Contributions to real-time systems

Liliana Cucu-Grosjean

THÈSE D'HABILITATION à DIRIGER des RECHERCHES
de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Liliana CUCU-GROSJEAN

Pour obtenir

l'HABILITATION à DIRIGER des RECHERCHES de
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet :

Contributions à l'étude des systèmes temps réel

Contributions to real-time systems

soutenue le 12 mai 2014

devant le jury composé de :

Sanjoy Baruah (Université de Caroline de Nord)	Rapporteur
Giorgio Buttazzo (Scuola Superiore Sant'Anna)	Rapporteur
Yvon Trinquet (Ecole Centrale de Nantes)	Rapporteur
Avner Bar-Hen (Université Paris 5)	Examinateur
Philippe Baptiste (Ministère de l'Enseignement Supérieur et de la Recherche)	Examinateur
Alix Munier-Kordon (Université Paris 6)	Examinateur
Isabelle Puaut (Université de Rennes)	Examinateur
Pascal Richard (Université de Poitiers)	Examinateur
Françoise Simonot-Lion (Université de Lorraine)	Examinateur

,

Contents

,

Introduction

The use of computers to control safety-critical real-time functions has increased rapidly over the past few years. As a consequence, real-time systems — computer systems where the correctness of each computation depends on both the logical results of the computation and the time at which these results are produced — have become the focus of important study. Since the concept of "time" is important in real-time application systems, and since these systems typically involve the sharing of one or more resources among various contending processes, the concept of scheduling is integral to real-time system design and analysis. Scheduling theory, as it pertains to a finite set of requests for resources, is a well-researched topic. However, requests in a real-time environment are often of a recurring nature. Such systems are typically modelled as finite collections of simple, highly repetitive tasks, each of which generates jobs in a predictable manner.

The *scheduling algorithm* determines which job[s] should be executed at each time-instant. When there is at least one schedule satisfying all constraints of the system, the system is said to be *schedulable*.

Uniprocessor real-time systems have been well studied since the seminal paper of Liu and Layland [48], which introduces a model of periodic systems. There is a tremendous amount of literature considering scheduling algorithms, feasibility tests and schedulability tests for uniprocessor scheduling. During the last years the real-time systems have evolved to more complex models and architectures. The two main contributions of our work covers these two aspects with results on (probabilistic) models and (multiprocessor) architectures. A real-time model is probabilistic if at least one parameter is described by a random variable.

This document does not contain all our results and it presents a synthesis of the main concepts and the description of the most relevant results. For each contribution we will provide an exhaustive list of our related results and their place in the real-time literature.

The order of the presentation of results is chronological and motivated by how my[1] reasoning evolved during this period since my PhD thesis.

My first year working on probabilistic (at ISEP following the suggestion of Prof. Eduardo Tovar) has been spent on understanding how to get closer two worlds that

[1] This part of the introduction shares with the reader on how I moved from deterministic scheduling to probabilistic timing analysis and the utilisation of "I" seems more appropriate.

seem so different:

- the real-time community imposing worst-case reasoning as basic rule for obtaining safe results
- the probabilistic community attracted by average behavior of systems and the central part of a distribution describing the parameters.

After publishing a first result on probabilistic [13], I had the conviction that the differences were less important than the misunderstanding surrounding the results on probabilistic but by that time I didn't have the means to explain what was the misunderstanding. Continuing my visits in Europe, I had started to work on multiprocessor platforms and mainly on the predictability of such platforms (joining Prof. Joel Goossens at ULB). The predictability of an algorithm indicates that any task, that starts earlier and it is shorter than other task, should finish the execution before that latter task. After proving that the first result on predictability had an incorrect proof, we had proposed the most general result (as it was in 2010), here general is defined with respect to the class of scheduling algorithms and architectures. Since then, a more general result has been obtained in 2013 in collaboration with J. Goossens and E. Grolleau [35].

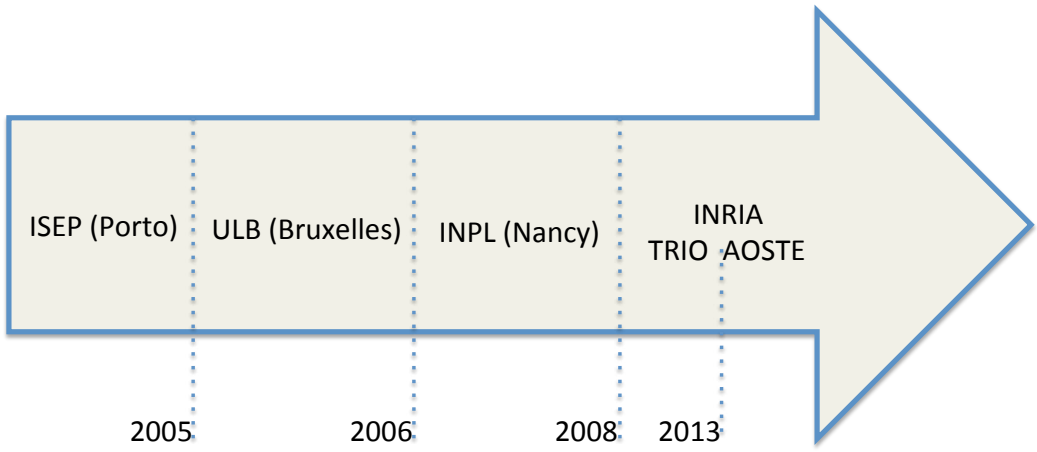


Figure 1: A timeline

After working at ISEP (Porto) on probabilistic schedulability analysis and at ULB (Bruxelles) on multiprocessor schedulability analysis, I had naturally worked on probabilistic multiprocessor analysis during my first years at INPL (Nancy), where I had joined Prof. F. Simonot-Lion's team. My first result on this topic [14] has received criticism for the fact that the inputs for such problem (probability distributions for execution times) are impossible to obtain. This feedback had made me realize that no probabilistic schedulability analysis will be understood and accepted by the community as long as no methods for obtaining probability distributions exist. At this point the first Dagstuhl seminar on scheduling (2008) gave me the

opportunity to meet Prof. Alan Burns and this was the event "provoking" in 2009 my work in the FP7 STREP project PROARTIS. At the end of this project we have proposed several important results that solve partially the problem of obtaining probability distributions for probabilistic worst-case execution times [10, 17, 65]. Therefore it is now a good timing to advance on probabilistic schedulability analysis [56] and my unpublished work on probabilistic multiprocessor scheduling [14] will soon meet the appropriate PhD student to become a published paper.

During the first Dagstuhl seminar on scheduling, discussions with Prof. Jane Liu (co-author of one of the first papers on probabilistic real-time systems) made me, also, understand that a second mandatory condition for probabilistic real-time reasoning was to establish the bases of the probabilistic worst-case reasoning. In 2013 I have provided the arguments to sustain the importance of the probabilistic worst-case reasoning and its impact on independence property. These arguments are related to the definition of probabilistic worst-case execution time as it is proposed by Edgar and Burns [21]. The paper had proposed this new concept without underlining its important impact on imposing a probabilistic worst-case reasoning. Underling this impact is probably the most important contribution of this thesis.

Organization of this document In Chapter 1 we present our contribution to the scheduling of real-time systems on multiprocessor platforms. We start the chapter by providing an exhaustive list of all results we proposed on this topic and then we detail two contributions: the predictability results and the schedulability analysis. We present the proofs of these results as their details are important. The first result on predictability is a corrected and extended version of previous results from [38]. The results on schedulability analysis provide a methodology for feasibility tests based on intervals.

In Chapter 2 we present our contribution to the definition and the analysis of probabilistic real-time systems. We start the chapter by providing an exhaustive list of all results we proposed on this topic and then we detail two contributions. The first contribution concerns the proposition of probabilistic worst-case reasoning within response time analysis of systems with multiple probabilistic parameters. The second contribution concerns the estimation of probabilistic worst-case execution times and its relation to rare events theory. It indicates that the real-time community should use the mathematical results on the tails of the probability distributions.

In Chapter 3 we present the perspectives of our results and the vision of how these results will modify the real-time community that needs to face data deluge and increased connectivity with less critical systems that will coexist within one larger system built following a mixed-critical model.

Chapter 1

Multiprocessor real-time systems

This chapter contains results on multiprocessor scheduling of real-time systems. These results are mainly obtained during my stay at Université Libre de Bruxelles in collaboration with Prof. Joël Goossens's research group.

These results belong to three main classes:

- *Predictability* of scheduling algorithms on multiprocessor architectures. After proving that previous existing results on predictability for identical platforms was incorrect, we have extended them to the case of unrelated platforms and to a more general class of scheduling algorithms. Our result allows to understand the "mandatory" properties of a scheduling algorithm for multiprocessor architectures and it indicates what algorithms are not predictable.

  J1 L. Cucu-Grosjean and J. Goossens, *"Predictability of Fixed-Job Priority Schedulers on Heterogeneous Multiprocessor Real-Time Systems"*, Information Processing Letters 110(10): 399-402, April 2010

  O1 E. Grolleau, J. Goossens and L. Cucu-Grosjean, *"On the periodic behavior of real-time schedulers on identical multiprocessor platforms"*, arxiv.org, May 2013

  The result of this class is published in J1 and it concerns memoryless algorithms. This result and the definition of memoryless algorithms are presented in Section 1.1. Recent work allowed us to extend J1 to algorithms that do not have this memoryless constraint in O1.

- *Schedulability tests* for job-level fixed-priority scheduling algorithms for periodic tasks, whatever the type of architecture (identical, uniform or unrelated). These tests were the first exact tests of the real-time literature for such general architectures. They are based on the construction of feasibility intervals,

that have the existence guaranteed by the periodicity properties of feasible schedules. This methodology was used later by other authors for different real-time multiprocessor scheduling problems (parallel tasks, scheduling simulation, task-level optimal scheduling algorithms, etc).

The following publications belong to this class of results on multiprocessor systems:

J2 L. Cucu-Grosjean and J. Goossens, *"Exact Schedulability Tests for Real-Time Scheduling of Periodic Tasks on Unrelated Multiprocessor Platforms"*, Journal of System Architecture, 57(5): 561-569, March 2011

C4 B. Miramond and L. Cucu-Grosjean *Generation of static tables in embedded memory with dense scheduling*, Conference on Design and Architectures for Signal and Image Processing (DASIP), Edinburgh, October 2010

C3 L. Idoumghar, L. Cucu-Grosjean and R. Schott, *Tabu Search Type Algorithms for the Multiprocessor Scheduling Problem*, the 10th IASTED International Conference on Artificial Intelligence and Applications (AIA), Innsbruck, February 2010

C2 Cucu L. and Goossens, J. *"Feasibility Intervals for Multiprocessor Fixed-Priority Scheduling of Arbitrary Deadline Periodic Systems "*, the 10th Design, Automation and Test in Europe (DATE), ACM Press, Nice, April 2007

C1 Cucu L. and Goossens J., *"Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors"*, the 11th IEEE International Conference on Emerging Technologies and Factory Automation, (ETFA), Prague, September 2006

The publication J2 is the most general of this class and it is an extended version of C1 and C2. The content of J2 is detailed in Section 1.2. The conferences publications C4 and C5 make use of the results of J1 in different contexts (C4 for the utilisation of genetic algorithms and C5 for the introduction of memory issues in a multiprocessor context).

- Model of *parallel tasks* for multiprocessor real-time. Historically, the first models of tasks on multiprocessor architectures forbid the parallelism of tasks without an appropriate justification. In C5 we have provided arguments in favor of parallel tasks and the first model of parallel tasks on identical processors. This model is the starting point of several current results on real-time parallel tasks. The results were published originally in C5 and J3 is an extended version of C5.

  J3 Collette S., Cucu L. and Goossens J., *"Integrating job parallelism in real-time scheduling theory"*, Information Processing Letters, 106(5): 180-187, May 2008

C5 Collette S., Cucu L. and Goossens, J. *"Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism"*, the 15th International Conference on Real-Time and Network systems (RTNS), Nancy, March 2007

Organization of the chapter This chapter has two parts. The first part (within Section 1.1) presents the results on predictability. Most of the proofs are provided as they indicate the misunderstanding within the previous existing proofs. The second part of this chapter (within Section 1.2) collect the main results on feasibility intervals for a large class of scheduling algorithms and architectures.

During this chapter we assume the notions defined below as known.

From a theoretical and a practical point of view, we can distinguish (at least) between three kinds of multiprocessor architectures (from less general to more general):

Identical parallel machines Platforms on which all processors are identical, in the sense that they have the same computing power.

Uniform parallel machines By contrast, each processor in a uniform parallel machine is characterized by its own computing capacity, a job that is executed on processor π_i of computing capacity s_i for t time units completes $s_i \times t$ units of execution.

Unrelated parallel machines In unrelated parallel machines, there is an execution rate $s_{i,j}$ associated with each job-processor pair, a job J_i that is executed on processor π_j for t time units completes $s_{i,j} \times t$ units of execution.

We consider real-time systems that are modeled by set of jobs and implemented upon a platform comprised of several *unrelated* processors. We assume that the platform

- is fully *preemptive*: an executing job may be interrupted at any instant in time and have its execution resumed later with no cost or penalty.
- allows *global* inter-processor migration: a job may begin execution on any processor and a preempted job may resume execution on the same processor as, or a different processor from, the one it had been executing on prior to preemption.
- forbids *job parallelism*: each job is executing on at most one processor at each instant in time.

The *scheduling algorithm* determines which job[s] should be executed at each time-instant. Fixed-Job Priority (FJP) schedulers assign priorities to jobs *statically* and execute the highest-priority jobs on the available processors. Dynamic Priority

(DP) schedulers assign priorities to jobs *dynamically* (at each instant of time). Popular FJP schedulers include: the Rate Monotonic (RM), the Deadline Monotonic (DM) and the Earliest Deadline First (EDF) [48]. Popular DP schedulers include: the Least Laxity First (LLF) and the EDZL [11, 58].

The specified execution requirement of job is actually an upper bound of its actual value, i.e., the worst case execution time (WCET) is provided. The actual execution requirement may vary depending on the input data, and on the system state (caches, memory, etc.). The *schedulability analysis*, determining whether all jobs *always* meet their deadlines, is designed by considering a finite number of (worst-case) scenarios (typically a single scenario) assuming that the scheduler is *predictable* with the following definition: For a predictable scheduling algorithm, one may determine an upper bound on the completion-times of jobs by analyzing the situation under the assumption that each job executes for an amount equal to the upper bound on its execution requirement; it is guaranteed that the actual completion time of jobs will be no later than this determined value.

