A Markovian model for the computation time of real–time applications

Luca Abeni
DISI - University of Trento
Via Sommarive 5, 38123, Trento, Italy
Email: luca.abeni@unitn.it

Daniele Fontanelli
DII - University of Trento
Via Sommarive 9, 38123, Trento, Italy
Email: daniele.fontanelli@unitn.it

Luigi Palopoli, Bernardo Villalba Frías
DISI - University of Trento
Via Sommarive 5, 38123, Trento, Italy
Email: {luigi.palopoli,br.villalbafrias}@unitn.it

***Abstract*—Many real–time applications consist of a cyclic execution of computation activities (jobs) with stochastic computation time. In order to identify the probability that such applications will meet their deadlines, it is crucial to have a model for the random process describing the computation time. In many interesting applications, a Markovian model, in which the system stochastically switches within a discrete set of operating conditions (modes), is apparently a good fit for the actual behaviour of the process. In this paper, we discuss procedures and methods for the collection of samples of the computation time and for the identification of the underlying models based on the theory of hidden Markov models (HMM).**

I. Introduction

In many computing applications, software components are called to a tight interaction with the external environment. In these cases, it is imperative that the results of each computation are delivered with the correct timing (e.g., in order to generate an appropriate reaction to an event or to generate the signals that control the evolution of a system). Such software applications are commonly called real–time applications. Examples can be found in many measurement applications, such as smart grid state estimation [1], smart meters [2], signal processing [3], energy management units [4], wireless measurement applications [5], power substations [6], mechanical measures [7], target tracking [8] and visually servoed measurement instruments [9], [10].

A common misconception on real–time applications is that they are required to be efficient and "fast". This is certainly desirable, but is not really the defining feature of a real–time system. Much more important is *predictability*: every time the application is triggered by an external event (which can simply be the expiration of a timer), it executes a *job* that has to produce a result within a specified deadline. The deadline is said to be *met* if the job finishes before the deadline and is *missed* otherwise.

The traditional scientific literature on real–time systems considers that even a single deadline miss is a critical failure for the application [11]. A system designed under this hypotheses is called *hard real–time*. Starting from the seminal work of Liu and Layland [12], several authors have produced analysis techniques to guarantee that all the jobs generated by the applications, executing on the same processor, meet their deadlines. A very popular technique due to Pandya et al. [13] is based on the computation of the worst–case response time, i.e., of the maximum possible interval between the time an application is triggered and the time its result is delivered. Any analysis of this type has to account for the amount of computation time requested by an application in the worst case, for the frequency of the job activation requests and for the delays that the execution could suffer because of the presence of other applications in the system competing for the processor (scheduling delay). The scheduling delay is relatively easy to account for if the scheduler uses either static or dynamic priorities [12]. Much more critical is the knowledge of the worst–case execution time (WCET). This parameter is very difficult to measure or even to estimate when the application has strong data dependencies and/or when the architecture used for the execution is pipelined or uses a cache memory hierarchy. In such cases, the spread between average case and worst case could be large and the latter could be very improbable, hence escape the observation even from a large collection of execution traces.

To deal with this problem, different authors advocate the use of the so–called probabilistic WCET (pWCET) [14]. The idea is that for each execution, a task experiences a WCET, which changes depending on the input data set and on random effects in the system hardware. The pWCET can be obtained from measurements using statistical techniques derived from the Extreme Value Theory (EVT) [15] and allows the designer to provide hard real–time guarantees through standard technique, as those in [13], and associate the result with a level of confidence.

The use of the pWCET is grounded in the traditional vision that even a single deadline violation is a critical failure, and it simply estimates the probability that such a failure will actually occur. This is very useful for many industrial cases but is of little help when the application is resilient to occasional deadline misses. Such applications are called *soft real–time* and can be found in several areas ranging from multimedia (e.g., streaming or surveillance) to visual based robot control [16], [17]. For a soft real–time application, designers are interested in the Quality of Service (QoS) (e.g., measured by the frequency of deadline misses) delivered during every single run, rather than in the probability that for all possible runs a deadline will be missed. The notion of *probabilistic deadlines* has been proposed to accommodate for this requirement. While a "standard" deterministic deadline must always be respected,

