, i.e., the estimate is optimistic. The estimates based on the identified Hidden Markov Model, on the other hand, are very close to the experimental results.

III. Task Model

In this paper, we consider a periodic task τ consisting of a sequence of periodic jobs J_i, $i \in \mathbb{N}$, with period T. Each job has an execution time $c_i \in \mathbb{R}$.

We model the execution time distribution of the task according to an adapted version of the Markov Computation Time Model (MCTM) in Frías et al. [9]. The model is described by the set $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$, where

- $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$ is the set of N states, $m_n, n \in \mathbb{N}$.
- $\mathcal{P}$ is the $N \times N$ state transition matrix, where the element $p_{a,b}$ represents the conditional probability $\mathbb{P}(X_{i+1} = m_b | X_i = m_a)$ of being in state m_b at round $i+1$, given that at round i the state is m_a.
- $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$ is the set of execution time distributions, or emission distributions related to respective state. In this paper, these are modelled as Gaussian distributions with mean μ_n, and variance σ_n^2, i.e., $C_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$.

IV. Framework

In this section, we present and describe the framework that we have developed for the identification and validation of the probabilistic model. Specifically, the framework consists of the following steps:

1) Firstly, we apply the tree-based cross-validation approach [11] described in Section IV-A to identify the number of states in the HMM from the observations. An HMM with the identified number of states is fitted to the observations, according to the Expectation-Maximization [12] algorithm, using the likelihoods obtained with the Forward-backward algorithm [41]. The Gaussian distribution parameters and the transition matrix used as a starting point for the optimization is given by the outputs of the tree-based cross-validation.
2) Secondly, the obtained model is validated using a data consistency approach [13] described in Section IV-B. Here we derive expressions using outputs from the Forward-backward algorithm for application of the data consistency model validation.

In the following subsections we describe these steps on model identification and validation in more detail.

A. Tree-based Cross-Validation

In general, the number of states N is not known a priori, and must be identified, for example based on logged data. In order to identify a number of states N_{opt} that allows for capturing the execution sequence properties without overfitting, a tree-based cross-validation approach is applied, as described in Shinozaki [11]. The execution time sequence $cs = \{c_1, c_2, \ldots, c_{NS}\}$ consisting of execution times from $NS \in \mathbb{N}$ jobs, is split into M non-overlapping folds cs_f with index f.

$$cs = \cup_{f=1}^{M} cs_f$$
$$cs_f \cap cs_g = \emptyset, \quad f \neq g$$

For each fold with index f, we also define the complement cs_f^c, the remaining folds:

$$cs_f^c = \cup_{f \neq g} cs_g$$

For each fold an MCTM with $N > N_{\text{opt}}$ states is fitted to the remaining folds cs_f^c. The initial values of means and standard deviations for the emission distributions are given by k-means clustering with $k = N$.

The occupancy probability γ_{ni} is the probability of being in state n at round i, given the observation sequence cs_f, where $c_i \in cs_f$ and the model parameters retrieved from fitting to cs_f^c. The occupancy probabilities for each state index n and observation round i are calculated with the Viterbi algorithm [41].

Statistics containing all information from the sample needed for parameter value estimates for a statistical model are sufficient for the parameter. By calculating the sufficient statistics we can store the needed information from a sample in a compact manner. Sufficient statistics for likelihood estimates of a Markov chain model with Gaussian emission distribution are $a0$, $a1$ and $a2$ [11]. These are calculated for each fold index f and state index n:

$$a0_{fn} = \sum_{i, c_i \in cs_f} \gamma_{ni}$$
$$a1_{fn} = \sum_{i, c_i \in cs_f} c_i \gamma_{ni}$$
$$a2_{fn} = \sum_{i, c_i \in cs_f} c_i^2 \gamma_{ni}$$

For a set or cluster s of states, the maximum likelihood mean μ and variance ν for a fold index f can be calculated from the sufficient statistics from remaining folds:

$$\mu_{fs} = \frac{\sum_{g \neq f} \sum_{m_n \in s} a1_{gn}}{\sum_{g \neq f} \sum_{m_n \in s} a0_{gn}} \tag{1}$$

$$\nu_{fs} = \frac{\sum_{g \neq f} \sum_{m_n \in s} a2_{gn}}{\sum_{g \neq f} \sum_{m_n \in s} a0_{gn}} - \mu_{fs}^2 \tag{2}$$

These are then used to calculate a likelihood per fold index f and cluster s:

$$L_{fs} = -\frac{1}{2}\times \sum_{m_n \in s} \left(\ln(2\pi\nu_{fs}) a0_{fn} + \frac{a2_{fn} - 2\mu_{fs} a1_{fn} + \mu_{fs}^2 a0_{fn}}{\nu_{fs}} \right) \tag{3}$$

The likelihood for a cluster is then calculated by summation of the likelihoods of each fold index.

$$L_s = \sum_{f=1}^{M} L_{fs} \tag{4}$$

A tree is created, and initially all states are placed in a cluster in the root node. The cross validated likelihood is calculated for the tree consisting of only this cluster. Attempts are made to split the leaf nodes of the tree, so that the node's cluster is split into two clusters. The possible ways of splitting the states in a tree node are:

1) 2-means clustering of 2D data points of mean and standard deviation of each state is performed, and the resulting split is evaluated.
2) The states are ordered with respect to increasing mean, and each possible split along the ordered states is evaluated.
3) The modes are ordered with respect to increasing standard deviation, and each possible split along the ordered states is evaluated.