1.1 Predictability of Fixed-Job Priority Schedulers on Heterogeneous Multiprocessor Real-Time Systems

The results presented in this section were published in [15].

In this section we extend and correct [38] by considering unrelated multiprocessor platforms and by showing that any FJP schedulers are predictable on these platforms. Ha and Liu [38] "showed" that FJP schedulers are predictable on *identical* multiprocessor platforms. However, while the result is correct, an argument used in the proof is not. Han and Park studied the predictability of the LLF scheduler for identical multiprocessors [39]. To the best of our knowledge a single work addressed heterogeneous architectures, indeed we have showed in [12] that any FJP schedulers are predictable on *uniform* multiprocessors.

1.1.1 Definitions and assumptions

Within Section 1.1 we consider that a real-time system is modelled as a finite collection of independent recurring tasks, each of which generates a potentially infinite sequence of *jobs*. Every job is characterized by a 3-tuple (r_i, e_i, d_i), i.e., by a release time (r_i), an execution requirement (e_i), and a deadline (d_i), and it is required that a job completes execution between its arrival time and its deadline.

We consider multiprocessor platforms π composed of m unrelated processors: $\{\pi_1, \pi_2, \ldots, \pi_m\}$. Execution rates $s_{i,j}$ are associated with each job-processor pair, a job J_i that is executed on processor π_j for t time units completes $s_{i,j} \times t$ units of execution. For each job J_i we assume that the associated set of processors $\pi_{n_{i,1}} > \pi_{n_{i,2}} > \cdots > \pi_{n_{i,m}}$ are ordered in a decreasing order of the execution rates relative

to the job: $s_{i,n_{i,1}} \geq s_{i,n_{i,2}} \geq \cdots \geq s_{i,n_{i,m}}$. For identical execution rates, the ties are broken arbitrarily, but consistently, such that the set of processors associated with each job is *totally* ordered. For the processor-job pair (π_j, J_i) if $s_{i,j} \neq 0$ then π_j is said to be an *eligible* processor for J_i.

Definition 1 (Schedule $\sigma(t)$)**.** *For any set of jobs $J \stackrel{\text{def}}{=} \{J_1, J_2, J_3, \ldots\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$ we define the* schedule $\sigma(t)$ *of system τ at time-instant t as $\sigma : \mathbb{N} \to \mathbb{N}^m$ where $\sigma(t) \stackrel{\text{def}}{=} (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ with*
$\sigma_j(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if there is no job scheduled on } \pi_j \text{ at time-instant } t; \\ i, & \text{if job } J_i \text{ is scheduled on } \pi_j \text{ at time-instant } t. \end{cases}$

Definition 2 (Work-conserving algorithm)**.** *An unrelated multiprocessor scheduling algorithm is said to be* work-conserving *if, at each instant, the algorithm schedules jobs on processors as follows: the highest priority (active) job J_i is scheduled on its fastest (and eligible) processor π_j. The very same rule is then applied to the remaining active jobs on the remaining available processors.*

Throughout this paper, J denotes a (finite or infinite) set of jobs: $J \stackrel{\text{def}}{=} \{J_1, J_2, J_3, \ldots\}$. We consider any FJP scheduler and without loss of generality we consider jobs in a decreasing order of priorities ($J_1 > J_2 > J_3 > \cdots$). We suppose that the actual execution time of each job J_i can be any value in the interval $[e_i^-, e_i^+]$ and we denote by J_i^+ the job defined as $J_i^+ \stackrel{\text{def}}{=} (r_i, e_i^+, d_i)$. The associated execution rates of J_i^+ are $s_{i,j}^+ \stackrel{\text{def}}{=} s_{i,j}, \forall j$. Similarly, J_i^- is the job defined from J_i as follows: $J_i^- = (r_i, e_i^-, d_i)$. Similarly, the associated execution rates of J_i^- are $s_{i,j}^- \stackrel{\text{def}}{=} s_{i,j}, \forall j$. We denote by $J^{(i)}$ the set of the i highest priority jobs (and its schedule by $\sigma^{(i)}$). We denote also by $J_-^{(i)}$ the set $\{J_1^-, \ldots, J_i^-\}$ and by $J_+^{(i)}$ the set $\{J_1^+, \ldots, J_i^+\}$ (and its schedule by $\sigma_+^{(i)}$). Note that the schedule of an ordered set of jobs using a work-conserving and FJP scheduler is unique. Let $S(J)$ be the time-instant at which the lowest priority job of J begins its execution in the schedule. Similarly, let $F(J)$ be the time-instant at which the lowest priority job of J completes its execution in the schedule.

Definition 3 (Predictable algorithm)**.** *A scheduling algorithm is said to be* predictable *if $S(J_-^{(i)}) \leq S(J^{(i)}) \leq S(J_+^{(i)})$ and $F(J_-^{(i)}) \leq F(J^{(i)}) \leq F(J_+^{(i)})$, for all $1 \leq i \leq \#J$ and for all schedulable $J_+^{(i)}$ sets of jobs.*

Definition 4 (Availability of the processors)**.** *For any ordered set of jobs J and any set of m unrelated processors $\{\pi_1, \ldots, \pi_m\}$, we define the* availability of the processors $A(J,t)$ *of the set of jobs J at time-instant t as the set of available processors: $A(J,t) \stackrel{\text{def}}{=} \{j \mid \sigma_j(t) = 0\} \subseteq \{1, \ldots, m\}$, where σ is the schedule of J.*

1.1.2 Proof from Ha and Liu [38]

The result we extend in this work is the following:

Theorem 5. *For any FJP scheduler and identical multiprocessor platform, the start time of every job is predictable, that is,* $S(J_-^{(i)}) \leq S(J^{(i)}) \leq S(J_+^{(i)})$.

We give here the first part of the original (adapted with our notations) proof of Ha and Liu, Theorem 3.1, page 165 of [38].

Proof from [38]. Clearly, $S(J^{(1)}) \leq S(J_+^{(1)})$ is true for the highest-priority job J_1. Assuming that $S(J^{(k)}) \leq S(J_+^{(k)})$ for $k < i$, we now prove $S(J^{(i)}) \leq S(J_+^{(i)})$ by contradiction. Suppose $S(J^{(i)}) > S(J_+^{(i)})$. Because we consider a FJP scheduler, every job whose release time is at or earlier than $S(J_+^{(i)})$ and whose priority is higher than J_i has started by $S(J_+^{(i)})$ according to the maximal schedule $\sigma_+^{(i)}$. From the induction hypothesis, we can conclude that every such job has started by $S(J_+^{(i)})$ in the actual schedule $\sigma^{(i)}$. Because $e_k \leq e_k^+$ for all k, in $(0, S(J_+^{(i)}))$, the total demand of all jobs with priorities higher than J_i in the maximal schedule $\sigma_+^{(i)}$ is larger than the total time demand of these jobs in the actual schedule $\sigma^{(i)}$. In $\sigma_+^{(i)}$, a processor is available at $S(J_+^{(i)})$ for J_i to start; a processor must also be available in $\sigma^{(i)}$ at or before $S(J_+^{(i)})$ on which J_i or a lower-priority job can be scheduled.

Counter-example. The use of the notion of *"total demand"* used in the original proof is not appropriate considering multiprocessor platforms. Consider, for instance, two set of jobs: $J \stackrel{\text{def}}{=} \{J_1 = J_2 = (0, 3, \infty)\}$ and $J' \stackrel{\text{def}}{=} \{J_3 = (1, 5, \infty), J_4 = (1, 1, \infty)\}$. In $(0, 2)$ the total demands of both job sets are identical (i.e., 6 time units), if we schedule the system using FJP schedulers, e.g., $J_1 > J_2$ and $J_3 > J_4$ it is not difficult to see that in the schedule of J' a processor is available at time 2 while in the schedule of J we have to wait till time-instant 3 to have available processor(s).

1.1.3 Predictability

In this section we prove our main property, the predictability of FJP schedulers on unrelated multiprocessors which is based on the following lemma.

Lemma 1.1.1. *For any schedulable ordered set of jobs J (using a FJP and work-conserving scheduler) on an arbitrary set of unrelated processors $\{\pi_1, \ldots, \pi_m\}$, we have $A(J_+^{(i)}, t) \subseteq A(J^{(i)}, t)$, for all t and all i. In other words, at any time-instant the processors available in $\sigma_+^{(i)}$ are also available in $\sigma^{(i)}$. (We consider that the sets of jobs are ordered in the same decreasing order of the priorities, i.e., $J_1 > J_2 > \cdots > J_\ell$ and $J_1^+ > J_2^+ > \cdots > J_\ell^+$.)*

Proof. The proof is made by induction by ℓ (the number of jobs). Our inductive hypothesis is the following: $A(J_+^{(k)}, t) \subseteq A(J^{(k)}, t)$, for all t and $1 \leq k \leq i$.

The property is true in the base case since $A(J_+^{(1)}, t) \subseteq A(J^{(1)}, t)$, for all t. Indeed, $S(J^{(1)}) = S(J_+^{(1)})$. Moreover J_1 and J_1^+ are both scheduled on their fastest (same) processor $\pi_{n_{1,1}}$, but J_1^+ will be executed for the same or a greater amount of time than J_1.

We will show now that $A(J_+^{(i+1)}, t) \subseteq A(J^{(i+1)}, t)$, for all t.

Since the jobs in $J^{(i)}$ have higher priority than J_{i+1}, then the scheduling of J_{i+1} will not interfere with higher priority jobs which have already been scheduled. Similarly, J_{i+1}^+ will not interfere with higher priority jobs of $J_+^{(i)}$ which have already been scheduled. Therefore, we may build the schedule $\sigma^{(i+1)}$ from $\sigma^{(i)}$, such that the jobs $J_1, J_2, \ldots, J_i$, are scheduled at the very same instants and on the very same processors as they were in $\sigma^{(i)}$. Similarly, we may build $\sigma_+^{(i+1)}$ from $\sigma_+^{(i)}$.

Note that $A(J^{(i+1)}, t)$ will contain the same available processors as $A(J^{(i)}, t)$ for all t except the time-instants at which $J^{(i+1)}$ is scheduled, and similarly $A(J_+^{(i+1)}, t)$ will contain the same available processors as $A(J_+^{(i)}, t)$ for all t except the time-instants at which $J_+^{(i+1)}$ is scheduled. From the inductive hypothesis we have $A(J_+^{(i)}, t) \subseteq A(J^{(i)}, t)$, we will consider time-instant t, from r_{i+1} to the completion of J_{i+1} (which is actually not after the completion of J_{i+1}^+, see below for a proof), we distinguish between four cases:

1. $A(J^{(i)}, t) = A(J_+^{(i)}, t) = 0$: in both situations no processor is available. Therefore, both jobs, J_{i+1} and J_{i+1}^+, do not progress and we obtain $A(J^{(i+1)}, t) = A(J_+^{(i+1)}, t)$. The progression of J_{i+1} is identical to J_{i+1}^+.

2. $A(J^{(i)}, t) \neq \emptyset$ and $A(J_+^{(i)}, t) = \emptyset$: if an eligible processor exists, J_{i+1} progress on an available processor in $A(J^{(i)}, t)$ not available in $A(J_+^{(i)}, t)$, J_{i+1}^+ does not progress. Consequently, $A(J_+^{(i+1)}, t) \subseteq A(J^{(i+1)}, t)$ and the progression of J_{i+1} is strictly larger than J_{i+1}^+.

3. $A(J^{(i)}, t) = A(J_+^{(i)}, t) \neq \emptyset$: if an eligible processor exists, J_{i+1} and J_{i+1}^+ progress on the same processor. Consequently, $A(J_+^{(i+1)}, t) = A(J^{(i+1)}, t)$ and the progression of J_{i+1} is identical to J_{i+1}^+.

4. $A(J^{(i)}, t) \neq A(J_+^{(i)}, t) \neq \emptyset$: if the faster processor in $A(J^{(i)}, t)$ and $A(J_+^{(i)}, t)$ is the same see previous case of the proof (case 3); otherwise J_{i+1} progress on a *faster* processor than J_{i+1}^+, that processor is not available in $A(J_+^{(i)}, t)$, consequently a slower processor remains idle in $A(J^{(i)}, t)$ but busy in $A(J_+^{(i)}, t)$. Consequently, $A(J_+^{(i+1)}, t) \subseteq A(J^{(i+1)}, t)$ and the progression of J_{i+1} is larger than J_{i+1}^+.

Therefore, we showed that $A(J_+^{(i+1)}, t) \subseteq A(J^{(i+1)}, t)\ \forall t$, from r_{i+1} to the completion of J_{i+1} and that J_{i+1} does not complete after J_{i+1}^+. For the time-instant after the completion of J_{i+1} the property is trivially true by induction hypothesis. □

Theorem 6. *FJP schedulers are predictable on unrelated platforms.*

Proof. In the framework of the proof of Lemma 1.1.1 we actually showed extra properties which imply that FJP schedulers are predictable on unrelated platforms: (i) J_{i+1} completes not after J_{i+1}^+ and (ii) J_{i+1} can be scheduled either at the very same instants as J_{i+1}^+ or may progress during *additional* time-instants (case (2) of the proof) these instants may precede the time-instant where J_{i+1}^+ commences its execution. □

Since multiprocessor real-time scheduling researchers pay attention to uniform platforms (at least nowadays), we find convenient to emphasize the following corollary:

Corollary 7. *FJP schedulers are predictable on uniform platforms.*

1.2 Exact Schedulability Tests for Periodic Tasks on Unrelated Multiprocessor Platforms

The results presented in this section were published in [16].

In this section, we study the global scheduling of periodic task systems on unrelated multiprocessor platforms. We first show two general properties which are well-known for uniprocessor platforms and which are also true for unrelated multiprocessor platforms: *(i)* under few and not so restrictive assumptions, we prove that feasible schedules of periodic task systems are periodic starting from some point in time with a period equal to the least common multiple of the task periods and *(ii)* for the specific case of synchronous periodic task systems, we prove that feasible schedules repeat from their origin. We then characterize, for task-level fixed-priority schedulers and for asynchronous constrained or arbitrary deadline periodic task models, *upper bounds* of the first time-instant where the schedule repeats. For task-level fixed-priority schedulers, based on the upper bounds and the predictability property, we provide *exact* schedulability tests for asynchronous constrained or arbitrary deadline periodic task sets.