This paper has received funding from the European Union's Horizon 2020 Research and Innovation Programme - Societal Challenge 1 (DG CONNECT/H) under grant agreement n° 643644 "ACANTO - A CyberphysicAl social NeTwOrk using robot friends".

a probabilistic deadline can be missed with a given probability. A probabilistic real–time guarantee states that the deadline will be respected with a probability at least equal to the one specified in the probabilistic deadline [18]. A probabilistic guarantee of this kind requires a system analysis and solution techniques based on queuing and stochastic systems theory and the recent literature on real–time systems offers several examples of techniques developed for this purpose [19], [20], [21].

A common assumption of these methods is that the stochastic process describing the computation time is independent and identically distributed (i.i.d.). An i.i.d. process makes the system's analysis tractable [22] and can be considered as a good approximation in many cases of interest [20]. However, there are also practical cases in which ignoring the correlation structure could lead to significant errors in the analysis. A possible situation is when the process behaves as a Markov Modulated Process (MMP). Essentially, the system switches between different operating modes through a Markov chain, and in each of these conditions the computation time is an i.i.d. process. We call this model Markov Computation Time Model (MCTM). In our previous work [23], we have shown that there are methods to extend the analysis developed for i.i.d. computation time to the case of MCTM. These methods require a model of the MCTM expressed in terms of a Markov chain, modelling the transitions between the different operating modes and the distribution of computation time in each mode. In this paper we focus on the problem of how to identify such a model from a long enough time series of the computation time. Among the different problems, the following are prominent: 1. how to measure the computation time in an efficient and accurate way, 2. how to estimate the number of operating modes (i.e., of states in the Markov chain) from a raw trace of execution times, 3. how many samples are necessary. In the paper, we will try to address these issues borrowing methods from the theory of the hidden Markov models (HMM).

The paper is organised as follows. In Section II, we discuss our measurement system exposing the most important problems related to measuring the computation time. In Section III, we describe our procedure for the identification of the MCTM. In Section IV, we show our results, obtained both from a synthetic data set and from a real robotic application. Finally, Section V discusses the obtained results and describes the possible future developments.

II. Measurement Technique and Platform

A real–time application is usually structured as a concurrent task τ, which consists of a stream of jobs denoted as J_i, with $i \in \mathbb{N}$. A pseudo–code example of a real–time task of this kind is shown in the following excerpt:

```
do{
        a_i = wait_for_next_activation();
        job_body();
        < f_i = get_time(); >
    }   while(true);
```

The first pseudo–call (`wait_for_next_activation()`) blocks the task until the activation event takes place. This event can be the expiration of a timer or an external event of different kind. When the function returns, the task is unblocked and the i–th job J_i can start. This happens at an absolute time a_i (said job activation time) that in our basic scheme is simply returned by the function. After job J_i starts, it needs to execute for a total time c_i, which changes depending on the input received and on architecture effects (e.g., cache, pipeline). The job computation (`job_body()`) finishes at a time f_i and the task is then blocked once again waiting for the next activation. The difference $R_i = f_i - a_i$ is called response time.

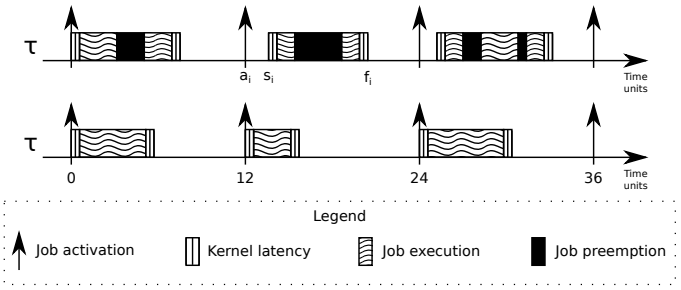

Figure 1. Example of the different delays a task con suffer. On the top image, a standard Linux Operating System is affected by the kernel latency and by the preemption of higher priority tasks. On the bottom, a Real–Time Operating System, where the real–time tasks do not suffer any preemption.