The split that gives the greatest increase in likelihood is selected. Nodes are split as long as the cross validated likelihood of the subtree is increased by the split, or until there is only one state in the tree node. The node splitting process is a greedy algorithm that may lead to a local maximum. Pseudo code is provided in Algorithm 1.

When a suitable number of states has been found, an MCTM with this number of states is fitted to the execution time samples, starting from initial values taken from the node clusters.

B. Data Consistency Model Validation

We evaluate whether the fitted model as described in Section IV-A,is valid, with respect to observations. Thus, we generate samples from the model and using a data consistency approach [13] we compare the generated samples to observations. If the observed data is consistent with data generated from the model, the model can be used in schedulability analysis.

The model validation can be performed with the same observations used for the model estimate, to evaluate whether the model can capture the properties of the observations. The evaluation can also be performed with observations from other runs of the program, to evaluate whether the model and parameters are valid in these cases, for different inputs or different hardware states.

The data consistency approach we apply is described by Lindholm et al. [13]. The log-likelihood under the proposed model is estimated for samples generated from the model and for the observed samples. Using a log-likelihood based statistic, an estimate is calculated of the probability of generating the observed sample or a sample with higher dispersion, from the evaluated model. This is equivalent to the probability that we reject the model on the basis of the observed data being overdispersed, assuming that the data is generated from the model. Another way of describing it is that the model is underdispersed compared with the observations. This probability of falsely rejecting the model or Probability of False Alarm due to underdispersion (PFA_u) is similar to the p-value concept in hypothesis evaluation. While the p-value is the probability of obtaining the test results or more extreme values assuming the null hypothesis is correct, the PFA_u is the probability of obtaining data with at least the observed variability, assuming they are generated from the proposed model.

We denote the observed execution times with an underline, as $\underline{\mathbf{c}}$. In our case, we evaluate a single model with a probability distribution $p(\mathbf{c}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$, where $\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*$ are the parameters of the fitted MCTM.

Using the model, we generate trajectories denoted with tilde $\tilde{\mathbf{c}} \sim p(\mathbf{c}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$. Using $c_{1:t}$ to denote the samples at rounds 1 to t from the trajectory, the conditional likelihood of an execution time measurement in a trajectory under the model is:

$$p_t = p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}{p(c_{1:t-1}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}$$

This can be calculated from the scaling factors resulting from the Forward-backward algorithm. We denote these as sca_i. From Rabiner [41] we have the probability of the observations expressed in terms of the scaling factors:

$$p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{\prod_{i=1}^{t} sca_i}$$

From this it is clear that the conditional probability can be written as:

$$p_t = p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{sca_t}$$

The conditional log-likelihood of a data point is:

$$z_t \triangleq \ln p(c_t|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = -\ln sca_t \tag{5}$$

Conditional probabilities of outputs for each state separately can be estimated using the transition matrix and the scaled forward variables $\hat{\alpha}$:

$$\begin{aligned} &p(c_t, X_t = j|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\ &= p(c_t|X_t = j) \sum_{k=1}^{N} p_{k,j} p(X_{t-1} = k|c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) \\ &= \frac{1}{\sigma_j \sqrt{2\pi}} e^{-\frac{(c_t - \mu_j)^2}{2\sigma^2}} \sum_{k=1}^{N} p_{k,j} \hat{\alpha}_{k,t-1} \end{aligned}$$

Algorithm 1 Pseudo code describing the tree cluster splitting process. The likelihood increase for possible splits of a cluster are calculated using the pre-computed sufficient statistics.

function TREECLUSTERSPLITTING($suffStats$, N)
  $tree \leftarrow createNode()$
  $tree.states \leftarrow [1:N]$ ▷ Add all states to the root cluster
  $tree.[leftStates, rightStates, advantage] \leftarrow calcSplitAdvantage(tree, suffStats)$ ▷ Find the best split
  while $(tree.advantage > 0)$ **and** $(nLeafNodes(tree) \leq N)$ **do** ▷ While the likelihood increases, and we can split leaves
    for $node \in leaves(tree)$ **do**
      if $node.advantage > 0$ **then**
        $node.leftChild \leftarrow createNode()$ ▷ Add new leaf nodes and split the state cluster
        $node.leftChild.states \leftarrow node.leftStates$
        $node.rightChild \leftarrow createNode()$
        $node.rightChild.states \leftarrow node.rightStates$
        $node.states \leftarrow \emptyset$
      end if
    end for
    for $node \in leaves(tree)$ **do**
      $node.[leftStates, rightStates, advantage] \leftarrow calcSplitAdvantage(node, suffStats)$ ▷ Find the best split of the leaf
    end for
    for $node \in tree; post-order$ **do**
      $node.advantage \leftarrow maxAdvantageChildren(node)$ ▷ Move the highest likelihood increase to the root
    end for
  end while
  return $tree$ ▷ Return the tree with N_{opt} leaf clusters
end function

From Rabiner [41] we have that:

$$\begin{aligned}\alpha_{k,t} &= p(c_{1:t}, X_t = k|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)\\ \hat{\alpha}_{k,t} &= \alpha_{k,t}\prod_{i=1}^{t} sca_i = \frac{p(c_{1:t}, X_t = k|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}{p(c_{1:t}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)}\\ &= p(X_t = k|c_{1:t}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)\end{aligned}$$