The problem of scheduling periodic task systems on multiprocessors was originally studied in [47]. Recent studies provide a better understanding of this scheduling problem and provide the first solutions, e.g., [9] presents a categorization of real-time multiprocessor scheduling problems and [19] an up-to-date state of the art. To the best of our knowledge, a single work [5] provides *exact* schedulability tests for the global scheduling of periodic systems on multiprocessors. Baker and Cirinei present a test for global preemptive priority-based scheduling of sporadic tasks on identical processors. Our work differs for several reasons: (i) we extend the model by considering unrelated platforms, and (ii) we provide a *schedulability interval* and the periodic characterization of the schedule. Such characterization is not straightforward since we know that not all uniprocessor schedulability results

or arguments are true for multiprocessor scheduling. For instance, the synchronous case (i.e., considering that all tasks start their execution synchronously) is not the worst case on multiprocessors anymore [33]. Another example is the fact that the first busy period (see [45] for details) does not provide a schedulability interval on multiprocessors (see [33] for such counter-examples). By schedulability interval we mean a finite interval such that, if no deadline is missed while only considering requests within this interval then no deadline will ever be missed.

The tests that we present have, for a set of n tasks, a time complexity $\mathcal{O}(\text{lcm}\{T_1, T_2, \ldots, T_n\})$, which is unfortunately also the case of most schedulability tests for *simpler uni*processor scheduling problems.

1.2.1 Definitions and assumptions

We consider the preemptive global scheduling of periodic task systems. A system τ is composed of n periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, where each task is characterized by a period T_i, a relative deadline D_i, an execution requirement C_i and an offset O_i. Such a periodic task generates an infinite sequence of jobs, with the k^{th} job arriving at time-instant $O_i + (k-1)T_i$ $(k = 1, 2, \ldots)$, having an execution requirement of C_i units, and a deadline at time-instant $O_i + (k-1)T_i + D_i$. It is important to note that we assume first that each job of the same task τ_i has the same execution requirement C_i; we relax this assumption then by showing that our analysis is *robust*.

We will distinguish between *implicit deadline* systems where $D_i = T_i, \forall i$; *constrained deadline* systems where $D_i \leq T_i, \forall i$ and *arbitrary deadline* systems where there is no relation between the deadlines and the periods.

In some cases, we will consider the more general problem of scheduling a set of jobs, each job J_j is characterized by a release time r_j, an execution requirement e_j and an absolute deadline d_j. The job J_j must execute for e_j time units over the interval $[r_j, d_j)$. A job becomes *active* from its release time to its completion.

A periodic system is said to be *synchronous* if there is a time-instant where all tasks make a new request simultaneously, i.e., $\exists t, k_1, k_2, \ldots k_n$ such that $\forall i : t = O_i + k_i T_i$ (see [30] for details). Without loss of generality, we consider $O_i = 0, \forall i$ for synchronous systems. Otherwise the system is said to be *asynchronous*.

We denote by $\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_i\}$, by $O^i_{\max} \stackrel{\text{def}}{=} \max\{O_1, \ldots, O_i\}$, by $O_{\max} \stackrel{\text{def}}{=} O^n_{\max}$, by $P_0 \stackrel{\text{def}}{=} 0$, $P_i \stackrel{\text{def}}{=} \text{lcm}\{T_1, \ldots, T_i\}$ $(0 < i \leq n)$ and by $P \stackrel{\text{def}}{=} P_n$. In the following, the quantity P is called the task set *hyper-period*.

We consider multiprocessor platforms π composed of m unrelated processors (or one of its particular cases: uniform and identical platforms): $\{\pi_1, \pi_2, \ldots, \pi_m\}$. Execution rates $s_{i,j}$ are associated with each task-processor pair. A task τ_i that is executed on processor π_j for t time units completes $s_{i,j} \times t$ units of execution. For each task τ_i we assume that the associated set of processors $\pi_{n_{i,1}} > \pi_{n_{i,2}} > \cdots > \pi_{n_{i,m}}$ are ordered in a decreasing order of the execution rates relative to the task: $s_{i,n_{i,1}} \geq s_{i,n_{i,2}} \geq \cdots \geq s_{i,n_{i,m}}$. For identical execution rates, the ties are broken arbitrarily, but consistently, such that the set of processors associated with each task

is *totally* ordered. Consequently, the *fastest* processor relative to task τ_i is $\pi_{n_{i,1}}$, i.e., the first processor of the ordered set associated with the task. Moreover, for a task τ_i in the following we consider that a processor π_a is *faster* than π_b (relative to its associated set of processors) if $\pi_a > \pi_b$ even if we have $s_{i,a} = s_{i,b}$. For the processor-task pair (π_j, τ_i) if $s_{i,j} \neq 0$ then π_j is said to be an *eligible* processor for τ_i. Note that these concepts and definitions can be trivially adapted to the scheduling of jobs on unrelated platforms.

In this work we consider a discrete model, i.e., the characteristics of the tasks and the time are integers. Moreover, in the following we will illustrate main concepts and definitions using the following system.

Example 1. *Let τ be a system of three periodic tasks*

	O_i	C_i	D_i	T_i
τ_1	0	2	6	6
τ_2	5	4	6	6
τ_3	4	5	7	6

Let π be a multiprocessor platform of 2 unrelated processors π_1 and π_2. We have $s_{1,1} = 1, s_{1,2} = 2, s_{2,1} = 2, s_{2,2} = 1, s_{3,1} = 2$ and $s_{3,2} = 1$. Note that according to our definition above and regarding task τ_1 we have that $\pi_2 > \pi_1$.

The notions of the state of the system and the schedule are as follows.

Definition 1 (State of the system $\theta(t)$)**.** *For any arbitrary deadline task system $\tau = \{\tau_1, \ldots, \tau_n\}$ we define the* state $\theta(t)$ *of the system τ at time-instant t as $\theta : \mathbb{N} \to (\mathbb{Z} \times \mathbb{N} \times \mathbb{N})^n$ with $\theta(t) \stackrel{\text{def}}{=} (\theta_1(t), \theta_2(t), \ldots, \theta_n(t))$ where*

$$\theta_i(t) \stackrel{\text{def}}{=} \begin{cases} (-1, x, 0), & \text{if no job of task } \tau_i \text{ was activated before or at } t. \text{ In this case, } x \text{ time units remain until the first activation of } \tau_i. \text{ We have } 0 < x \leq O_i. \\ (y, z, u), & \text{otherwise. In this case, } y \text{ denotes the number of active jobs of } \tau_i \text{ at } t, \ z \text{ denotes the time elapsed at time-instant } t \text{ since the arrival of the oldest active job of } \tau_i, \text{ and } u \text{ denotes the amount that the oldest active job has executed for. If there is no active job of } \tau_i \text{ at } t, \text{ then } u \text{ is } 0. \end{cases}$$

Note that at any time-instant t several jobs of the same task might be active and we consider that the oldest job is scheduled first, i.e., the FIFO rule is used to serve the various jobs of a given task.

Definition 2 (Schedule $\sigma(t)$)**.** *For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$ we define the* schedule $\sigma(t)$ *of system τ at time-instant t*

as $\sigma : \mathbb{N} \to \{0, 1, \ldots, n\}^m$ *where* $\sigma(t) \stackrel{\text{def}}{=} (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ *with*

$$\sigma_j(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if there is no task scheduled on } \pi_j \\ & \text{at time-instant } t; \\ i, & \text{if task } \tau_i \text{ is scheduled on } \pi_j \text{ at time-instant } t. \end{cases} \qquad \forall 1 \leq j \leq m.$$

A system τ is said to be *schedulable* on a multiprocessor platform if there is at least one feasible schedule, i.e., a schedule in which all tasks meet their deadlines. If A is an algorithm which schedules τ on a multiprocessor platform to meet its deadlines, then the system τ is said to be *A-schedulable*.

Note that for a feasible schedule from Definition 1 we have $0 \leq y \leq \lceil \frac{D_i}{T_i} \rceil$, $0 \leq z < T_i \cdot \lceil \frac{D_i}{T_i} \rceil$ and $0 \leq u < C_i$.

The scheduling algorithms considered in this paper are work-conserving (see Definition 2) and *deterministic* with the following definition:

Definition 3 (Deterministic algorithm)**.** *A scheduling algorithm is said to be* deterministic *if it generates a unique schedule for any given set of jobs.*

Note that we assume in this work that no two jobs/tasks share the same priority. It follows by Definition 2 that a processor π_j can be idle and a job J_i can be active at the same time if and only if $s_{i,j} = 0$.

Moreover, we will assume that the decision of the scheduling algorithm at time t is not based on the past, nor on the actual time t, but only on the characteristics of active tasks and on the state of the system at time t. More formally, we consider *memoryless* schedulers.

Definition 4 (Memoryless algorithm)**.** *A scheduling algorithm is said to be* memoryless *if the scheduling decision it made at time t depends only on the characteristics of active tasks and on the current state of the system, i.e., on* $\theta(t)$*.*

Consequently, for memoryless and deterministic schedulers we have the following property:

$$\forall t_1, t_2 \text{ such that } \theta(t_1) = \theta(t_2) \text{ then } \sigma(t_1) = \sigma(t_2).$$

Note that popular real-time schedulers, e.g., EDF and Deadline Monotonic (DM), are memoryless: the priority of each task is only based on its absolute (EDF) or relative (DM) deadline.

In the following, we will distinguish between two kinds of schedulers:

Definition 5 (Task-level fixed-priority)**.** *A scheduling algorithm is a* task-level fixed-priority *algorithm if it assigns the priorities to the tasks beforehand; at run-time each job priority corresponds to its task priority.*

Definition 6 (Job-level fixed-priority)**.** *A scheduling algorithm is a* job-level fixed-priority *algorithm if and only if it satisfies the condition that: for every pair of jobs* J_i *and* J_j*, if* J_i *has higher priority than* J_j *at some time-instant, then* J_i *always has higher priority than* J_j*.*

Popular task-level fixed-priority schedulers include the RM and the DM algorithms; popular job-level fixed-priority schedulers include the EDF algorithm, see [48] for details.

Definition 7 (Job J_i^k of a task τ_i)**.** *For any task τ_i, we define J_i^k to be the k^{th} job of task τ_i, which becomes active at time-instant $R_i^k \stackrel{\text{def}}{=} O_i + (k-1)T_i$. For two tasks τ_i and τ_j, by $J_i^k > J_j^\ell$ we mean that the job J_i^k has a higher priority than the job J_j^ℓ.*

Definition 8 (Executed time $\epsilon_i^k(t)$ for a job J_i^k)**.** *For any task τ_i, we define $\epsilon_i^k(t)$ to be the amount of time already executed for J_i^k in the interval $[R_i^k, t)$.*

We now introduce the availability of the processors for any schedule $\sigma(t)$.

Definition 9 (Availability of the processors $a(t)$)**.** *For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$ we define the* availability of the processors *$a(t)$ of system τ at time-instant t as the set of available processors $a(t) \stackrel{\text{def}}{=} \{j \mid \sigma_j(t) = 0\} \subseteq \{1, \ldots, m\}$.*

In Example 1 for a preemptive task-level fixed-priority algorithm with τ_1 the highest priority task and τ_3 the lowest priority task, we obtain the schedule provided in Figure 1.1. In this figure the execution on the first processor is indicated by black boxes. The releases of jobs are indicated by $\uparrow$ and the deadlines by $\downarrow$.

For instance we have $\theta_2(2) = (-1, 3, 0)$, $\theta_1(3) = (3, 0, 0)$ and $\theta_3(6) = (2, 1, 3)$. Moreover $\sigma(6) = (2, 1)$ and $\sigma(7) = (3, 0)$. Concerning task τ_1, its job J_1^2 becomes active at $R_1^2 = 6$ and this job has a higher priority than the job J_2^1 of task τ_2. For this schedule $a(1) = \{1\}$ and $a(6) = \emptyset$.

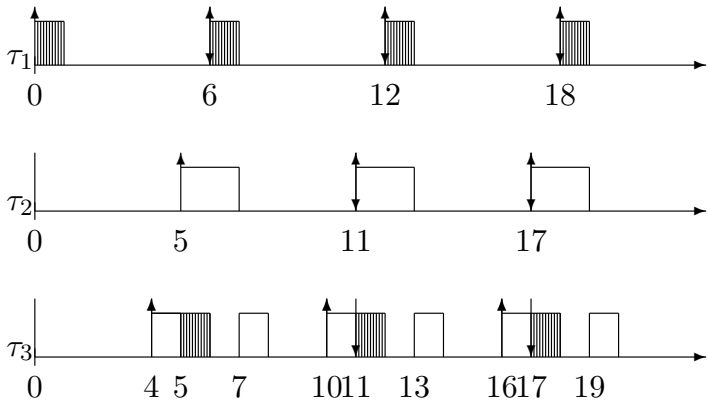


Figure 1.1: Schedule (obtained using a task-level fixed-priority scheduling) for the task set of Example 1

1.2.2 Periodicity of feasible schedules

We assume in this section that each job of the same task τ_i has the same execution requirement C_i. We will relax this assumption in Section 1.2.3.

This section contains four parts. In each part of this section we give results concerning the periodicity of feasible schedules. By periodicity (assuming that the period is γ) of a schedule σ, we mean there is a time-instant t_0 and an interval length γ such that $\sigma(t) = \sigma(t+\gamma), \forall t \geq t_0$. For instance, in Figure 1.1 the schedule is periodic from $t = 5$ with a period equal to 6.

The first part of this section provides periodicity results for a general scheduling algorithm class: deterministic, memoryless and work-conserving schedulers. The second part of this section provides periodicity results for synchronous periodic task systems. The third and the fourth parts of this section present periodicity results for task-level fixed-priority scheduling algorithms for constrained and arbitrary deadline systems, respectively.

Periodicity of deterministic, memoryless and work-conserving scheduling algorithms

We show that feasible schedules of periodic task systems obtained using deterministic, memoryless and work-conserving algorithms are periodic starting from some point. Moreover, we prove that the schedule repeats with a period equal to P for a sub-class of such schedulers. Based on that property, we provide two interesting corollaries for preemptive task-level fixed-priority algorithms (Corollary 1) and for preemptive deterministic EDF [1] (Corollary 2).

We first present two basic results in order to prove Theorem 1.