If one inserts a call for time measurement at the end of the function (which we denote by `get_time()`, within brackets in the example), it is possible to measure R_i and record it. However, this strategy does not necessarily produce an accurate measurement of the c_i. The first problem is that not all operating systems have accurate system calls to measure time. The second problem is that the execution of the Operating System calls to block and unblock the task and to do any kind of I/O operation usually have a latency, which in some cases could be time varying. Moreover, even the function used to measure time has a latency and then affects the time measure accuracy. Finally, if the task shares the CPU with other tasks, its execution could be "preempted" and the task could accumulate a "scheduling delay" that changes from job to job depending on the other tasks present in the system. All these effects are exemplified in the top half of Figure 1, where we see three different jobs being delayed by a different amount of time depending on the situation.

In order to address these issues, we obviously need a real–time operating system. A reliable solution can be based on the use of the Linux operating system. The reasons for this choice are the following.

1. As most POSIX compliant system, Linux features a standard function for time measurement (`gettimeofday`), which provides measurement with microsecond granularity. The recent version of Linux contain a facility (called high resolution timers), which offer a function call (`clock_gettime`) with a nanosecond resolution; the execution latency of the call is usually much greater than a nanosecond, but this solution reportedly enables measurements with sub–microsecond precision.

2. The Linux kernel can be equipped with a technology (called RT–preempt[1]) that reduces both the latency of the system calls and their variability. For the experiments reported in this paper,

[1] https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO

we used a WandBoard[2] running Ubuntu with the 4.8.1 Linux kernel and equipped with the RT–preempt patch. On this type of machine, the maximum latency of the system calls reported by a specialised website[3] is in the order of 50 μs, which is two order of magnitude below the minimum computation time that we measured.

3. The Linux kernel allows one to assign a fixed priority to the tasks, through the `SCHED_RR` or the `SCHED_FIFO` policy. If the task is assigned priority 99, which is the highest priority in the system, there is a theoretical guarantee that it will not undergo any scheduling delay. This is exactly what we did for the data collected in this paper. To be fair, the real guarantee that the task obtains in this way is that it will not suffer any preemption from another task, as shown in the bottom half of Figure 1. However, a task could still be preempted by the execution of interrupt handlers (which for standard operating systems are unaffected by the scheduling choices). Once again the use of a real–time kernel (e.g., Linux patched with the RT–preempt) can solve the problem, because interrupt handlers are executed for the most part within Kernel threads, which have an assigned priority and are not allowed to preempt tasks with a higher priority.

As discussed in the next sections, the execution traces will be analysed using HMM identification techniques to identify if a MCTM exists that fits the data. Our application scenario is not the typical one for HMM identification. Indeed, more often than not, HMM identification is carried out on "short" time series with a small number of possible measurable symbols [24], [25]. In our case, the number of possible observations equates the number of possible computation time, which is generally a large number (in some applications the possible values of the computation time could span from fractions of millisecond to hundreds of milliseconds, in both cases quite larger than the latencies induced by the real–time kernel), and hence facilitate the convergence of numeric integration methods [26]. On the other hand, collecting a large number of samples is relatively easier for the computation time than for other processes. In this unusual application context for HMM identification, it is imperative to understand the number of samples required to come up with a reasonably stable estimate of the HMM parameters.

III. Hidden Markov Models Identification

Due to their variability, the execution times c_j of a real–time task τ can be described as a stochastic process; unfortunately, the random variables composing the process are not independent (for example, the execution time of job J_j might depend on the execution time of job J_{j-1}). If the execution times process was composed by independent variables, it would have been possible to describe the execution time c_j of job J_j with a probability mass function $U_j(c) = \mathbf{Pr}\{c_j = c\}$; if all these random variables were identically distributed, the execution times of the tasks could have been described using a single probability mass function $U(c)$. But unfortunately, there are many cases in which this is not a good model for the task's execution times.