The conditional log-likelihood of each state and data point is:

$$\begin{aligned}z_{t,j} &\triangleq \ln p(c_t|X_t = j, c_{1:t-1}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)\\ &= -\ln\sigma_j - \frac{\ln 2\pi}{2} - \frac{(c_t - \mu_j)^2}{2\sigma^2} + \ln\sum_{k=1}^{N} p_{k,j}\hat{\alpha}_{k,t-1} \quad (6)\end{aligned}$$

We denote the mean of the log-likelihood of data points in generated trajectories as $\mathbb{E}[\tilde{z_t}]$ and the variance as $\mathbf{Var}[\tilde{z_t}]$. The test statistic T for a trajectory is defined:

$$T(\mathbf{c}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) = \frac{1}{n}\sum_{t=1}^{n}\frac{z_t - \mathbb{E}[\tilde{z_t}]}{\mathbf{Var}[\tilde{z_t}]} \quad (7)$$

T statistics are defined similarly for each state by replacing z_t with $z_{t,j}$. S is defined as the random event of a generated sample resulting in a higher T-statistic than the observed one:

$$S(\tilde{\mathbf{c}}, \underline{\mathbf{c}}) : T(\tilde{\mathbf{c}}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*) > T(\underline{\mathbf{c}}; \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$$

When the probability of S, $\mathbb{P}_{\tilde{\mathbf{c}}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*}(S(\tilde{\mathbf{c}}, \underline{\mathbf{c}}))$ is close to 0 or close to 1, it indicates that the observed data is inconsistent with the proposed model.

Lindholm et al. define PFA_u, the probability of falsely rejecting a model due to under-dispersion of the generated log likelihoods as:

$$PFA_u \triangleq \mathbb{P}_{\tilde{\mathbf{c}}|\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*}(S(\tilde{\mathbf{c}}, \underline{\mathbf{c}})) \quad (8)$$

Lindholm et al. also define the probability of falsely rejecting the model due to under- or overdispersion as:

$$PFA = \min(PFA_u, 1 - PFA_u)$$

However, in our work, we use PFA_u, as an under-dispersion of data generated from the proposed model indicates that the model is optimistic with regard to tail estimates. We note that values of PFA_u that are close to 1 also indicate model inconsistency, but that this relates to over-dispersion of data generated from the model.

Pseudo code for the data consistency approach is given in Algorithm 2.

V. EVALUATION

A. Test Setup

A Raspberry Pi 3B+ single board computer with quad-core 1.4 GHz BCM2837B0 is utilized in the tests. Arch Linux ARM kernel 4.14.87 with `PREEMPT_RT` patch 4.14.87-49 is configured with fully preemptible kernel and timer frequency of 100Hz. The SD card low latency mode and `dwc_otg` FIQ are disabled. A test program is pinned to a core that is isolated from load-balancing and scheduling algorithms. The scaling governor is set to performance for all cores and USB is disabled during the run. The program is run in user space with FIFO scheduling and maximum priority. The `ftrace` utility `trace-cmd` is used to log release (`sched_wakeup`) and scheduling (`sched_switch`) events, and to generate trace reports with nanosecond precision from the trace logs.

The model identification and validation framework is applied offline using the recorded traces.

Algorithm 2 Pseudo code describing the data consistency validation process.

function DATACONSISTENCYVALIDATION($\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, \underline{c}$)
  for $i \in 1 : M'$ **do**
    $traj1 \leftarrow generateTraj(\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, M', length(\underline{c}))$ ▷ Generate M' trajectories of the same length as observations.
    $simZ1[i] \leftarrow calcZ(traj1, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$ ▷ Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eq. 5 and 6.
  end for
  $EZ \leftarrow mean(simZ1)$ ▷ Estimate $\mathbb{E}[z_t]$ and N $\mathbb{E}[z_{t,j}]$ across M' values for each round t.
  $VarZ \leftarrow var(simZ1)$ ▷ Estimate $\mathbf{Var}[z_t]$ and N $\mathbf{Var}[z_{t,j}]$ across M' values for each round t.
  for $i \in 1 : M$ **do**
    $traj2 \leftarrow generateTraj(\mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*, M, length(\underline{c}))$ ▷ Generate M trajectories of the same length as observations.
    $simZ2[i] \leftarrow calcZ(traj2, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$ ▷ Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eq. 5 and 6.
    $Tsim[i] \leftarrow calcT(simZ2, EZ, VarZ)$ ▷ Calculate $M \times (N+1)$ Ts for z_t and $z_{t,j}$ as in Eq. 7 from simulated trajectories.
  end for
  $obsZ \leftarrow calcZ(\underline{c}, \mathcal{M}_*, \mathcal{P}_*, \mathcal{C}_*)$ ▷ Calculate log likelihoods z_t and N $z_{t,j}$ for the samples as in Eq. 5 and 6.
  $Tobs \leftarrow calcT(obsZ, EZ, VarZ)$ ▷ Calculate $N+1$ Ts for z_t and $z_{t,j}$ as in Eq. 7 from observations.
  $PFA_u \leftarrow count(Tsim > Tobs)/M$ ▷ Estimate the probability of S for the entire model and per state.
  return PFA_u
end function

B. Implementation

The tree-based cross validation approach described in Section IV-A and the data consistency criterion model validation described in Section IV-B are implemented in R[1], utilizing the R packages `depmixS4` [42] and `data.tree`. Evaluation code as well as test programs and scripts are available online [2].