Lemma 1. *For any deterministic and memoryless algorithm A, if an asynchronous arbitrary deadline system τ is A-schedulable on m unrelated processor platform π, then the feasible schedule of τ on π obtained using A is periodic with a period* divisible *by P.*

Proof. First note that from any time instant $t_0 \geq O_{\max}$ all tasks are released, and the configuration $\theta_i(t_0)$ of each task is a triple of finite integers (α, β, γ) with $\alpha \in \{0, 1, \ldots, \lceil \frac{D_i}{T_i} \rceil\}$, $0 \leq \beta < \max_{1 \leq i \leq n}(T_i \times \lceil \frac{D_i}{T_i} \rceil)$ and $0 \leq \gamma < \max_{1 \leq i \leq n} C_i$. Therefore, there is a finite number of different system states, hence we can find two distinct instants t_1 and t_2 $(t_2 > t_1 \geq t_0)$ with the same state of the system $(\theta(t_1) = \theta(t_2))$. The schedule repeats from that instant with a period dividing $t_2 - t_1$, since the scheduler is deterministic and memoryless.

Note that, since the tasks are periodic, the arrival pattern of jobs repeats with a period equal to P from $O_{\max}$.

We now prove by contradiction that $t_2 - t_1$ is necessarily a multiple of P. We suppose that $\exists k_1 < k_2 \in \mathbb{N}$ such that $t_i = O_{\max} + k_i P + \Delta_i, \forall i \in \{1, 2\}$ with

[1] i.e., the method chosen to break deadline ties must be deterministic.

$\Delta_1 \neq \Delta_2$, $\Delta_1, \Delta_2 \in [0, P)$ and $\theta(t_1) = \theta(t_2)$. This implies that there are tasks for which the time elapsed since the last activation at t_1 and the time elapsed since the last activation at t_2 are not equal. But this is in contradiction with the fact that $\theta(t_1) = \theta(t_2)$. Consequently, Δ_1 must be equal to Δ_2 and, thus, we have $t_2 - t_1 = (k_2 - k_1)P$. □

For a sub-class of schedulers, we will show that the period of the schedule is P and we provide first a definition (inspired from [32]):

Definition 10 (Request-dependent scheduler)**.** *A scheduler is said to be* request-dependent *when* $\forall i, j, k, \ell : J_i^{k+h_i} > J_j^{\ell+h_j}$ *if and only if* $J_i^k > J_j^\ell$*, where* $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.

Note that Definition 10 (request-dependent) is stronger than Definition 6 (job-level fixed-priority) ; informally speaking Definition 10 requires that the same total order is used *each* hyper-period between "corresponding" jobs (J_i^k "corresponds" to $J_i^{k+h_i}$). For instance in Example 1 for the schedule of Figure 1.1 we have $J_1^1 > J_2^1$ and $J_1^3 > J_2^3$ because the hyper-period is 6.

The next lemma extends results given for arbitrary deadline task systems in the *uniprocessor* case (see [29], p. 55 for details).

Lemma 2. *For any preemptive, job-level fixed-priority and request-dependent algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O_{max}$ and k such that $R_i^k \leq t \leq R_i^k + D_i$, if there is no deadline missed up to time $t + P$, then $\epsilon_i^k(t) \geq \epsilon_i^{k+h_i}(t + P)$ with $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.*

Proof. Note first that the function $\epsilon_i^k(\cdot)$ is a non-decreasing discrete[2] step function with $0 \leq \epsilon_i^k(t) \leq C_i, \forall t$ and $\epsilon_i^k(R_i^k) = 0 = \epsilon_i^{k+h_i}(R_i^{k+h_i}), \forall k$.

Note, also that, since the tasks are periodic, the arrival pattern of jobs repeats with a period equal to P from O_{max}.

The proof is made by contradiction. Thus, we assume that a *first* time-instant t exists such that there are j and k with $R_j^k \leq t \leq R_j^k + D_j$ and $\epsilon_j^k(t) < \epsilon_j^{k+h_j}(t + P)$. This assumption implies that:

1. either there is a time-instant t' with $R_j^k \leq t' < t$ such that $J_j^{k+h_j}$ is scheduled at $t'+P$ while J_j^k is not scheduled at t'. We obtain that there is at least one job $J_\ell^{k_\ell+h_\ell}$ of a task τ_ℓ with $\ell \in \{1, 2, \ldots, n\}$ that is not scheduled at $t' + P$, while $J_\ell^{k_\ell}$ is scheduled at t' ($h_\ell \stackrel{\text{def}}{=} \frac{P}{T_\ell}$). This implies (since the tasks are scheduled according to a request-dependent algorithm, the arrival pattern of jobs repeats with a period equal to P from O_{max} and there is no deadline missed) that $J_\ell^{k_\ell}$ has not finished its execution before t', but $J_\ell^{k_\ell+h_\ell}$ has finished its execution

[2]We assume that the time unit chosen is small enough such that all execution requirements/deadlines are natural numbers.

before $t' + P$. Therefore, we have $\epsilon_\ell^{k_\ell+h_\ell}(t' + P) = C_\ell$, which implies that $\epsilon_\ell^{k_\ell}(t') < \epsilon_\ell^{k_\ell+h_\ell}(t' + P)$. This last relation is in contradiction with the fact that t is the first such time instant.

2. or there is at least one time-instant $t''+P < t+P$ when $J_j^{k+h_j}$ is scheduled on a faster processor than the processor on which J_j^k is scheduled at t''. This implies (since the tasks are scheduled according to a request-dependent algorithm) that there is, at least, one job $J_\ell^{k_\ell+h_\ell}$ of a task τ_ℓ with $\ell \in \{1, 2, \ldots, n\}$ that is not scheduled at $t'' + P$, while $J_\ell^{k_\ell}$ is scheduled at t'' ($h_\ell \stackrel{\text{def}}{=} \frac{P}{T_\ell}$). As proved in the first case, this situation leads us to a contradiction with our assumption.

□

For the task set of Example 1 and the schedule of Figure 1.1, we have from Lemma 2 that for $t = 6$ and $k = 1$, $\epsilon_3^1(6) = 3 = \epsilon_3^2(12)$ and no deadline missed up to 12.

Theorem 1. *For any preemptive job-level fixed-priority and request-dependent algorithm A and any A-schedulable asynchronous arbitrary deadline system τ on m unrelated processors the schedule is periodic with a period equal to P.*

Proof. By Lemma 1 we have $\exists t_i = O_{\max} + k_i P + d, \forall i \in \{1, 2\}$ with $0 \leq d < P$ such that $\theta(t_1) = \theta(t_2)$. We know also that the arrivals of task jobs repeat with a period equal to P from $O_{\max}$. Therefore, for all time-instants $t_1 + kP$, $\forall k < k_2 - k_1$ (i.e. $t_1 + kP < t_2$), the time elapsed since the last activation at $t_1 + kP$ is the same for all tasks. Moreover since $\theta(t_1) = \theta(t_2)$ we have $\epsilon_i^{\ell_i}(t_1) = \epsilon_i^{\ell_i+\frac{(k_2-k_1)P}{T_i}}(t_2)$ with $\ell_i \stackrel{\text{def}}{=} \lceil \frac{t_1-O_i}{T_i} \rceil$, $\forall i$. But by Lemma 2 we also have $\epsilon_i^{\ell_i}(t_1) \geq \epsilon_i^{\ell_i+\frac{P}{T_i}}(t_1 + P) \geq \cdots \geq \epsilon_i^{\ell_i+\frac{(k_2-k_1)P}{T_i}}(t_2), \forall i$. Consequently we obtain $\theta(t_1) \geq \theta(t_1 + P) \geq \cdots \geq \theta(t_2)$ and $\theta(t_1) = \theta(t_2)$ which [3] implies that $\theta(t_1) = \theta(t_1 + P) = \cdots = \theta(t_2)$. □

Corollary 1. *For any preemptive task-level fixed-priority algorithm A, if an asynchronous arbitrary deadline system τ is A-schedulable on m unrelated processors, then the schedule is periodic with a period equal to P.*

Proof. The result is a direct consequence of Theorem 1, since task-level fixed-priority algorithms are job-level fixed-priority and request-dependent schedulers. □

Corollary 2. *A feasible schedule obtained using deterministic request-dependent global EDF on m unrelated processors of an asynchronous arbitrary deadline system τ is periodic with a period equal to P.*

[3] where $\theta(t) \geq \theta(t')$ means that, for each task, the system state at time t and t' are identical regarding n_1 and t_2, moreover if $n_1 > 0$ we must have that the time elapsed for the oldest active request of the task at time t is not shorter than the time elapsed for the oldest active request of the task at time t'.

Proof. The result is a direct consequence of Theorem 1, since EDF is a job-level fixed-priority scheduler. □

The particular case of synchronous arbitrary deadline periodic systems

In this section, we deal with the periodicity of feasible schedules of *synchronous* arbitrary deadline periodic systems. Using the results obtained for deterministic, memoryless and work-conserving algorithms, we study synchronous arbitrary deadline periodic systems and the periodicity of feasible schedules of these systems under preemptive task-level fixed-priority scheduling algorithms.

In the following, and without loss of generality, we consider the tasks ordered in decreasing order of their priorities $\tau_1 > \tau_2 > \cdots > \tau_n$.

Lemma 3. *For any preemptive task-level fixed-priority algorithm A and for any synchronous arbitrary deadline system τ on m unrelated processors, if no deadline is missed in the time interval $[0, P)$ and if $\theta(0) = \theta(P)$, then the schedule of τ is periodic with a period P that begins at time-instant 0.*

Proof. Since at time-instants 0 and P the system is in the same state, i.e. $\theta(0) = \theta(P)$, then at time-instants 0 and P a preemptive task-level fixed-priority algorithm will make the same scheduling decision and the scheduled repeats from 0 with a period equal to P. □

Theorem 2. *For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ on m unrelated processors, if all deadlines are met in $[0, P)$ and $\theta(0) \neq \theta(P)$, then τ is not A-schedulable.*

Proof. The proof is provided in [16]. □

Corollary 3. *For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ on m unrelated processors, if τ is A-schedulable, then the schedule of τ obtained using A is periodic with a period P that begins at time-instant 0.*

Proof. Since τ is A-schedulable, we know by Theorem 2 that $\theta(0) = \theta(P)$. Moreover, a deterministic and memoryless scheduling algorithm will make the same scheduling decision at those instants. Consequently, the schedule repeats from its origin with a period of P. □

Task-level fixed-priority scheduling of asynchronous constrained deadline systems

In this section we describe another important result: any feasible schedule on m unrelated processors of asynchronous constrained deadline systems, obtained using preemptive task-level fixed-priority algorithms, is periodic from some point (Theorem 3) and we characterize that point.

Without loss of generality, we consider the tasks ordered in a decreasing order of their priorities $\tau_1 > \tau_2 > \cdots > \tau_n$.

Theorem 3. *For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous constrained deadline system τ on m unrelated processors, the schedule is periodic with a period P_n from time-instant S_n where S_i is defined inductively as follows:*

- $S_1 \stackrel{\text{def}}{=} O_1$;
- $S_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{S_{i-1}-O_i}{T_i} \rceil T_i\}, \forall i \in \{2, 3, \ldots, n\}$.

Proof. The proof is provided in [16]. □

Example 2. *Let τ be a system of three periodic tasks*

	O_i	C_i	D_i	T_i
τ_1	0	2	6	6
τ_2	5	4	6	6
τ_3	4	5	6	6

Let π be a multiprocessor platform of 2 unrelated processors π_1 and π_2. We have $s_{1,1} = 1, s_{1,2} = 2, s_{2,1} = 2, s_{2,2} = 1, s_{3,1} = 2$ and $s_{3,2} = 1$.

Using a task-level fixed-priority scheduling algorithm with τ_1 the highest priority, we obtain the schedule provided in Figure 1.2. For this schedule and the notations of Theorem 3, we have $S_1 = 0$, $S_2 = 5$ and $S_3 = 10$. In Figure 1.2 the execution on the first processor is indicated by black boxes.

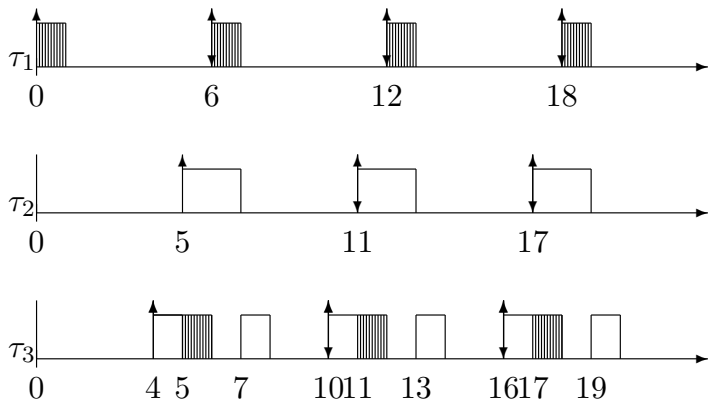


Figure 1.2: Schedule (obtained using a task-level fixed-priority scheduling) for the task set of Example 2

Task-level fixed-priority scheduling of asynchronous arbitrary deadline systems

In this section we present another important result: any feasible schedule on m unrelated processors of asynchronous arbitrary deadline systems, obtained using preemptive task-level fixed-priority algorithms, is periodic from some point (Theorem 4).

Corollary 4. *For any preemptive task-level fixed-priority algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O_i$ and k such that $R_i^k \leq t \leq R_i^k + D_i$, if there is no deadline missed up to time $t + P$, then $\epsilon_i^k(t) \geq \epsilon_i^{k+h_i}(t + P)$ with $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.*

Proof. This result is a direct consequence of Lemma 2 since preemptive task-level fixed-priority algorithms are job-level fixed-priority and request-dependent schedulers. □

In the following corollary we use the notation $\theta_i(t) = (\alpha_i(t), \beta_i(t), \gamma_i(t)), \forall \tau_i$.