A more general formulation is instead to consider the stochastic process of the execution times as non–stationary. A tractable model can be derived if the process is assumed to behave as a Markov Modulated Process (MMP), which we define Markov Computation Time Model (MCTM). This choice comes from the nature of the stochastic process at hand: data–driven application execution times depend on the actual input data quality, which is very often consistent along consecutive executions. With this assumption, we show how a MCTM is effectively modelled by means of a *hidden Markov model*, or HMM. In this case, we assume that the execution times depend on an internal state s_j of the task that can assume a finite number of values, i.e. $s_j \in \mathcal{S} = \{S_0, \ldots, S_{n-1}\}$. If the task is in state S_i, then the execution time of the current job is distributed according to a probability mass function $U_i(c) = \mathbf{Pr}\{c_j = c \,|s_j = S_i\}$, i.e., there is a different execution times probability distribution for each state. Of course, the internal state s_j cannot be directly measured and it is assumed to be modelled as a Markov chain, i.e. the probability to be in state S_k at time j only depends on the value of the state at time $j-1$ [26].

In our HMM with n different states $\mathcal{S}$, the transition probabilities $p_{h,\,k} = \mathbf{Pr}\{s_j = S_k \,|s_{j-1} = S_h\}$ describe the stochastic transitions between the states. As is customary in the Markov chain literature [27], by denoting $\pi_j \in \mathbb{R}^n$, the row vector of probabilities of being in each state, we have immediately that

$$\pi_{j+1} = \pi_j M,$$

where $M = (p_{h,\,k})$ is the transition probability matrix. The output probability distributions set $\mathcal{U} = \{U_0(c), U_1(c), \ldots, U_{n-1}(c)\}$ models all the probability mass functions associated to each state. Therefore, the problem we intended to solve reads as: *Given a sequence of execution times measured as described in Section II, find the HMM parameters (number of states n, state transition probabilities M, and output probabilities $\mathcal{U}$) that better describe the process of the execution times.*

In the literature, this is known as the "*HMM Learning*" problem. Assuming the knowledge of the number of internal sates n, this problem can be addressed by using some well–known techniques such as the Baum–Welch algorithm [28]. The Baum–Welch algorithm is a form of Expectation Maximisation algorithm maximising the likelihood function L, which is defined as the probability for the identified HMM $\theta \triangleq (M, \mathcal{U})$ to generate a sequence of measured output values $c_0, \ldots, c_T$

$$L = \mathbf{Pr}\{c_0, \ldots, c_T \,|\theta\}.$$

The algorithm estimates M and $\mathcal{U}$ starting from two initial guesses that are iteratively refined. In each step of the iteration, the new estimations for M and $\mathcal{U}$ are computed based on the probability $\xi_j(h,\, k)$ for the HMM to be in state S_h at job J_j and in state S_k at job J_{j+1}, given the sequence of observed computation times $c_0, \ldots, c_{j+1}$

$$\xi_j(h,\, k) = \mathbf{Pr}\{s_j = S_h \wedge s_{j+1} = S_k \,|c_0, \ldots, c_{j+1}\}.$$

Based on this, a new estimation of M can be computed as

$$p_{h,\,k} = \frac{\sum_j \xi_j(h,\, k)}{\sum_j \sum_i \xi_j(h,\, i)}.$$

[2] www.wandboard.org

[3] https://www.osadl.org/Hardware-overview.qa-farm-hardware.0.html#ARM_TI

Although this algorithm works perfectly from the theoretical point of view, it can be subjected to numerical stability problems (underflow) when the number of observed computation times is too high, as noticed in [29]. This happens because $\xi_j(h,\, k)$ is computed based on the so called *forward probabilities* $\alpha_j(h)$, i.e. probability for the HMM to be in state S_h at job J_j

$$\alpha_j(h) = \mathbf{Pr}\left\{s_j = S_h \wedge c_0,\, \ldots,\, c_j\right\},$$

and *backward probabilities* $\beta_j(h)$, i.e. probability to see computation times $c_{j+1},\, \ldots,\, c_T$ given that the task is in state S_h at job J_j

$$\beta_j(h) = \mathbf{Pr}\left\{c_{j+1},\, \ldots,\, c_T \,|s_j = S_h\right\}.$$

The forward probabilities $\alpha_j(h)$ can be computed using an inductive expression $\alpha_j(h) = \sum_k \alpha_{j-1}(k) p_{k,\,h} U_h(c_j)$, and the backward probabilities can be computed as $\beta_j(h) = \sum_k \beta_{j+1}(k) p_{h,\,k} U_k(c_{j+1})$. Since we have a lot of measured execution times, j can become large, and, hence, $\alpha_j(\cdot)/\beta_j(\cdot)$ are multiplied a large number of times for probabilities that are less than 1, thus becoming 0 due to underflow errors. This problem can be solved by modifying the Baum–Welch algorithm to use conditional probabilities instead of joint probabilities, as shown in [29].