Four folds are used for the cross validation. As described in Section IV-A, the MCTM is fitted to three folds, and the sufficient statistics using the fitted model are calculated for the remaining fold. The occupancy probabilities are determined by application of the Viterbi algorithm. A new MCTM with the number of states given by the tree-based cross validation is created, and initialized with the means and variances from the clusters, as given in Eq. 1 and Eq. 2 averaged over all folds. This model is then fitted to the entire training set, and the fitted model is validated with the data consistency criterion.

Values of the probability of false alarm due to underdispersion, PFA_u, are estimated for the entire model using z_t as in Eq. 5, and for each state in the model using $z_{t,j}$ as in Eq. 6. First, 100 trajectories are generated for estimation of the mean and variance of the log likelihood. The trajectories are generated using the simulate function in `depmixS4`, and log likelihoods are retrieved from `depmixS4`'s forward and scaling variable resulting from the Forward-backward algorithm. Second, 100 new trajectories are generated for calculation of T values as given by Eq. 7. Referring to Algorithm 2, both M' and M are set to 100.

C. Markov Chain Test Program

In a first test, a program with a known Markov Chain behavior is evaluated. The test program contains a state machine with three states. The program keeps an array of 100 integers, initialized from a random uniform distribution in the range [0, 4711]. It executes a job periodically at a 5ms interval. In the job, a state transition is performed, given the following transition matrix:

$$\mathcal{P} = \begin{pmatrix} 0.7 & 0.1 & 0.2 \\ 0.5 & 0.1 & 0.4 \\ 0.5 & 0.2 & 0.3 \end{pmatrix} \tag{9}$$

Depending on the current state, elements in the array are increased with 43 and a modulo operation with 4711 is performed. The first state has the shortest average execution time, the second state the middle and the third state the longest average execution time.

Logs are created from 21 runs of the program, one is used for model parameter estimation, and 20 in the model evaluation. In each run, the task releases 10 000 jobs. A python script is used to calculate the execution time for each job. The steady state is considered, so the logs from the first 250 jobs and the last executed job instance are excluded. The execution times of the first 250 are slightly lower, due to the program always starting in state 1 and possibly due to the system state. The last job's execution time is much longer due to produced status output before termination.

The execution time sequence used for estimating models is displayed in Fig. 1.

The tree based cross-validation approach is applied with 8 initial states to the training execution time sequence. The fitted model has six remaining states. The means and standard deviations of the states and PFA_u values are displayed in Table I, and the estimated transition matrix is given by:

$$\mathcal{P} = \begin{pmatrix} 0.51 & 0.18 & 0.08 & 0.02 & 0.19 & 0.007 \\ 0.45 & 0.27 & 0.05 & 0.04 & 0.18 & 0.005 \\ 0.36 & 0.14 & 0.07 & 0.047 & 0.38 & 0.005 \\ 0.31 & 0.18 & 0.07 & 3.7 \times 10^{-5} & 0.43 & 0.012 \\ 0.34 & 0.15 & 0.15 & 0.06 & 0.30 & 0.008 \\ 0.12 & 0.45 & 0.02 & 0.18 & 0.12 & 0.11 \end{pmatrix} \tag{10}$$

Based on the means and standard deviations from Table I, we can see that states 1 and 2 represent the program state with the lowest mean execution time, states 3 and 4 represent the middle program state and states 5 and 6 represent the highest

[1] https://www.r-project.org/

[2] https://github.com/annafriebe/MarkovChainETFramework

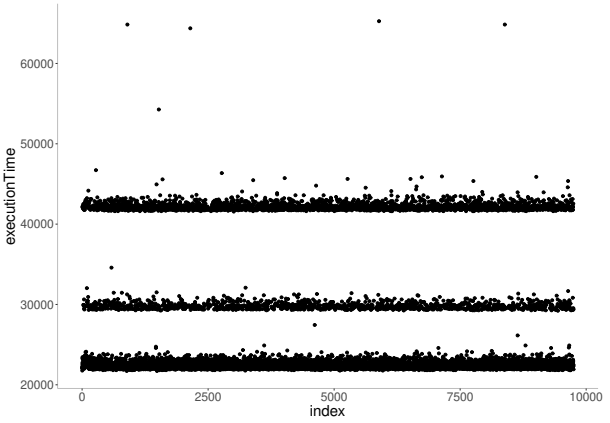


Fig. 1. The execution time sequence from the Markov Chain Test program used for estimating models. Times are given in nanoseconds.

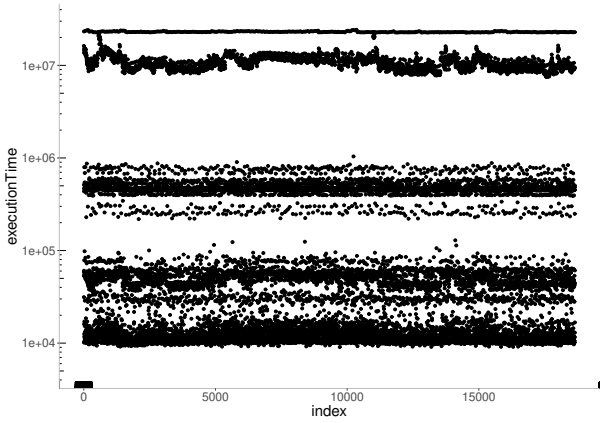


Fig. 2. The execution time sequence of the video decoding process. Times are given in nanoseconds, log scale.