Corollary 5. *For any preemptive task-level fixed-priority algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O_{\max}^i$, if there is no deadline missed up to time $t + P$, then either $(\alpha_i(t) < \alpha_i(t + P))$ or $[\alpha_i(t) = \alpha_i(t + P)$ and $\gamma_i(t) \geq \gamma_i(t + P)]$.*

Proof. The proof is provided in [16]. □

Lemma 4. *For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous arbitrary deadline system τ of n tasks on m unrelated processors, let $\sigma^{(i)}$ the schedule obtained by considering only the task subset $\tau^{(i)}$. Moreover, let the set of $\{\widehat{S}_1, \cdots, \widehat{S}_n\}$ be defined inductively as follows:*

- $\widehat{S}_1 \stackrel{\text{def}}{=} O_1$
- $\widehat{S}_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{\widehat{S}_{i-1} - O_i}{T_i} \rceil T_i\} + P_i, \qquad (i > 1)$

If $\theta_{i+1}(\widehat{S}_{i+1}) \neq \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$, then $\nexists t \in [\widehat{S}_{i+1}, \widehat{S}_{i+1} + P_{i+1})$ such that at t there is at least one available (and eligible) processor for τ_{i+1} in $\sigma^{(i)}$ and no job of τ_{i+1} is scheduled at t in $\sigma^{(i+1)}$.

Proof. The proof is made by contradiction and it is provided in [16]. □

Theorem 4. *For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous arbitrary deadline system τ on m unrelated processors, the schedule is periodic with a period P_n from time-instant $\widehat{S}_n$ where $\widehat{S}_n$ are defined inductively as follows:*

- $\widehat{S}_1 \stackrel{\text{def}}{=} O_1$

- $\widehat{S}_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{\widehat{S}_{i-1} - O_i}{T_i} \rceil T_i\} + P_i, \qquad (i > 1)$

Proof. The proof is made by induction and it is provided in [16]. □

1.2.3 Exact schedulability tests

In the previous sections, we assumed that the execution requirement of each task is constant while the designer actually only knows an upper bound on the actual execution requirement, i.e., the worst case execution time (WCET). Consequently, we need tests that are *robust* relatively to the variation of the execution times up to a worst case value. More precisely, we need *predictable* schedulers on the considered platforms.

In order to provide exact schedulability tests for various kind schedulers and platforms we use the predictability results provided in Section 1.1.

We introduce and formalize the notion of the *schedulability interval* necessary to provide the exact schedulability tests:

Definition 11 (Schedulability interval)**.** *For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$, the* schedulability interval *is a finite interval such that if no deadline is missed while considering only requests within this interval then no deadline will ever be missed.*

Asynchronous constrained deadline systems and task-level fixed-priority schedulers

Now we have the material to define an exact schedulability test for asynchronous constrained deadline periodic systems.

Corollary 6. *For any preemptive task-level fixed-priority algorithm A and for any asynchronous constrained deadline system τ on unrelated multiprocessors, τ is A-schedulable if all deadlines are met in $[0, S_n + P)$ and $\theta(S_n) = \theta(S_n + P)$, where S_i is defined inductively in Theorem 3. Moreover, for every task τ_i only the deadlines in the interval $[S_i, S_i + \text{lcm}\{T_j \mid j \leq i\})$ have to be checked.*

Proof. Corollary 6 is a direct consequence of Theorem 3 and Theorem 6, since task-level fixed-priority algorithms are job-level fixed-priority schedulers. □

The schedulability test given by Corollary 6 may be improved as it was done in the uniprocessor case [31]. The uniprocessor proof is also true for multiprocessor platforms since it does not depend on the number of processors, nor on the kind of platforms but on the availability of the processors.

Theorem 5 ([31])**.** *Let X_i be inductively defined by $X_n = S_n, X_i = O_i + \lfloor \frac{X_{i+1} - O_i}{T_i} \rfloor T_i$ $(i \in \{n-1, n-2, \ldots, 1\}$; thus τ is A-schedulable if and only if all deadlines are met in $[X_1, S_n + P)$ and if $\theta(S_n) = \theta(S_n + P)$.*

Asynchronous arbitrary deadline systems and task-level fixed-priority policies

Now we have the material to define an exact schedulability test for asynchronous arbitrary deadline periodic systems.

Corollary 7. *For any preemptive task-level fixed-priority algorithm A and for any asynchronous arbitrary deadline system τ on m unrelated processors, τ is A-schedulable if and only if all deadlines are met in $[0, \widehat{S}_n + P)$ and $\theta(\widehat{S}_n) = \theta(\widehat{S}_n + P)$, where $\widehat{S}_i$ are defined inductively in Theorem 4.*

Proof. Corollary 7 is a direct consequence of Theorem 4 and Theorem 6, since task-level fixed-priority algorithms are job-level fixed-priority schedulers. □

Note that the length of our (schedulability) interval is proportional to P (the least common multiple of the periods) which is unfortunately also the case of most schedulability intervals for the *simpler uni*processor scheduling problem (and for identical platforms or simpler task models). In practice, the periods are often *harmonics*, which limits the term P.

The particular case of synchronous arbitrary deadline periodic systems

In this section we present exact schedulability tests in the particular case of synchronous arbitrary deadline periodic systems.

Corollary 8. *For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ, τ is A-schedulable on m unrelated processors if and only if all deadlines are met in the interval $[0, P)$ and $\theta(0) = \theta(P)$.*

Chapter 2

Probabilistic real-time systems

This chapter concerns probabilistic and statistical approaches for real-time systems. The first results on probabilistic real-time systems published have been published in the early 90's. In our opinion the limitations of these first results came from the reasoning on average that cannot provide real-time guarantees (see Section 2.1.1). The current breakthrough of the probabilistic approaches within the real-time community have been done following three steps:

- Detection of *misunderstanding within probabilistic real-time models*. The independence between tasks is a property of real-time systems that is often used for its basic results. Any complex model takes into account different dependences caused by sharing resources other than the processor. On another hand, the probabilistic operations require, generally, the (probabilistic) independence between the random variables describing some parameters of a probabilistic real-time system. The main (original) criticism to probabilistic is based on this hypothesis of independence judged too restrictive to model real-time systems. In reality the two notions of independence are different. Providing arguments to underline this confusion was the main topic of the following invited talks:

  - L. Cucu-Grosjean, *"Probabilistic real-time systems"*, Keynote of the 21st International Conference on Real-time Networks and Systems (RTNS), October 2013
  - L. Cucu-Grosjean, *"Probabilistic real-time scheduling*, Ecole Temps Réel (ETR2013), August 2013
  - L. Cucu-Grosjean, *"Independence - a misunderstood property of and for (probabilistic) real-time systems*, the 60th Anniversary of A. Burns, York, March 2013

- *Probabilistic worst-case reasoning*. Guaranteeing real-time constraints require worst-case reasoning in order to provide safe solutions. We have proposed

such reasoning in different contexts (optimal scheduling algorithms, response time analysis, estimation of worst-case execution times). These results have built the bases of certifiable probabilistic solutions for real-time systems.

J6 L. Santinelli and L. Cucu-Grosjean, *A Probabilistic Calculus for Probabilistic Real-Time Systems*, ACM Transactions on Embedded Computing Systems, to appear in 2014

J5 D. Maxim and L. Cucu-Grosjean, *"Towards an analysis framework for tasks with probabilistic execution times and probabilistic inter-arrival times"*, Special issue related to the Work in Progress of the 24th Euromicro Inlt Conference on Real-Time Systems (ECRTS 2012), ACM SIGBED Review 9(4):33-36, 2012

J4 L. Santinelli and L. Cucu-Grosjean, *"Towards Probabilistic Real-Time Calculus"*, Special issue related to the 3rd Workshop on Compositional Theory and Technology for Real-Time Embedded Systems(CRTS 2010), ACM SIGBED Review 8(1), March 2011

C11 D. Maxim and L. Cucu-Grosjean, *"Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters"*, IEEE Real-Time Systems Symposium (RTSS), Vancouver, December 2013

C10 R. Davis, L. Santinelli, S. Altmeyer, C. Maiza and L. Cucu-Grosjean, *"Analysis of Probabilistic Cache Related Pre-emption Delays"*, the 25th Euromicro Conference on Real-time Systems (ECRTS), Paris, July 2013

C9 Y. Lu, T. Nolte, I. Bate et L. Cucu-Grosjean, *"A statistical response-time analysis of real-time embedded systems"*, the 33rd IEEE Real-time Systems Symposium (RTSS), San Juan, December 2012

C8 D. Maxim, M. Houston, L. Santinelli, G. Bernat, R. Davis and L. Cucu-Grosjean, *"Re-Sampling for Statistical Timing Analysis of Real-Time Systems"*, the 20th International Conference on Real-Time and Network Systems, ACM Digital Library, Pont à Mousson, Novembre 2012

C7 D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean and R. Davis, *Optimal Priority Assignment for Probabilistic Real-Time Systems*, the 19th International Conference on Real-Time and Network Systems (RTNS), Nantes, September 2011

C6 L. Santinelli, P. Meumeu Yomsi, D. Maxim and L. Cucu-Grosjean, *A Component-Based Framework for Modeling and Analyzing Probabilistic Real-Time Systems*, the 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Toulouse, September 2011

We have studied the probabilistic response time analysis of systems with multiple probabilistic parameters either by using bounds based on real-time calculus (J6, J4), extreme value theory (C9), direct calculation (J5, C11) or in

a context of component-based systems (C6). The probabilistic methods have, generally, high complexity and by upper-bounding the input probability distributions we provide safe and faster results (C8). Worst-case reasoning is also provided to estimate statically the probabilistic worst-case execution time of a task (C10).

- Proposition of *methods for obtaining probability distributions* of the parameters of real-time systems. Obtaining these distributions is, generally, based on statistical methods. We have proposed and validated for the first time a statistical method for the estimation of probabilistic worst-case execution times of a program (task). This method is currently under patent submission and its details are confidential. Its exploitation plan concerns four different embedded industries (avionics, aerospace, automotive and rail) and related methods are also available:

  J8 F. J. Cazorla, E. Quinones, T. Vardanega, L. Cucu-Grosjean, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo and D. Maxim, *"PROARTIS: Probabilistically Analyzable Real-Time System"*, ACM Transactions on Embedded Computing Systems, 12(2s):94, 2013

  J7 L. Yue, I. Bate, T. Nolte and L. Cucu-Grosjean, *"A New Way about using Statistical Analysis of Worst-Case Execution Times"*, Special issue related to the WIP session of the 23rd Euromicro Conference on Real-Time Systems (ECRTS), ACM SIGBED Review, 8(3), September 2011

  C13 F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega and F. Cazorla, *"Measurement-Based Probabilistic Timing Analysis: Lessons from an Integrated-Modular Avionics Case Study"*, the 8th IEEE International Symposium on Industrial Embedded Systems (SIES), Porto, June 2013

  C12 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones and F. Cazorla, *"Measurement-Based Probabilistic Timing Analysis for Multi-path Programs"*, the 24th Euromicro Conference on Real-Time Systems (ECRTS), Pissa, July 2012

  We introduce the first static probabilistic timing analysis for randomized architectures in J8 and the corresponding measurement-based approach in C12. The hypotheses of the mathematical basis of a measurement-based method (as proposed in C12) may be fulfilled also by statistical treatment (J7). A case study of a measurement-based approach is provided for avionics in C13.

Organization of the chapter This chapter has three parts. The first part (Section 2.1) introduces the definitions and assumptions as well as the context of the results we present in Sections 2.3 and 2.2. In Section 2.2 we provide the response

time analysis published in C11. In Section 2.3 we provide the main details of the measurement-based probabilistic timing analysis published in C12. For both results we underline the differences between the independence between tasks and the independence as required by the probabilistic or the statistical method.

2.1 Definitions and assumptions. Context of the probabilistic real-time systems

A random variable $\mathcal{X}$[1] has associated a probability function (PF) $f_{\mathcal{X}}(.)$ with $f_{\mathcal{X}}(x) = P(\mathcal{X} = x)$. The possible values $X^0, X^1, \cdots, X^k$ of $\mathcal{X}$ belong to the interval $[x^{min}, x^{max}]$, where k is the number of possible values of $\mathcal{X}$. We associate the probabilities to the possible values of a random variable $\mathcal{X}$ by using the following notation

$$\mathcal{X} = \begin{pmatrix} X^0 = X^{min} & X^1 & \cdots & X^k = X^{max} \\ f_{\mathcal{X}}(X^{min}) & f_{\mathcal{X}}(X^1) & \cdots & f_{\mathcal{X}}(X^{max}) \end{pmatrix}, \tag{2.1}$$

where $\sum_{j=0}^{k_i} f_{\mathcal{X}}(X^j) = 1$. A random variable may be also specified using its cumulative distribution function (CDF) $F_{\mathcal{X}}(x) = \sum_{z=x^{min}}^{x} f_{\mathcal{X}}(z)$.

Definition 8. *The probabilistic execution time (pET) of the job of a task describes the probability that the execution time of the job is equal to a given value.*

For instance the j^{th} job of a task τ_i may have a pET

$$\mathcal{C}_i^j = \begin{pmatrix} 2 & 3 & 5 & 6 & 105 \\ 0.7 & 0.2 & 0.05 & 0.04 & 0.01 \end{pmatrix} \tag{2.2}$$

If $f_{\mathcal{C}_i^j}(2) = 0.7$, then the execution time of the j^{th} job of τ_i has a probability of 0.7 to be equal to 2.

The definition of the probabilistic worst-case execution (pWCET) of a task is based on the relation $\succeq$ provided in Definition 9.

Definition 9. *[49] Let $\mathcal{X}$ and $\mathcal{Y}$ be two random variables. We say that $\mathcal{X}$ is worse than $\mathcal{Y}$ if $F_{\mathcal{X}}(x) \leq F_{\mathcal{X}}(x)$, $\forall x$, and denote it by $\mathcal{X} \succeq \mathcal{Y}$.*

For example, in Figure 2.1 $F_{\mathcal{X}_1}(x)$ never goes below $F_{\mathcal{X}_2}(x)$, meaning that $\mathcal{X}_2 \succeq \mathcal{X}_1$. Note that $\mathcal{X}_2$ and $\mathcal{X}_3$ are not comparable.

Definition 10. *The probabilistic worst-case execution time (pWCET) $\mathcal{C}_i$ of a task τ_i is an upper bound on the pETs $\mathcal{C}_i^j$ of all jobs of τ_i $\forall j$ and it may be described by the relation $\succeq$ as $\mathcal{C}_i \succeq \mathcal{C}_i^j$, $\forall j$.*

Graphically this means that the CDF of $\mathcal{C}_i$ stays under the CDF of $\mathcal{C}_i^j$, $\forall j$.

Following the same reasoning we define for a task τ_i the probabilistic minimal inter-arrival time (pMIT) denoted by $\mathcal{T}_i$.

[1] We use a calligraphic typeface to denote random variables, e.g., $\mathcal{X}, \mathcal{C}$, etc.