Given a sequence of observed execution times, the revised Baum–Welch algorithm is able to estimate the transition matrix M and the output probability distributions $\mathcal{U}$ if the number n of distinct internal states s_j is available. In other words, the dimension of M and the number of different PMFs $\mathcal{U}$ should be known upfront. Hence, the problem is how to estimate such a value. In order to correctly dimension n, we made use of a *cross–validation* approach for the likelihood [30]. Therefore, to estimate the number of states n we adopted a gradient–like approach:

1) Set $n = 1$, then execute the EM procedure and evaluate the cross–validated likelihood L;
2) Set $L^\star = L$;
3) Set $n = n + 1$, then execute the EM procedure and evaluate the cross–validated likelihood L;
4) If $L \leq L^\star$, go to step 2);
5) The optimal number of states is $n - 1$, with cross–validated likelihood $L^\star$.

If has to be noted that this algorithm is based on the experimental evidence, also reported in Section IV, that increasing the number of states more than needed also increases the cross–validated likelihood. Although only a conjecture at this point, which is grounded to the theoretical results on gradient–based methods [26] and will be the subject of future work, this approach proves to be efficient in all the test cases adopted. Another remarkable feature is that the cross–validated likelihood can also be used to establish the maximum number of execution time measures to be used in the identification, again using a gradient–like algorithm. Even though more data means higher estimation accuracy, we found out that the number of measures to be analysed can be limited according to the observed improvement of the cross–validated likelihood gradient, as discussed in the next section.

IV. Results

The validation of the proposed method is based on both synthetic data and a real robotic application. First of all, the effectiveness of the cross–validation method has been tested using some synthetic data generated in simulation from a known HMM, with the output probability distributions represented in Table I and the following transition probability matrix:

$$M = \begin{bmatrix} 0.80 & 0.20 & 0.00 \\ 0.00 & 0.70 & 0.30 \\ 0.15 & 0.25 & 0.60 \end{bmatrix}.$$

The samples synthetically generated from this HMM have been divided in 20 partitions, using one of them for identifying the HMM and the remaining 19 for computing the cross–validated likelihood. This has been repeated multiple times, for different numbers of states (ranging from 1 to 6) and using different numbers of measures for training the HMM. The ratio between the cross–validated likelihoods and the one obtained for 30000 measures and $n = 3$ states are reported in percentage in Table II as a function of the number of measures versus the number of states. From the table, two main outcomes can be identified: a) the HMM with $n = 3$ states is consistently the one having the largest cross–validated likelihood, and this confirms that the cross–validated likelihood is a good indicator for selecting the "best" HMM parameters; b) using more than 5000 measures the improvements in the cross–validated likelihood are negligible (less than one thousandth), hence 5000 measures are sufficient to correctly identify the HMM parameters. In such a case, the transition probability matrix $\widehat{M}$ estimated with 5000 measures is

$$\widehat{M} = \begin{bmatrix} 0.805 & 0.195 & 0.000 \\ 0.000 & 0.693 & 0.307 \\ 0.160 & 0.252 & 0.588 \end{bmatrix},$$

which has an error that is less than 1%. The output probability distributions $\mathcal{U}$ (not reported here for the sake of brevity) also looks pretty similar to the original ones. Notice that the algorithm has been also tested, but not reported for space limits, on standard i.i.d. processes, ending up with an HMM with $n = 1$ states, as expected.