TABLE I
STATE MEANS AND STANDARD DEVIATIONS (IN NANOSECONDS) AND CORRESPONDING PFA_u VALUES FOR THE MODEL ESTIMATED FROM THE TRAINING SEQUENCE.

State	1	2	3	4	5	6	All
mean	22240	22859	29652	30248	42185	41203	NA
stddev	222	420	221	409	383	9 190	NA
PFA_u							
test 1	0.22	0.21	0.18	0.18	0.94	0.44	0.31
test 2	0.52	0.49	0.33	0.33	0.14	0.56	0.87
test 3	0.38	0.37	0.24	0.24	0.43	0.05	0.02
test 4	0.28	0.27	0.19	0.18	0.53	0.20	0.19
test 5	0.36	0.36	0.22	0.22	0.42	0.22	0.24
test 6	0.14	0.14	0.12	0.14	0.75	0.10	0.06
test 7	0.26	0.26	0.18	0.18	0.67	0.38	0.19
test 8	0.00	0.00	0.01	0.01	0.58	0.00	0.00
test 9	0.58	0.58	0.43	0.39	0.01	0.44	0.89
test 10	0.55	0.53	0.42	0.40	0.02	0.62	0.92
test 11	0.27	0.26	0.19	0.20	0.76	0.24	0.17
test 12	0.43	0.43	0.29	0.26	0.04	0.31	0.41
test 13	0.48	0.47	0.25	0.23	0.03	0.31	0.50
test 14	0.74	0.73	0.50	0.46	0.01	0.69	0.99
test 15	0.28	0.28	0.19	0.20	0.58	0.31	0.14
test 16	0.29	0.28	0.21	0.21	0.41	0.14	0.56
test 17	0.37	0.37	0.21	0.21	0.09	0.13	0.76
test 18	0.36	0.36	0.24	0.23	0.37	0.12	0.41
test 19	0.28	0.28	0.21	0.21	0.53	0.59	0.61
test 20	0.19	0.20	0.18	0.19	0.81	0.16	0.11
train	0.40	0.39	0.28	0.26	0.45	0.48	0.94

program state and state 6 also some outliers. If we sum the values of columns 1-2, 3-4 and 5-6 for each row in Eq. 10, we see that for rows 1-5 they sum up to values similar to the corresponding transition probabilities in 9.

We see in Table I that the model is valid for all but one of the test sequences (test 8).

D. Video Decompression

A video is generated with images from the Tears of Steel open movie project[3]. The video is created with `ffmpeg` from

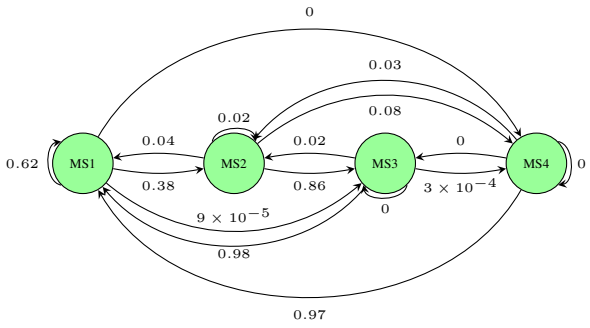


Fig. 3. Estimated transition probabilities between the macro states.

frames 5000–8999 of the `1080bis-png` images[4]. The frame rate is set to 25 fps.

A trace is logged during decoding of the video with `ffmpeg` in native frame rate. The sequence of execution times from the decoding is displayed in Fig. 2. We consider the steady state, therefore the first 250 and the last 50 execution time measurements have been discarded, as outliers are seen on visual inspection. From the figures it is clear that the execution times are separated in distinct groups, the lower with execution times below 0.15 ms, accounting for approximately 57% of the samples, a slightly higher with execution times below 1 ms and a peak at around 0.45 ms, accounting for about 22% of the samples, a higher and more varying 10 ms accounting for approximately 19% of the samples, and the high with execution times above 22.5 ms accounting for less than 2% of the samples. These groups are considered as macrostates, and the transition probabilities between the states are displayed in Fig. 3. The execution time sequence has been separated outside of the framework and the transition probabilities in Fig. 3 are estimated directly from the sequence. The framework analysis is applied to each of these groups separately. The PFA_u values are with respect to the sequence used for model estimation.

[3]https://mango.blender.org/

[4]https://media.xiph.org/tearsofsteel/tearsofsteel-1080bis-png/

1) Macro state 1: Execution times below 0.15 ms: The execution times $c_i < 0.15$ ms are extracted from the video decompression log, resulting in a log of 10 538 samples. A Markov chain model is identified in the first two steps of the framework - the tree based cross validation approach described in Section IV-A, and fitting to the observations. The initial number of states is 20, and the resulting Markov model has 13 states. The data consistency criterion PFA_u for each state and for the entire model is calculated for the observations. The features and PFA_u values for the model is given in Table II.

2) Macro state 2: Execution times in the range between 0.15 and 1 ms: The execution times $0.15 \leq c_i < 1$ ms consist of 4165 samples. The tree-based cross validation with 20 states in the initial Markov Model is applied and the resulting Markov Chain has 13 states. The state means and standard deviations of the estimated model, and the associated PFA_u values, are displayed in Table III.