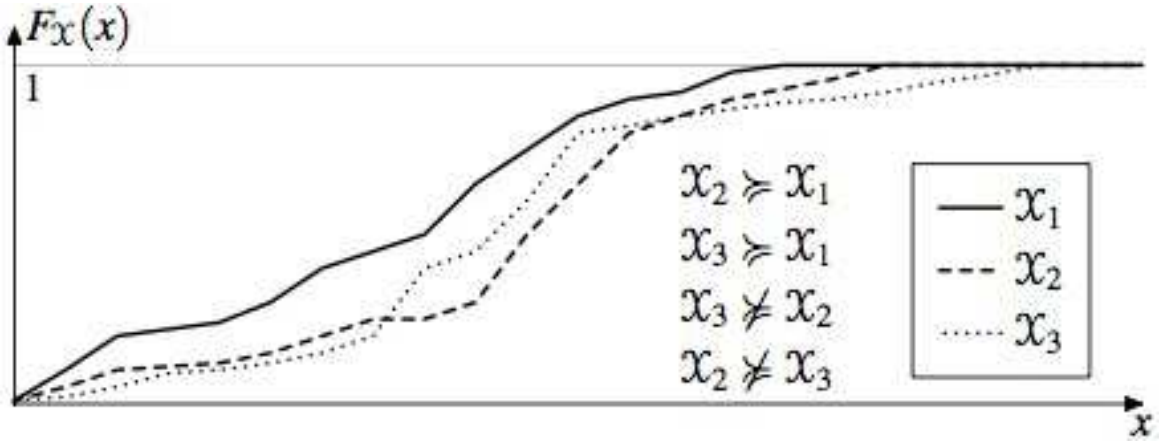


Figure 2.1: Possible relations between the CDFs of various random variables

Definition 12. *The probabilistic inter-arrival time (pIT) of a job of a task describes the probability that the job's arrival time occurs at a given value.*

Definition 13. *The probabilistic minimal inter-arrival time (pMIT) $\mathcal{T}_i$ of a task τ_i is a bound on the minimal inter-arrival times $\mathcal{T}_i^j$, $\forall j$ of all jobs of that task and it may be described by the relation $\succeq$ as $\mathcal{T}_i^j \succeq \mathcal{T}_i$, $\forall j$.*

Graphically this means that the CDF of $\mathcal{T}_i$ stays below the CDF of $\mathcal{T}_i^j$, $\forall j$.

Definition 14. *Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are (probabilistically)* **independent** *if they describe two events such that the outcome of one event does not have any impact on the outcome of the other.*

Definition 15. *The sum $\mathcal{Z}$ of two (probabilistically)* **independent** *random variables $\mathcal{X}$ and $\mathcal{Y}$ is the* **convolution** *$\mathcal{X} \otimes \mathcal{Y}$ where $P\{\mathcal{Z} = z\} = \sum_{k=-\infty}^{k=+\infty} P\{\mathcal{X} = k\}P\{\mathcal{Y} = z - k\}$.*

$$\begin{pmatrix} 3 & 7 \\ 0.1 & 0.9 \end{pmatrix} \otimes \begin{pmatrix} 0 & 4 \\ 0.9 & 0.1 \end{pmatrix} = \begin{pmatrix} 3 & 7 & 11 \\ 0.09 & 0.82 & 0.09 \end{pmatrix}$$

A complementary operator to the convolution is the operator $\ominus$, defined by $\mathcal{X} \ominus \mathcal{Y} = \mathcal{X} \otimes (-\mathcal{Y})$.

Definition 16. *The* **coalescion** *of two partial random variables, denoted by the operator $\oplus$ represents the combination of the two partial random variables into a single (partial) random variable so that values that appear multiple times are kept only once gathering the summed probability mass of the respective values.*

$$\begin{pmatrix} 5 & 8 \\ 0.18 & 0.02 \end{pmatrix} \oplus \begin{pmatrix} 5 & 6 \\ 0.72 & 0.08 \end{pmatrix} = \begin{pmatrix} 5 & 6 & 8 \\ 0.9 & 0.08 & 0.02 \end{pmatrix}$$

Classification of existing results on probabilistic real-time

The first papers in the real-time community with bearing on our work used the terms *stochastic analysis* [25], *probabilistic analysis* [55, 63], *statistical analysis* [3] and *real-time queuing theory* [45] interchangeably. Since the publication of [20], the notion of *stochastic analysis* of real-time systems has been used regularly by the community, regardless of the approach. We use the terms *probabilistic analysis* or *statistical analysis* depending on the approach on which our solution is based. While the former provides the probability or chance of occurrence of future event, the latter searches for a model or some properties when studying some (often large) mass of data of observed past events. The term of *stochastic analysis* is appropriate in our context for models that are time-evolving.

The results on probabilistic real-time systems may be classified following four main types:

Schedulability analysis The seminal paper of Lehoczky [46] proposes the first schedulability analysis of task systems with probabilistic execution times. This result and several improvements [67], [41] consider a specific case of PFs for the pETs. Tia et al. [64] and Gardner [26] propose probabilistic analyses for specific schedulers. Abeni et al. [4] proposes probabilistic analyses for tasks executed in isolation and a recent work consider time-evolving models [59]. The most general analysis for probabilistic systems with (worst-case) execution times of tasks described by random variables is proposed in [20]. The most general analysis for probabilistic systems with (worst-case) execution times and inter-arrival times of tasks described by random variables is proposed in [56]. A time-evolving model of pETs is introduced in [51] and an associated schedulability analysis on multiprocessors is presented.

This class of problems is the most studied among probabilistic methods. The next step to complete the existing results is to provide statistical methods for such problems as it has been done in [50]. The statistical methods have the advantage to be able to study more complex models or architectures. Comparisons between the two methods should close this class of problems.

Re-sampling with respect to probabilistic worst-case parameters Schedulability analyses may have an important complexity directly related to the number of possible values of the random variables describing the parameters. This complexity may be decreased by using re-sampling techniques that ensure the safeness (the new response time PF upper bounds the result obtained without re-sampling) [57, 62]. For instance the response time analysis of [56] becomes interesting for large systems of tasks by using the re-sampling at the level of pWCETs and pMITs. As indicated in [57] there is no optimal re-sampling technique and, thus, any new schedulability analyses should be provided with its own efficient re-sampling technique.

The estimation of the probabilistic parameters Since the seminal paper of Edgar and Burns [21], different papers [17, 34, 40, 42, 66] have proposed solutions for the problem of estimating pWCETs. To our best knowledge one paper proposes

a pET estimation [18]. Estimating pMIT is an open problem and one short paper provides hints for this estimation [53].

Optimal scheduling algorithms To our best knowledge there is one paper presenting optimal fixed-priority uniprocessor scheduling algorithms for tasks with pWCETs [55]. For time-evolving pETs of tasks an optimal fixed-priority algorithm is proposed for several processors [52].

2.1.1 Comparing probabilistic real-time average reasoning and probabilistic real-time worst-case reasoning

We underline the differences between probabilistic real-time average reasoning and probabilistic real-time worst-case reasoning by comparing two existing probabilistic real-time models. For this discussion only the parameters related to the arrival of the tasks are relevant.

Real-time systems with probabilistic MIT

This model has been introduced in papers like [4, 46].

Within this model, for a task τ_i the pMIT $\mathcal{T}_i$ is defined by a distribution as follows:

$$\mathcal{T}_i = \begin{pmatrix} T^0 = T^{min} & T^1 & \cdots & T^k = T^{max} \\ f_{\mathcal{T}_i}(T^{min}) & f_{\mathcal{T}_i}(T^1) & \cdots & f_{\mathcal{T}_i}(T^{max}) \end{pmatrix}$$

For instance, τ_1 has a pMIT $\mathcal{T}_1 = \begin{pmatrix} 5 & 10 \\ 0.3 & 0.7 \end{pmatrix}$ indicating that the MIT of τ_1 is equal to 5 with a probability of 0.3 and to 10 with a probability of 0.7 .

Real-time systems with probabilistic number of arrivals

This model has been used in papers like [8, 44].

Within this model, for a task τ_i^* the number of possible arrivals $\mathcal{N}_i$ within a time interval of length t_Δ is defined by a distribution as follows:

$$\mathcal{N}_i = \begin{pmatrix} N^0 = N^{min} & N^1 & \cdots & N^k = N^{max} \\ f_{\mathcal{N}_i}(N^{min}) & f_{\mathcal{N}_i}(N^1) & \cdots & f_{\mathcal{N}_i}(N^{max}) \end{pmatrix}$$

For instance if $\mathcal{N}_1 = \begin{pmatrix} 1 & 2 & 4 \\ 0.4 & 0.3 & 0.3 \end{pmatrix}$ for $t_\Delta = 12$, then the task τ_1^* has at most 4 arrivals from $t = 0$ to $t = 12$.

The first model provides information to a schedulability analysis, information that the second model does not provide

- **Probabilistic MIT**: The task τ_1 has at most two arrivals before $t = 7$ (with a probability 0.3).

- **Probabilistic number of arrivals**: It is not possible to estimate how many times τ_1^* was released from $t = 0$ to $t = 7$. Different situations are possible like those described in Figure 2.2.

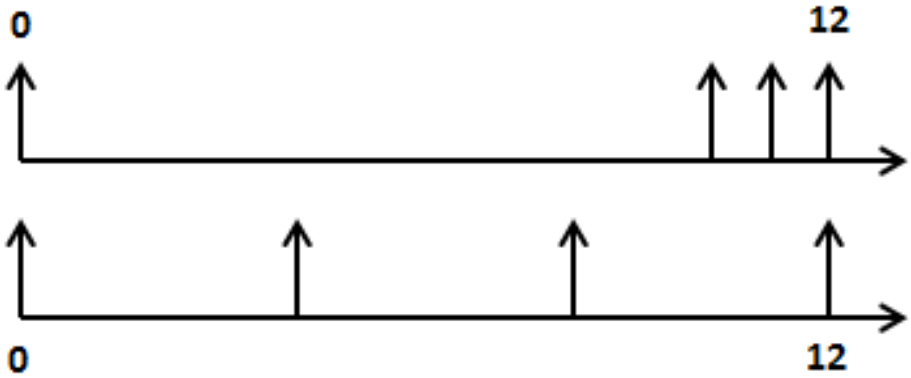


Figure 2.2: The arrivals defined using the number of arrivals may correspond to these situations

The first model can also provide the information that the second model provides to a schedulability analysis

- **Probabilistic MIT**: From $t = 0$ to $t = 12$ there are three scenarios of arrivals for task τ_1:
  - 3 arrivals at $t = 0$, $t = 5$ and $t = 10$ with a probability of 0.21;
  - 2 arrivals at $t = 0$ and $t = 5$ with a probability of 0.09;
  - 2 arrivals at $t = 0$ and $t = 10$ with a probability of 0.7.

  Thus, from $t = 0$ to $t = 12$ the possible number of arrivals of τ_1 is described by $\begin{pmatrix} 2 & 3 \\ 0.79 & 0.21 \end{pmatrix}$.
- **Probabilistic number of arrivals**: from $t = 0$ to $t = 12$ the number of arrivals of τ_1^* is $\mathcal{N}_1 = \begin{pmatrix} 1 & 2 & 4 \\ 0.4 & 0.3 & 0.3 \end{pmatrix}$ and it is provided by the model.

2.2 Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters

The results presented in this section were published in [56].
They were obtained during the PhD thesis of Dorin Maxim [54].

In this section we consider a system of n synchronous tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ to be scheduled on one processor according to a preemptive fixed-priority task-level scheduling policy. Without loss of generality, we consider that τ_i has a higher priority than τ_j for $i < j$. We denote by $hp(i)$ the set of tasks' indexes with higher priority than τ_i. By synchronous tasks we understand that all tasks are released simultaneously the first time at $t = 0$.

Each task τ_i generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \ldots, \infty$. All jobs are assumed to be independent of other jobs of the same task and those of other tasks.

A task τ_i is represented by a tuple $(\mathcal{C}_i, \mathcal{T}_i)$, where $\mathcal{C}_i$ is its probabilistic worst-case execution time and $\mathcal{T}_i$ its probabilistic minimal inter-arrival time. A job of a task must finish its execution before the arrival of the next job of the same task, i.e., the arrival of a new job represents the deadline of the current job[2]. Thus, the deadline of a task may also be represented by a random variable $\mathcal{D}_i$ which has the same distribution as its pMIT, $\mathcal{T}_i$. Alternatively, we can consider the deadline described by a distribution different from the distribution of its pMIT if the system under consideration calls for such model [4,60], or the simpler case when the deadline of a task is given as one value. The latter case is probably the most frequent in practice, nevertheless we prefer to propose an analysis as general as possible and we consider tasks with implicit deadlines, i.e., having the same distribution as the pMIT.

Definition 17 (Job deadline miss probability)**.** *For a job $\tau_{i,j}$ the deadline miss probability $\text{DMP}_{i,j}$ is the probability that the j^{th} job of task τ_i misses its deadline and it is equal to:*

$$\text{DMP}_{i,j} = P(\mathcal{R}_{i,j} > D_i). \tag{2.3}$$

where $\mathcal{R}_{i,j}$ is the response time distribution of the j^{th} job of task τ_i.

We show in Theorem 6 that the case when tasks are simultaneously released yields the greatest response time distribution for each task respectively. Here, greatest is defined with respect to the relation $\succeq$ and it indicates that the response time distribution of the first job upper bounds the response time distribution of any other job of that task. Since we are considering synchronous tasks, calculating the response time distribution of the first job of a task provides the worst-case response time distribution of the task and, implicitly, its worst-case DMP.

In this section we address the problem of computing the response time distributions and, implicitly, Deadline Miss Probabilities of tasks with pMITs and pWCETs. The response time of a job is the elapsed time between its release and its completion. Since we consider jobs with probabilistic parameters, the response time of a job is also described by a random variable. The solution that we describe is exponential in the number of tasks and the size of the random variables representing the task parameters. We describe techniques to decrease the analysis duration and to make it tractable.

[2]In the analysis of generalized multi-frame tasks this is known as the frame separation constraint.

2.2.1 Response time analysis

We present an analysis computing the response time distribution of a given task. Since the system under consideration is a task-level fixed-priority preemptive one, then a given task is not influenced by tasks of lower priority but only by those of higher priority. Thus, we consider without loss of generality, the task of interest to be the lowest priority task τ_n in a set of n tasks.