For what concerns the experiments, the proposed approach has been used to model the execution times of a robotic application in which computer vision algorithms are used to identify a ribbon placed on the floor, from images grabbed by a camera placed on–board a wheeled mobile robot. From those measures, both the position and orientation of the robot are estimated on–line. The lane detection algorithm executes on a WandBoard running Ubuntu with the 4.8.1 Linux kernel, and the execution times of the various jobs (each job is in charge of processing a video frame) have been measured as described in Section II. An increasing number of execution time measurements (ranging from 500 to 50000) has then been used to learn the parameters of an HMM with a number of states ranging from 1 to 8. Again, the percentage of the ratio of the cross–validated likelihoods with the one obtained for 50000 measures and $n = 6$ states are reported in percentage in Table III. From these data it is possible to see on a real experiment how the gradient–like algorithm is well funded, since increasing the number of states up to 6 greatly increases the cross–validated likelihood, while increasing further the

Table I. OUTPUT PROBABILITY DISTRIBUTIONS FOR THE HMM OF THE FIRST EXPERIMENT.

$\mathcal{S}$	Computation time c															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
S_0	0.004	0.159	0.787	0.05	-	-	-	-	-	-	-	-	-	-	-	-
S_1	-	-	0.050	0.232	0.198	0.134	0.157	0.101	0.062	0.034	0.026	0.006	-	-	-	-
S_2	-	-	-	-	-	-	0.006	0.048	0.072	0.109	0.132	0.147	0.131	0.159	0.125	0.071

Table II. RATIO BETWEEN THE CROSS–VALIDATED LIKELIHOODS AND THE ONE OBTAINED FOR 30000 MEASURES AND $n = 3$ STATES, REPORTED IN PERCENTAGE. DIFFERENT NUMBERS OF STATES n AND MEASURES USED TO IDENTIFY THE HMM ARE DEPICTED. DATA ARE COLLECTED FROM SYNTHETIC DATA.

#	n					
	1	2	3	4	5	6
500	12.04	4.69	2.52	5.09	6.18	7.13
1000	11.52	3.61	1.06	2.12	2.61	4.18
2000	11.32	3.26	0.34	0.62	1.11	1.60
3000	11.37	3.28	0.30	0.40	0.81	1.10
4000	11.32	3.25	0.27	0.39	0.60	0.81
5000	11.31	3.15	0.16	0.23	0.39	0.55
7500	11.28	3.18	0.18	0.20	0.33	0.46
10000	11.22	3.09	0.12	0.13	0.20	0.25
15000	11.20	3.08	0.03	0.04	0.08	0.15
20000	11.14	2.98	0.03	0.04	0.06	0.07
25000	11.20	2.96	0.02	0.03	0.05	0.08
30000	11.19	2.95	0	0.02	0.06	0.07

Table III. RATIO BETWEEN THE CROSS–VALIDATED LIKELIHOODS AND THE ONE OBTAINED FOR 50000 MEASURES AND $n = 6$ STATES, REPORTED IN PERCENTAGE. DIFFERENT NUMBERS OF STATES n AND MEASURES USED TO IDENTIFY THE HMM ARE DEPICTED. DATA ARE COLLECTED FROM THE ROBOTIC EXPERIMENT.

#	n							
	1	2	3	4	5	6	7	8
500	23.90	23.08	30.59	34.73	39.75	52.24	45.25	49.72
1000	17.30	12.12	14.09	14.75	22.71	16.84	25.57	24.59
2000	10.53	7.31	5.74	6.45	7.90	8.44	9.80	11.36
3000	10.03	5.12	3.87	3.72	4.83	5.23	6.75	7.16
4000	9.17	4.44	2.94	3.01	3.19	4.11	4.79	5.53
5000	8.94	4.04	2.46	2.26	3.05	3.34	4.01	4.32
7500	8.77	3.78	2.05	1.79	1.74	2.07	2.68	2.77
10000	8.55	3.46	1.72	1.31	1.20	1.39	2.09	2.07
12500	8.50	3.35	1.65	1.15	0.97	0.98	1.21	1.54
15000	8.40	3.23	1.56	0.99	0.89	0.80	1.07	1.23
20000	8.31	3.09	1.34	0.78	0.60	0.63	0.81	0.82
30000	8.20	2.93	1.17	0.57	0.33	0.27	0.28	0.36
40000	8.13	2.89	1.04	0.44	0.18	0.09	0.10	0.19
50000	8.07	2.82	1.05	0.35	0.11	0	0.01	0.02

number of states does not seem to have too many advantages. Moreover, using more than 15000 results in a negligible improvement in terms of likelihood. Of course, we cannot compare the transition probability matrix M and $\mathcal{U}$ since the actual value is unknown in this case. We decided next to model the execution times of this robotic application using a 6–states HMM trained with 15000 measures of the execution time.