3) Macro state 3: Execution times in the range between 1 and 22.5 ms: The execution times $1 \leq c_i < 22.5$ ms are 3598 samples. The tree-based cross validation does not generally find a solution in this case - for many starting values the `depmixS4` fit function is unable to complete the expectation maximization step. Starting from 24 initial states, a solution with 14 states is found. The state means and standard deviations and corresponding PFA_u values for the model are listed in Table IV.

4) Macro state 4: Execution times above 22.5 ms: 346 observations from the execution time sequence belong in macro state 4, $c_i \geq 22.5$ ms. The tree-based cross validation starting with 15 states identifies a model with 8 states. The state means and standard deviations and corresponding PFA_u values for the model estimated from the execution time sequence are shown in Table V.

VI. Discussion

The evaluation allows us to conclude that a Hidden Markov Model with Gaussian emission distributions can be appropriate to model execution time sequence data, and that the proposed framework can be used to identify and validate such a model.

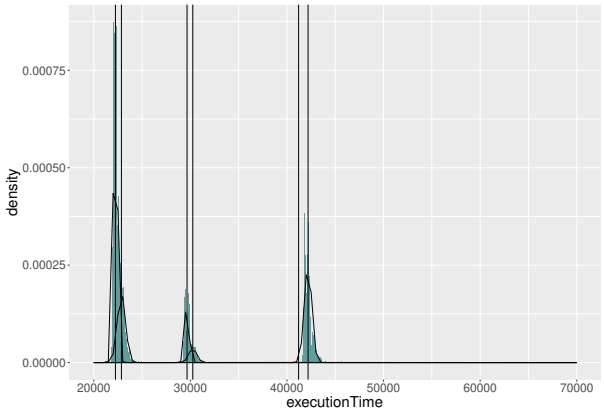
The analysis of the Markov Chain test program shows that the methods can be used to estimate the number of modes, the transition matrix, means and standard deviations to fit the model. While the test program is constructed to display Markov Chain properties, we show that the execution time distributions in each state can be modeled by a combination of modes with Gaussian emission distributions. Compared to the video decompression test, the program has a simple structure and a small memory footprint.

We also note that a Hidden Markov Model with Gaussian emission distributions appears to be valid in relation to the execution time sequences in the video decompression test.

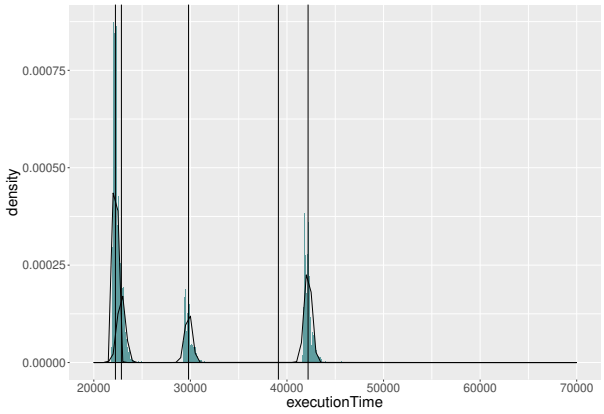
The `depmixS4` methods used in the tree-based cross validation step are somewhat sensitive to the initial number of states, and the step may fail if this number is too large or too small. These methods can also fail if there are significant gaps between the execution time values, which is why the video decompression sequence is separated into macrostates. The framework could be expanded to manage separation into macrostates and find a suitable initial number of states.

We also note that in some cases the number of states in the final models vary significantly. The heuristic algorithm for splitting the nodes could be adapted to evaluate a more exhaustive selection of possible splits, or replaced by an optimization algorithm such as for example simulated annealing [43].

Due to randomization utilized in many of the methods within the framework, different random seeds cause varying results. This is illustrated in Fig. 4, where the Gaussian distributions of two models are visualized on top of a normalized histogram of the sequence they are estimated from. The Gaussian distributions are scaled with their respective stationary distribution probabilities.



(a)



(b)

Fig. 4. Normalized histograms of execution times (in nanoseconds) of the execution time sequence of the Markov chain test program. Two different estimated models from different random seeds are visualized with the Gaussian distributions of the states scaled with their respective stationary distribution probability, and their means displayed as vertical lines. In (a) we see the six state model from Section V-C, and in (b) a five state model estimated with the framework applied to the same execution time sequence but initialized with another random seed.

We have conducted preliminary tests with the validation step performed with observations from running video decompression on another part of the "Tears of Steel" movie, that indicate that the identified model is not valid in this case. This may be due to input dependencies or cache related effects that cause

TABLE II
STATE MEANS AND STANDARD DEVIATIONS (IN NANOSECONDS) AND CORRESPONDING PFA_u VALUES WITH THE ESTIMATED MODEL FOR MACROSTATE 1.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**All**
mean	70 662	76 370	10 082	10 671	10 920	15 954	43 703	54 343	28 470	12 927	31 295	62 083	39 794	NA
stddev	15 236	2 360	230	640	578	2 585	1 069	4 095	1 908	1 034	3 024	2 159	2 065	NA
PFA_u	0.51	0.45	0.24	0.24	0.24	0.22	0.06	0.22	0.17	0.24	0.23	0.47	0.10	0.02

TABLE III
STATE MEANS AND STANDARD DEVIATIONS (IN NANOSECONDS) AND CORRESPONDING PFA_u VALUES WITH THE ESTIMATED MODEL FOR MACROSTATE 2.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**All**
mean	465 766	418 676	252 165	446 819	552 131	530 612	399 345	773 326	681 703	481 032	301 732	586 839	452 460	NA
stddev	9 790	9 832	13 083	4 442	33 902	5 549	3 868	37 217	9 397	13 513	13 543	13 245	4 530	NA
PFA_u	0.35	0.45	0.46	0.39	0.18	0.20	0.48	0.24	0.20	0.31	0.46	0.17	0.37	0.37