We remind first the response time analysis for tasks that have only the (WC)ETs described by a random variable [20]. The response time $\mathcal{R}_{i,j}$ of a job $\tau_{i,j}$ that is released at time instant $\lambda_{i,j}$ is computed using the following equation:

$$\mathcal{R}_{i,j} = \mathcal{B}_i(\lambda_{i,j}) \otimes \mathcal{C}_i \otimes \mathcal{I}_i(\lambda_{i,j}), \tag{2.4}$$

where $\mathcal{B}_i(\lambda_{i,j})$ is the accumulated *backlog* of higher priority tasks released before $\lambda_{i,j}$ and still active (not completed yet) at time instant $\lambda_{i,j}$. $\mathcal{I}_i(\lambda_{i,j})$ is the sum of the execution times of higher priority tasks arriving after $\lambda_{i,j}$ and that could preempt the job under analysis, $\tau_{i,j}$. In the case of synchronous tasks the backlog is equal to $\mathcal{B}_n = \bigotimes_{i \in hp(n)} \mathcal{C}_i$.

Equation (2.4) is solved iteratively, integrating new possible preemptions by modifying the tail of the response time distribution $\mathcal{R}_{i,j}$ at each iteration. Iterations stop either when there are no more preemptions to be integrated or when all new values of the tail of the response time distribution are larger than the deadline of the task.

Hypothesis of (probabilistic) independence: Equation (2.4) is based on the operation of convolution $\otimes$ that requires probabilistic independence between $\mathcal{C}_i, \forall i$.

- **Case of pETs.** If the random variables $\mathcal{C}_i$ describe pETs of the task, then the random variables are not independent and Equation (2.4) cannot be applied with the current knowledge of the literature.
- **Case of pWCETs.** If the random variable $\mathcal{C}_i$ describes the pWCET of the task τ_i, then the random variables are independent as they are obtained as upper bounds for pETs of all jobs. Thus Equation (2.4) can be applied.

Intuition of our analysis: In the case when the MIT is described by a random variable, we modify Equation (2.4) to take into account the fact that a preemption can occur at different time instants with different probabilities. This is done by making a copy of $\mathcal{R}_{i,j}$ for each value in the pMIT distribution of the preempting task and scaling each copy with its probability. We then modify the tail of each copy in order to integrate, as in Equation (2.4), the execution requirement of the preempting task. Distributions are then coalesced and the process is repeated until either there are no more preemptions to be integrated or the newly obtained values in the tails of each copy of the response time distribution are larger than the tasks'

deadline. Note that if the MIT of the preempting task is deterministic, then the analysis is the same as Equation (2.4). Furthermore, our analysis can handle any combination of probabilistic and deterministic parameters, and in the case that all parameters are deterministic, the returned result is the same as the one provided by the worst-case response time analysis in [43].

We present now a numerical example of the analysis and we then introduce it formally.

Example 11. *We introduce the response time computation for the lowest priority task of a task system. To ease the presentation, we start with a deterministic task set and move forward a probabilistic version.*

Let τ be a task system $\{\tau_1, \tau_2\}$ scheduled under task-level fixed-priority scheduling policy with τ_1 at higher priority and τ_2 at lower priority. The tasks are described by the following worst-case values: τ_1 by $(C_1 = 2, T_1 = 5)$ and τ_2 by $(C_2 = 4, T_2 = 7)$, with $D_i = T_i, \forall i$. A deterministic analysis of this task system would conclude that it is unschedulable, since the response time of $\tau_{2,1}$ (the first job of τ_2) is greater than its deadline.

First generalization (pMIT): *We add more information to the analysis, namely we use the fact that not all jobs of τ_1 arrive after 5 units of time, but the pMIT of τ_1 is described by $\mathcal{T}_1 = \begin{pmatrix} 5 & 6 \\ 0.2 & 0.8 \end{pmatrix}$ meaning that an instance of τ_1 has a 20% probability of arriving 5 units of time after the previous instance, and a 80% probability of arriving 6 time units after the previous instance. All other parameters of the system have their original worst-case values.*

In this case, $\tau_{2,1}$ has an 80% probability of finishing execution before its deadline, i.e., the cases when $\tau_{1,2}$ arrives at $t = 6$.

Second generalization (pWCET): *We add more information to the deterministic analysis by assuming that τ_2 has a pWCET described by $\mathcal{C}_2 = \begin{pmatrix} 3 & 4 \\ 0.9 & 0.1 \end{pmatrix}$ meaning that only 10% of the jobs generated by τ_2 have an execution requirement of 4 time units and the other 90% require only 3 time units. All other parameters of the system are considered with their worst-case values.*

In this case $\tau_{2,1}$ has a high probability, 90%, of finishing its execution before its deadline, namely the cases when it requires 3 units of execution time.

Combining the two cases: *If we combine both cases presented above by taking into consideration the pMIT of $\mathcal{T}_1$ and the pWCET of τ_2, we note that $\tau_{2,1}$ misses its deadline only when the two worst-case scenarios happen at the same time, i.e., $\tau_{1,2}$ arrives at $t = 5$ and $\tau_{2,1}$ executes for 4 units of time. The probability of the two combined scenarios corresponds to $\tau_{2,1}$ having a probability of $DMP_2 = 0.2 \times 0.1 = 0.02$ of missing its deadline.*

For this example we have found the deadline miss probability of $\tau_{2,1}$ by (manually) exploring all possible combinations of minimal inter-arrival times and worst-case execution times of the two tasks. This is not always possible, considering that a system may have an important number of tasks and each parameter distribution

may have tens, hundreds or even thousands of values, leading to a large number of possible combinations.

The analysis that we introduce below computes the worst-case response time distribution of a task by means of convolution of random variables, ensuring in this way that all scenarios have been taken into account without needing to explicitly investigate all of them.

The analytical response time computation: *The probabilistic representation of the system under analysis is* $\tau = \{\tau_1(\mathcal{C}_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathcal{T}_1 = \begin{pmatrix} 5 & 6 \\ 0.2 & 0.8 \end{pmatrix}), \tau_2(\mathcal{C}_2 = \begin{pmatrix} 3 & 4 \\ 0.9 & 0.1 \end{pmatrix}, \mathcal{T}_2 = \begin{pmatrix} 7 \\ 1 \end{pmatrix})\}$ *and we are interested in finding the response time distribution* $\mathcal{R}_{2,1}$ *of* $\tau_{2,1}$.

The computation starts by initialising the response time distribution $\mathcal{R}_{2,1}$ *with the combined execution time requirements of higher priority tasks, in this case* $\mathcal{C}_1$*, and adding it to the execution requirement of the task under analysis:*

$\mathcal{R}_{2,1} = \mathcal{C}_1 \otimes \mathcal{C}_2 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 3 & 4 \\ 0.9 & 0.1 \end{pmatrix} = \begin{pmatrix} 5 & 6 \\ 0.9 & 0.1 \end{pmatrix}$.

The possible preemption that may occur from $\tau_{1,2}$ *may be either at* $t = 5$ *with probability* 0.2 *or at* $t = 6$ *with probability* 0.8*. For each of these two cases we make a copy of* $\mathcal{R}_{2,1}$ *and proceed in the following way:*

$\mathcal{R}^1_{2,1} = \begin{pmatrix} 5 \\ 0.9 \end{pmatrix} \oplus \left(\begin{pmatrix} 6 \\ 0.1 \end{pmatrix} \otimes \begin{pmatrix} 2 \\ 1 \end{pmatrix}\right) \otimes \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 5 \\ 0.9 \end{pmatrix} \oplus \begin{pmatrix} 8 \\ 0.1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 5 & 8 \\ 0.9 & 0.1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 0.2 \end{pmatrix} = \begin{pmatrix} 5 & 8 \\ 0.18 & 0.02 \end{pmatrix}$ *represents the case when* $\tau_{1,2}$ *arrives at* $t = 5$ *preempting* $\tau_{2,1}$*. In this case,* $\tau_{1,2}$ *can only affect the tail of the distribution, i.e.,* $\tau_{2,1}$ *did not finish execution by* $t = 6$*. Two units of time are added to the tail of the distribution, and the entire resulting distribution is updated with the probability* 0.2 *of* $\tau_{1,2}$ *arriving at* $t = 5$*.*

$\mathcal{R}^2_{2,1} = \begin{pmatrix} 5 & 6 \\ 0.9 & 0.1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 0.8 \end{pmatrix}$ *represents the case that* $\tau_{1,2}$ *arrives at* $t = 6$ *and it does not preempt* $\tau_{2,1}$*. The tail of the distribution is not affected and the entire distribution is updated with the probability* 0.8 *of* $\tau_{1,2}$ *arriving at* $t = 6$*.*

The two copies of $\mathcal{R}_{2,1}$ *are coalesced and the final result is obtained:*

$\mathcal{R}_{2,1} = \mathcal{R}^1_{2,1} \oplus \mathcal{R}^2_{2,1} = \begin{pmatrix} 5 & 8 \\ 0.18 & 0.02 \end{pmatrix} \oplus \begin{pmatrix} 5 & 6 \\ 0.72 & 0.08 \end{pmatrix} = \begin{pmatrix} 5 & 6 & 8 \\ 0.9 & 0.08 & 0.02 \end{pmatrix}$.

The value 8 *of the response time distribution is not possible since* $\tau_{2,1}$ *will not be allowed to continue its execution past* $t = 7$*. Any value strictly greater that the jobs deadline is replaced by "DMP" and their summed probability mass represents the Deadline Miss Probability of the job:* $\mathcal{R}_{2,1} = \begin{pmatrix} 5 & 6 & DMP \\ 0.9 & 0.08 & 0.02 \end{pmatrix}$.

Since the earliest arrival of $\tau_{1,3}$ *is at* $t = 10$*, then this task can not preempt* $\tau_{2,1}$ *and the analysis stops. We have obtained the worst-case response time distribution of* τ_2 *and its Deadline Miss Probability,* 0.02*, which is exactly the same as the one*

obtained earlier by enumerating all possible scenarios.

Third generalization (probabilistic deadline): *In order to show the effect of a probabilistic deadline, we further generalize the task system by considering as given the pMIT distribution and hence the deadline distribution of* τ_2 *is equal to* $\mathcal{T}_2 = \begin{pmatrix} 7 & 8 \\ 0.3 & 0.7 \end{pmatrix}$. *In this case* $\tau_{2,1}$ *will miss its deadline if and only if the following worst-case scenario happens:* $\tau_{1,2}$ *arrives at* $t = 5$ *(probability* 0.2*),* $\tau_{2,1}$ *executes for* 4 *units of time (probability* 0.1*) and* $\tau_{2,2}$ *arrives at* $t = 7$ *(probability* 0.3*). The probability of this scenario happening is* $DMP_2 = 0.2 \times 0.1 \times 0.3 = 0.006$.

This probability can be obtained directly by applying Equation (2.10):

$\mathcal{B}_2 = \mathcal{R}_{2,1} \ominus \mathcal{D}_2 = \begin{pmatrix} 5 & 6 & 8 \\ 0.9 & 0.08 & 0.02 \end{pmatrix} \ominus \begin{pmatrix} 7 & 8 \\ 0.3 & 0.7 \end{pmatrix}$ *and*

$$\mathcal{B}_2 = \begin{pmatrix} -3 & -2 & -1 & 0 & 1 \\ 0.63 & 0.83 & 0.24 & 0.014 & 0.006 \end{pmatrix}.$$

The values that are less or equal to zero are discarded, since they represent the cases when the job finishes execution before or at the deadline. The strictly positive values are kept and their added probabilities represent the tasks' DMP. In this case the probability of the value 1 *is* 0.006 *as it is found by the descriptive method above.*

We introduce now a formal description of our analysis.

Description of our analysis: The arrival time of the j^{th} job of a task τ_n is computed for $j \geq 1$ as follows

$$\underbrace{\mathcal{T}_{n,j} = \mathcal{T}_n \otimes ... \otimes \mathcal{T}_n}_{j-1 \ times} \tag{2.5}$$

and for $j = 0$ we have $\mathcal{T}_{n,0} = 0$.

The worst-case response time of task τ_n is initialized as:

$$\mathcal{R}_n^0 = \mathcal{B}_n \otimes \mathcal{C}_n \tag{2.6}$$

where the backlog at the arrival of τ_n is equal to

$$\mathcal{B}_n = \bigotimes_{i \in hp(n)} \mathcal{C}_i \tag{2.7}$$

After adding the execution time of the task under analysis to the backlog accumulated at its arrival, its response time is updated iteratively with the possible preemptions as follows:

$$\mathcal{R}_n^i = \bigoplus_{j=1}^{k} \mathcal{R}_n^{i,j} \tag{2.8}$$

where i is the current iteration, k is the number of values in the random variable representing the pMIT distribution of the preempting task, j is the current

value taken into consideration from the pMIT distribution of the preempting task. $\mathcal{R}_n^{i,j}$ is the j^{th} copy of the response time distribution and it integrates the possible preemption in the following way:

$$\mathcal{R}_n^{i,j} = (\mathcal{R}_n^{i-1,head} \oplus (\mathcal{R}_n^{i-1,tail} \otimes \mathcal{C}_m^{pr})) \otimes \mathcal{P}_{pr} \tag{2.9}$$

where:

- n is the index of the task under analysis;
- i is the current step of the iteration;
- j represents the index of the current value taken into consideration from the pMIT distribution of the preempting task;
- $\mathcal{R}_n^{i-1,head}$ is the part of the distribution that is not affected by the current preemption under consideration;
- $\mathcal{R}_n^{i-1,tail}$ is the part of the distribution that may be affected by the current preemption under consideration;
- m is the index of the higher priority task that is currently taken into account as a preempting task;
- $\mathcal{C}_m^{pr}$ is the execution time distribution of the currently preempting task;
- $\mathcal{P}_{pr}$ is a fake random variable used to scale the j^{th} copy of the response time with the probability of the current value i from the pMIT distribution of the preempting task. This variable has one unique value equal to 0 and its associated probability is equal to the i^{th} probability in the pMIT distribution of the preempting job.