To validate the model, we replicated the experiments again and checked if the real–time performance predicted using the derived HMM matched the ones of the real application. The vision algorithm was, in this case, executed in a periodic task processing a frame every 200 ms. The task was scheduled using the SCHED_DEADLINE [31] policy provided by the Linux kernel, and reserving a fraction of the CPU idle time to this task. The experimental probability of respecting a relative deadline, averaged through 20 executions, was compared with the one estimated using the HMM. Additional comparisons have been carried out with an i.i.d. model, derived with the technique presented in [20]. The comparison is reported in Figure 2 and shows that the probability of missing a deadline

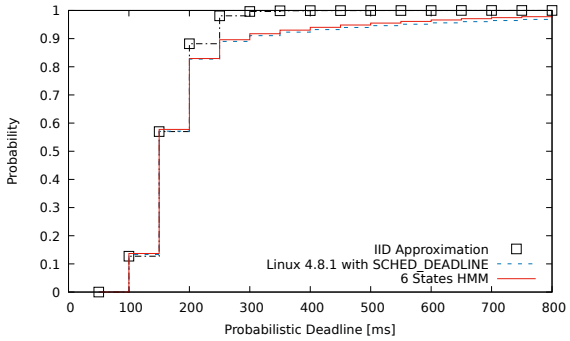

Figure 2. Probability of respecting the deadline for the robotic experiments.

obtained with the synthesised HMM are very tight to the ones measured experimentally. Indeed, a maximum error of less than one hundredth is measured for the proposed HMM model, while the i.i.d. approximation yields an error that is one order of magnitude larger. Moreover, it is evident how in this case the i.i.d. approximation results in an *overestimation* of the probability to respecting the deadline, and such an optimistic analysis can be dangerous when designing a real–time system.

V. CONCLUSIONS

In real–time applications with probabilistic guarantees it is important to know the stochastic process describing the computation times. A frequent choice is to ignore the correlation structure and to consider the process as i.i.d., which we have shown with experimental data that may fall short of producing decent results in some applications. In this paper, we have considered a more general Markovian model, the MCTM, which associates a different state with every working condition of the system and generates a different distribution in each of these cases and recovers the i.i.d. process as a special case. We have particularly focused on an effective procedure for identifying parameters of the MCTM, adapting ideas and techniques from HMM estimation. The effectiveness of the approach has been shown on both synthetic data and on a real data set taken from a robotic application.

Future research efforts will be directed toward the comparison with other existing algorithms and the application of the technique to a larger set of applications. Another direction will be the development of an iterative algorithm which will incrementally adapt the estimated parameters of the model as the samples are collected. Such a method could potentially pave the way for the development of adaptive scheduling schemes in which the scheduling parameters are adjusted online in order for the application to meet a target on the probability of missing the deadline.

REFERENCES

[1] A. Angioni, J. Shang, F. Ponci, and A. Monti, "Real-Time Monitoring of Distribution System Based on State Estimation," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 10, pp. 2234–2243, Oct 2016.

[2] M. S. Reza, M. Ciobotaru, and V. G. Agelidis, "Power system frequency estimation by using a newton-type technique for smart meters," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 3, pp. 615–624, March 2015.

[3] M. S. Reza and V. G. Agelidis, "A robust technique for single-phase grid voltage fundamental and harmonic parameter estimation," *IEEE Transactions on Instrumentation and Measurement*, vol. 64, no. 12, pp. 3262–3273, Dec 2015.

[4] E. Din, C. Schaef, K. Moffat, and J. Stauth, "A scalable active battery management system with embedded real-time electrochemical impedance spectroscopy," *IEEE Transactions on Power Electronics*, vol. PP, no. 99, pp. 1–1, 2016.