TABLE IV
STATE MEANS AND STANDARD DEVIATIONS (IN NANOSECONDS) AND CORRESPONDING PFA_u VALUES WITH THE ESTIMATED MODEL FOR MACROSTATE 3.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**All**
mean	12 146 116	9 907 489	9 408 568	11 763 973	8 753 004	13 172 000	13 071 808	11 726 350	15 321 999	10 722 598	10 652 123	8 223 196	10 074 260	11 246 543	NA
stddev	237 788	253 316	284 388	481 965	306 717	328 076	636 583	276 281	2 759 114	288 845	456 994	332 875	286 002	314 261	NA
PFA_u	0.07	0.16	0.17	0.08	0.17	0.03	0.07	0.08	0.45	0.14	0.18	0.00	0.16	0.07	0.80

TABLE V
STATE MEANS AND STANDARD DEVIATIONS (IN NANOSECONDS) AND CORRESPONDING PFA_u VALUES WITH THE ESTIMATED MODEL FOR MACROSTATE 4.

State	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**All**
mean	22 979 652	22 881 488	23 863 163	22 786 818	22 733 574	23 365 667	23 094 934	23 198 326	NA
stddev	57 727	682 112	197 495	36 255	66 411	140 413	54 839	110 609	NA
PFA_u	0.39	0.36	0.37	0.36	0.35	0.32	0.30	0.38	0.57

the Gaussian distribution parameters and transition matrix to change over time and between runs.

Chen and Beltrame [21] show that effects of a random replacement cache can be described by an adaptive Markov Chain, and the ARM processor on the Raspberry Pi applies a pseudo-random cache replacement policy. Our methods derive a homogeneous Markov Chain, and if the model changes significantly during the sequence used for model parameter estimation, the model will not be valid, and this will be reflected in the resulting PFA_u values.

Finally, we note that cache-related jitter in our evaluations may be exaggerated by the `ftrace` process running simultaneously.

VII. CONCLUSION AND FUTURE WORK

This work proposed a measurement-based framework for probabilistic modeling of execution times of real-time applications. It presented an end-to-end workflow that first identifies the structure of a Markov Chain model and fits the probabilistic distributions to the collected execution time data, and finally validates the obtained model on the collected data based on a data consistency approach.

As with all measurement-based approaches, the application of this framework requires that the observations used at analysis are representative of the observations at runtime.

In order for the models to be useful in cases where full representativity of the observations at analysis time is not realistic to achieve, the methods described in this paper need to be complemented with (i) a method for providing a safe over-approximation of the execution time distribution, and (ii) a method for dynamically updating the model to reflect the effects on execution time patterns due to changes in input, program state or hardware state.

It is worth noticing that the proposed framework presents a consistent combination of different probabilistic tools, but it can include other techniques as alternatives. For example, the approach proposed in [9] for the identification of the Markov model can be used in the first step of the proposed framework, as an alternative method. Further investigation on the tradeoffs among different techniques is needed, and it is deferred to future work.

The framework could be further extended and automated, e.g., by specifying required limits on the PFA_u values. If these are too close to 0 or 1, one can reject the model. Attempts can be made to identify new models in an iterative manner, until we find a model that is not rejected, or we reach an

iteration limit and deem the proposed model not consistent with the observations.

Finally, this paper focused on the single use case of video decompression. Other use cases will be analyzed in the future to better understand and investigate benefits and drawbacks of different probabilistic tools that can be included in this framework.

REFERENCES

[1] D. D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," *SIGCOMM Comput. Commun. Rev.*, vol. 22, no. 4, p. 14–26, 1992.

[2] P. Martí, J. M. Fuertes, G. Fohler, and K. Ramamritham, "Improving quality-of-control using flexible timing constraints: metric and scheduling," in *IEEE Real-Time Systems Symp. (RTSS)*, 2002, pp. 91–100.

[3] A. Löfwenmark and S. Nadjm-Tehrani, "Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective," *Journal of Systems Architecture*, vol. 87, pp. 1–11, 2018.

[4] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," *ACM Comput. Surv.*, vol. 50, no. 6, 2017.

[5] R. Wilhelm, "Mixed Feelings About Mixed Criticality (Invited Paper)," in *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, vol. 63, 2018, pp. 1:1–1:9.

[6] A. Čaušević, A. V. Papadopoulos, and M. Sirjani, "Towards a framework for safe and secure adaptive collaborative systems," in *IEEE Annual Computer Software and Applications Conf. (COMPSAC)*, vol. 2, 2019, pp. 165–170.

[7] V. Struhár, M. Behnam, M. Ashjaei, and A. V. Papadopoulos, "Real-time containers: A survey," in *Workshop on Fog Computing and the IoT (Fog-IoT)*, vol. 80, 2020, pp. 7:1–7:9.

[8] T. Cucinotta, L. Abeni, M. Marinoni, A. Balsini, and C. Vitucci, "Reducing temporal interference in private clouds through real-time containers," in *IEEE Int. Conf. on Edge Computing (EDGE)*, 2019, pp. 124–131.

[9] B. V. Frías, L. Palopoli, L. Abeni, and D. Fontanelli, "Probabilistic real-time guarantees: There is life beyond the i.i.d. assumption," in *IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2017, pp. 175–186.

[10] L. Abeni, D. Fontanelli, L. Palopoli, and B. V. Frías, "A markovian model for the computation time of real-time applications," in *IEEE Int. Instr. and Meas. Tech. Conf. (I2MTC)*, 2017, pp. 1–6.

[11] T. Shinozaki, "Hmm state clustering based on efficient cross-validation," in *IEEE Int. Conf. on Acoustics Speech and Signal Proc.*, vol. 1, 2006.

[12] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[13] A. Lindholm, D. Zachariah, P. Stoica, and T. B. Schön, "Data consistency approach to model validation," *IEEE Access*, vol. 7, pp. 59 788–59 796, 2019.

[14] R. I. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for Real-Time systems," *Leibniz Trans. Embedded Systems*, vol. 6, no. 1, pp. 03–1–03:60, 2019.

[15] ——, "A survey of probabilistic schedulability analysis techniques for Real-Time systems," *LITES: Leibniz Trans. Embedded Systems*, pp. 1–53, 2019.

[16] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega, "Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey," *ACM Comput. Surv.*, vol. 52, no. 1, 2019.

[17] E. Quinones, E. D. Berger, G. Bernat, and F. J. Cazorla, "Using randomized caches in probabilistic real-time systems," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, 2009, pp. 129–138.

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

[19] B. Lesage, D. Griffin, S. Altmeyer, and R. I. Davis, "Static probabilistic timing analysis for multi-path programs," in *IEEE Real-Time Systems Symp.*, 2015, pp. 361–372.

[20] B. Lesage, D. Griffin, S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis, "On the analysis of random replacement caches using static probabilistic timing methods for multi-path programs," *Real-Time Systems*, vol. 54, no. 2, pp. 307–388, 2018.

[21] C. Chen and G. Beltrame, "An adaptive markov model for the timing analysis of probabilistic caches," *ACM Trans. Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 1, pp. 1–24, 2017.

[22] A. Burns and S. Edgar, "Predicting computation time for advanced processor architectures," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, 2000, pp. 89–96.

[23] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *IEEE Real-Time Systems Symp. (RTSS)*, 2001, pp. 215–224.

[24] D. Griffin and A. Burns, "Realism in statistical analysis of worst case execution times," in *Int. Workshop on Worst-Case Execution Time Analysis (WCET)*, 2010.

[25] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-Based probabilistic timing analysis for multi-path programs," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, 2012, pp. 91–101.

[26] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *IEEE Real-Time Systems Symp. (RTSS)*, 2002, pp. 289–300.

[27] J. L. Diaz, J. M. Lopez, M. Garcia, A. M. Campos, K. Kim, and L. Lo Bello, "Pessimism in the stochastic analysis of real-time systems: Concept and applications," in *IEEE Int. Real-Time Systems Symp. (RTSS)*, 2004, pp. 197–207.

[28] G. A. Kaczynski, L. Lo Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, 2007, pp. 101–110.

[29] Y. Lu, T. Nolte, J. Kraft, and C. Norstrom, "Statistical-based response-time analysis of systems with execution dependencies between tasks," in *IEEE Int. Conf. on Engineering of Complex Computer Systems*, 2010, pp. 169–179.

[30] ——, "A statistical approach to response-time analysis of complex embedded real-time systems," in *IEEE Int. Conf. on embedded and real-time computing systems and applications*, 2010, pp. 153–160.

[31] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean, "A statistical response-time analysis of complex real-time embedded systems by using timing traces," in *IEEE Int. Symp. on Industrial and Embedded Systems*, 2011, pp. 43–46.

[32] ——, "A statistical response-time analysis of real-time embedded systems," in *IEEE Real-Time Systems Symp. (RTSS)*, 2012, pp. 351–362.

[33] J. P. Lehoczky, "Real-time queueing theory," in *IEEE Real-Time Systems Symp.*, 1996, pp. 186–195.

[34] B. Doytchinov, J. Lehoczky, and S. Shreve, "Real-time queues in heavy traffic with earliest-deadline-first queue discipline," *Annals of Applied Probability*, pp. 332–378, 2001.

[35] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *IEEE Real-Time Systems Symp. (RTSS)*, 1998, pp. 4–13.

[36] ——, "Qos guarantee using probabilistic deadlines," in *Euromicro Conf. on Real-Time Systems (ECRTS)*, 1999, pp. 242–249.

[37] ——, "Stochastic analysis of a reservation based system," in *Int. Workshop on Parallel and Distributed Real-Time Systems*, vol. 1, 2001.

[38] L. Palopoli, D. Fontanelli, L. Abeni, and B. V. Frias, "An analytical solution for probabilistic guarantees of reservation based soft real-time systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 27, no. 3, pp. 640–653, 2015.

[39] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1147–1156, 2012.

[40] N. Manica, L. Palopoli, and L. Abeni, "Numerically efficient probabilistic guarantees for resource reservations," in *IEEE Int. Conf. on Emerging Technologies & Factory Automation (ETFA)*, 2012, pp. 1–8.

[41] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proc. the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[42] I. Visser, M. Speekenbrink *et al.*, "depmixs4: an R package for hidden markov models," *Journal of Statistical Software*, vol. 36, no. 7, pp. 1–21, 2010.

[43] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.