[5] L. Angrisani, L. Battaglia, and F. Delfino, "Grid-based power measurement in digital wireless communication systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 56, no. 5, pp. 1565–1572, Oct 2007.

[6] Z. Zhu, S. Dai, L. Xiao, J. Zhang, Y. Teng, W. Guo, D. Zhang, Z. Gao, N. Song, Z. Zhang, Q. Qiu, X. Xu, G. Zhang, T. Ma, and L. Lin, "A real-time measuring and control system for the world's first hts power substation," *IEEE Transactions on Applied Superconductivity*, vol. 23, no. 3, June 2013.

[7] Q. Xue, H. Leung, R. Wang, B. Liu, and Y. Wu, "Continuous real-time measurement of drilling trajectory with new state-space models of kalman filter," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 1, pp. 144–154, Jan 2016.

[8] G. Plantier, N. Servagent, T. Bosch, and A. Sourice, "Real-time tracking of time-varying velocity using a self-mixing laser diode," *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 1, pp. 109–115, Feb 2004.

[9] N. Marturi, B. Tamadazte, S. Dembl, and N. Piat, "Visual Servoing-Based Depth-Estimation Technique for Manipulation Inside SEM," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 8, pp. 1847–1855, Aug 2016.

[10] D. Fontanelli, F. Moro, T. Rizano, and L. Palopoli, "Vision-Based Robust Path Reconstruction for Robot Control," *IEEE Trans. on Instrumentation and Measurement*, vol. 63, no. 4, pp. 826–837, April 2014.

[11] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Boston: Kluwer Academic Publishers, 1997.

[12] C. L. Liu and J. Layland, "Scheduling alghorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, 1973.

[13] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, p. 390, 1986.

[14] G. Bernat, A. Colin, and S. Petters, "pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems," *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS*, 2003.

[15] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Proceedings of the Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 2012.

[16] A. Cervin, B. Lincoln, J. Eker, K. Arzen, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of the IEEE International Conference on Real-Time and Embedded Computing Systems and Applications*. Gothenburg, Sweden, 2004.

[17] D. Fontanelli, L. Greco, and L. Palopoli, "Soft RealTime Scheduling for Embedded Control Systems," *Automatica*, vol. 49, pp. 2330–2338, July 2013.

[18] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation-based system," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium.*, San Francisco, California, April 2001.

[19] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadline," in *Proceedings of the Euromicro Conference on Real-Time Systems*. Pisa, Italy: IEEE, September 2012.

[20] L. Palopoli, D. Fontanelli, L. Abeni, and B. Villalba Frías, "An analytical solution for probabilistic guarantees of reservation based soft real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 640–653, March 2016.

[21] A. Mills and J. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*. Stockholm, Sweden: IEEE, April 2010.

[22] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[23] B. Villalba Frías, L. Palopoli, L. Abeni, and D. Fontanelli, "Probabilistic real–time guarantees: There is life beyond the i.i.d. assumption," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2017 IEEE*, to appear.

[24] D. Vasquez, T. Fraichard, and C. Laugier, "Incremental Learning of Statistical Motion Patterns With Growing Hidden Markov Models," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 403–416, Sept 2009.

[25] C. Li and S. V. Andersen, "Efficient blind system identification of non-Gaussian autoregressive models with HMM modeling of the excitation," *IEEE Transactions on Signal Processing*, vol. 55, no. 6, pp. 2432–2445, 2007.

[26] O. Cappé, E. Moulines, and T. Rydén, *Inference in Hidden Markov Models*, ser. Springer Series in Statistics. Springer, 2005.

[27] S. Karlin, *A first course in stochastic processes*. Academic press, 2014.

[28] L. E. Baum, "An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes," in *Proceedings of the 3rd Symposium on Inequalities*, 1972, pp. 1–8.

[29] P. A. Devijver, "Baum's forward-backward algorithm revisited," *Pattern Recognition Letters*, vol. 3, no. 6, pp. 369–373, 1985.

[30] T. Shinozaki, "Hmm state clustering based on efficient cross-validation," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 1. IEEE, 2006, pp. I–I.

[31] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016, spe.2335. [Online]. Available: http://dx.doi.org/10.1002/spe.2335