For each value $v_{m,i}^j$ in $\mathcal{T}_{(m,j)}$ for which there exists at least one value $v_{n,i}$ in $\mathcal{R}_n^{i-1}$ so that $v_n^i > v_{m,i}^j$, the distribution $\mathcal{R}_n^{i-1}$ is split in two parts:

- $\mathcal{R}_n^{i-1,head}$ which contains all values $v_{n,i}^-$ of $\mathcal{R}_n^{i-1}$ that are less or equal than $v_{m,i}^j$, i.e., $v_{n,i}^- \leq v_{m,i}^j$, and
- $\mathcal{R}_n^{i-1,tail}$ which contains all values $v_{n,i}^+$ of $\mathcal{R}_n^{i-1}$ that are greater than $v_{m,i}^j$, i.e., $v_{n,i}^+ > v_{m,i}^j$.

The iterations end when there are no more arrival values $v_{m,i}^j$ of any job i of any higher priority task τ_m that is smaller than any value of the response time distribution at the current step. A stopping condition may be explicitly placed in order to stop the analysis after a desired response time accuracy has been reached. For example, the analysis can be terminated once an accuracy of 10^{-9} has been

reached for the response time. In our case, the analysis stops when new arrivals of the preempting tasks are beyond the deadline of the task under analysis, i.e., the type of analysis required for systems where jobs are aborted once they reach their deadline.

Once the response time distribution of the jobs can be computed, then the Deadline Miss Probability is obtained by comparing the response time distribution with that of the deadline, as follows:

$$\mathcal{B}_i = \mathcal{R}_i \ominus \mathcal{D}_i = \mathcal{R}_i \oplus (-\mathcal{D}_i), \tag{2.10}$$

where the $\ominus$ operator indicates that the values of the distribution are negated. We use the notation $\mathcal{B}_i$ even though the resulting distribution is not a backlog distribution in the strict sense for the model we consider, but it is still the formula for computing backlog for systems where jobs are allowed to execute past their deadline.

The DMP of the job under analysis is given by the probability mass corresponding to the values strictly greater than 0, i.e., the job would need more units of time to finish its execution. The probability mass corresponding to the values less or equal to 0 gives the probability that the job finishes execution before its deadline and the next release.

2.2.2 Critical instant of a task with multiple probabilistic parameters

Lemma 5. *We consider a task system of n tasks with τ_i described by deterministic C_i and probabilistic $\mathcal{T}_i, \forall i \in \{1, 2, \cdots, n\}$. The set is ordered according to the priorities of the tasks and the system is scheduled preemptively on a single processor. The response time distribution $\mathcal{R}_{i,1}$ of the first job of task τ_i is greater than the response time distribution $\mathcal{R}_{i,j}$ of any j^{th} job of task τ_i, $\forall i \in \{1, 2, \cdots, n\}$.*

Proof. The response time distribution $\mathcal{R}_{i,j}$ of a job within a probabilistic system may be obtained by composing response time values $R^k_{i,j}$ of jobs within all corresponding deterministic systems obtained by considering all values of the minimal inter-arrival times and the probability associated with the respective scenario k and we have $\begin{pmatrix} R^k_{i,j} \\ p_{scenario_k} \end{pmatrix}$. For each of these deterministic systems we know from [48] that the critical instant of a task occurs whenever the task is released simultaneously with its higher priority tasks. Thus we have that $R^k_{i,1} \geq R^k_{i,j}, \forall k, j > 1$ and we obtain $\mathcal{R}_{i,1} \succeq \mathcal{R}_{i,j}$ as the associated probabilities of $R^k_{i,1}$ and$R^k_{i,j}, \forall k$ are the same. □

Theorem 6. *We consider a task system of n tasks with τ_i described by probabilistic $\mathcal{C}_i$ and $\mathcal{T}_i, \forall i \in \{1, 2, \cdots, n\}$. The set is ordered according to the priorities of the tasks and the system is scheduled preemptively on a single processor. The response time distribution $\mathcal{R}_{i,1}$ of the first job of task τ_i is greater than the response time distribution $\mathcal{R}_{i,j}$ of any j^{th} job of task τ_i, $\forall i \in \{1, 2, \cdots, n\}$.*

Proof. The response time distribution $\mathcal{R}_{i,j}$ of a job within a probabilistic system is obtained by convolving response time distributions $\mathcal{R}^l_{i,j}$ of jobs within all corresponding probabilistic systems obtained by considering tasks described by $C_i, \mathcal{T}_i, \forall i$. Then within each scenario l we have from Lemma 5 that $\mathcal{R}^l_{i,1} \succeq \mathcal{R}^l_{i,j}$. We have then $\mathcal{R}_{i,1} = \otimes_{l=1}^{nb_of scenarios} \mathcal{R}^l_{i,1} \succeq \otimes_{l=1}^{nb_of scenarios} \mathcal{R}^l_{i,j} = \mathcal{R}_{i,j}$. $\square$

2.2.3 Implementation and evaluation of the method

We implemented our response time analysis in MATLAB. The pseudo-code for the associated steps is presented in Algorithms 1 and 2 and the complete scripts are available[3].

Before we proceed with the description of the simulations performed we recall here the concept of re-sampling[4].

Definition 18. *[57] [**Re-sampling**] Let $\mathcal{X}_i$ be a distribution with n values describing a parameter of a task τ_i. The process of **re-sampling to k values** or **k-re-sampling** consists of reducing the initial distribution $\mathcal{X}_i$ from n values to a distribution $\mathcal{X}^*_i$ with k values.*

The re-sampling is safe with respect to the response time analysis as the response time $\mathcal{R}_i$ of any task τ_i of the initial system is greater than the response time $\mathcal{R}^*_i$ of the considered task within the re-sampled task system.

The re-sampling of a real-time distribution is performed in the following two sequential steps: *1*) selection of the k samples to be kept in the reduced distribution and *2*) redistribution of the probabilities from the values that are not kept.

Re-sampling for pWCET differs from re-sampling for pMIT in the second step, namely, as larger values of pWCET produce greater probabilistic response times as well as smaller values of pMIT produce greater probabilistic response times.

2.2.4 Evaluation with respect to the complexity

Probabilistic operations like convolutions of random variables may have a high complexity and we evaluated their impact on the complexity of our analysis.

In Figure 2.3, a 3D plot of the analysis duration is presented. On the z-axis the analysis duration is given in seconds, on the x-axis is the variation of the number of values per random variable, from 2 to 16 values, and on the y-axis is the number of tasks per task system, also from 2 to 16 tasks. Every point on the surface corresponds to the average analysis duration of 100 task sets. The worst-case utilization of each considered task is between 1.5 and 2 and the expected utilization is

[3]The scripts are available at `http://www.loria.fr/~maxim/probabilistic_anaysis_tool_-_matlab_scripts/probabilisticWorstCaseResponseTime.m`

[4]Note that in statistics, re-sampling has a different meaning from that used in real-time systems. For an example of re-sampling in real-time systems the reader may refer to [57]. In statistics, the technique presented here is designed by terms like Bootstrap of Jackkife, the reader may refer to [22].

Algorithm 1 Worst-case response time distribution computation

Input: Γ a task set and $target$ the index of the task we analyze
Output: $\mathcal{R}_{target}$ the worst-case response time distribution of τ_{target}

```
R_target = C_target; //initialize the response time with the execution time of the task under analysis
for (i = 1; i < target; i++) do
    R_target = R_target ⊗ C_i; //add the execution times of all higher priority tasks
end for
for (i = 1; i < target; i++) do
    A_i = T_i; //initialize the arrivals of each higher priority task with their inter-arrival times distribution
end for
for (i = 1; i < max(T_target); i++) do
    for (j = 1; j < target; j++) do
        if max(R_target) > min(A_j) and min(A_j) = i then
            R_target = doPreemption(R_target, A_j, C_j); //update the response time with the current possible preemption
            A_j = A_j ⊗ T_j; //the next arrival of τ_j
        end if
    end for
end for
R_target = sort(R_target)
```

Output: $\mathcal{R}_{target}$

between 0.5 and 1. The pWCETs are decreasing distributions while the pMITs are increasing distributions.

We note that the analysis duration of a task set with 16 tasks, each of its random variables having 16 values, takes in average 140 seconds, i.e., the highest point on the z-axis.

The analysis duration increases both with respect to the number of tasks per task system and with respect to the number of values per random variable, indicating the exponential complexity of the analysis. Nevertheless, solutions exist to make such analysis affordable even for large task systems with parameters described by large random variables.

A solution to decreasing the probabilistic analysis duration is re-sampling, which reduces the analysis duration while introducing minimal pessimism [57].

In Figure 2.4, the diagonal of the surface from Figure 2.3 is represented by a solid line, having an exponential behaviour. The same analysis is performed with re-sampling of the pWCET to 50 values and of the pMIT to 5 values both done after each iteration. The improvement is shown in the same figure, represented by the dotted line; this time the average analysis duration over 100 task sets each having 16 task with 16 values per random variable is only 1.29 seconds, compared to 140 seconds when no re-sampling was performed. We note the important gain in speed when the analysis is performed with re-sampling, even for systems that

Algorithm 2 doPreemption function

Input: $\mathcal{R}$ the current response time,
$\mathcal{A}$ the arrival distribution of the preempting job and
$\mathcal{C}$ the execution time distribution of the preempting job
Output: $\mathcal{R}$ the response time distribution updated with the current preemption
$\mathcal{R}_{intermediary} = empty$;
$\mathcal{A}_{fake} = empty$;
for $(i = 1; i < length(\mathcal{A}); i++)$ **do**
//constructing the fake random variable giving the probability of the preemption occurring
$\mathcal{A}_{fake}.value = 0$; //the value of the fake random variable
$\mathcal{A}_{fake}.probability = \mathcal{A}$(i).$probability$; //the probability of the fake random variable
Split $\mathcal{R}$ into *head* and *tail* according to the preemption value;
if *tail* != *empty* **then**
$tail = tail \otimes \mathcal{C}$;
end if
$\mathcal{R}_{intermediary} = head \oplus tail$;
$\mathcal{R}_{intermediary} = \mathcal{R}_{intermediary} \otimes \mathcal{A}_{fake}$;
$\mathcal{R} = \mathcal{R} + \mathcal{R}_{intermediary}$;
$\mathcal{R} = sort(\mathcal{R})$
end for
$\mathcal{R} = \text{sort}(\mathcal{R}_{intermediary})$
Output: $\mathcal{R}$

have 32 tasks and each random variable has 32 values it takes 11 seconds to perform the analysis, indicating that it is affordable even for considerable larger systems. We show in the next experiment that distributions with 5 values for pMIT bring significant increased precision with respect to the worst-case response time analysis.

2.2.5 Evaluation with respect to existing analysis

The second set of experiments that we performed show the precision that is gained by having tasks' parameters given as random variables. In order to do so we randomly generated probabilistic task systems to which we applied our analysis with different levels of re-sampling applied either at pWCET level or at pMIT level.

Precision gained by having a more detailed pMIT distribution

To show the increased precision brought by a more detailed pMIT distribution, we repeated three times the analysis on the generated task system, each time varying the re-sampling level of the pMIT, using 10 values, 5 values and 1 value, respectively. A pMIT distribution with only one value is in fact a worst-case MIT. The pWCET distribution was not re-sampled.

Figure 2.5 shows the task Deadline Miss Probability averaged over 100 task

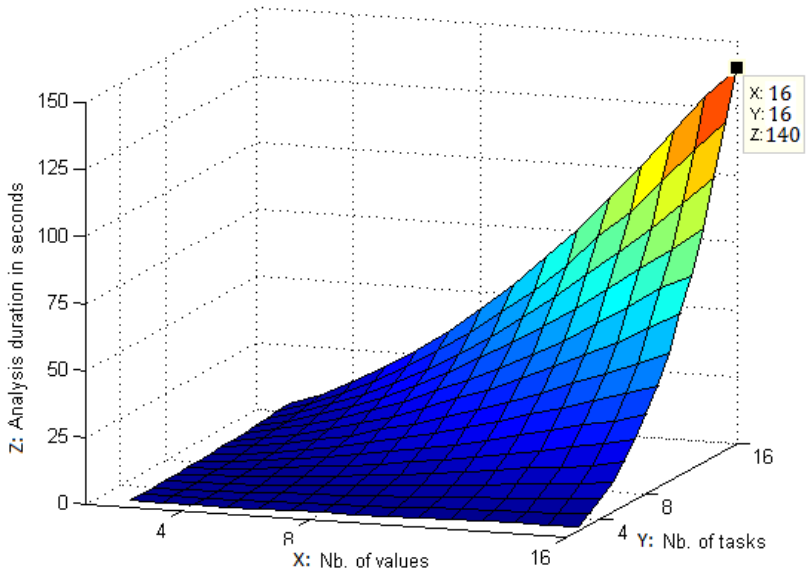


Figure 2.3: Analysis duration of random task system

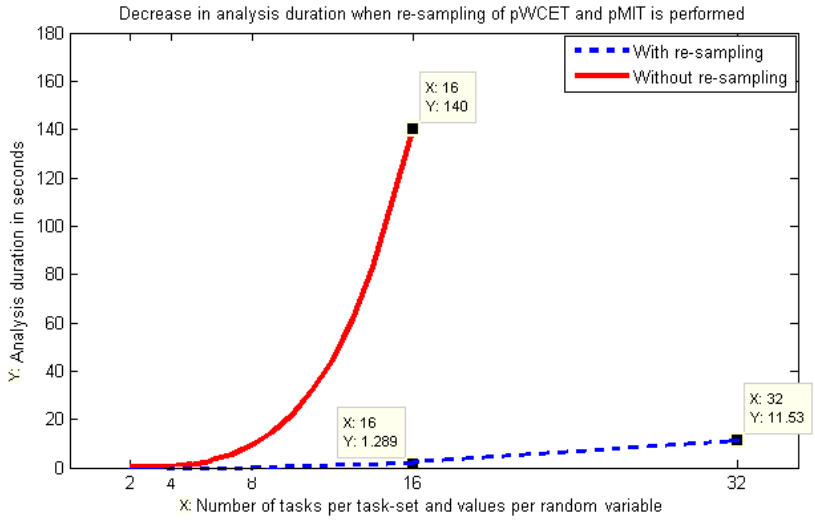


Figure 2.4: Analysis duration of random task systems , increasing both the number of tasks per task system and the number of values per random variable in the same time

systems of 10 tasks. We note that the deterministic reasoning that considers one value for the MIT (and no re-sampling for pWCET) provides a DMP equal to 0.0167 (left bar in Figure 2.5) - this is the case of the existing analysis presented in [20]. By considering a probabilistic reasoning with 10 values for the pMIT we decrease by a factor of 3 the DMP. Then further increasing the number of values within the pMIT only marginally decreases the DMP and this is shown by comparing the values of DMP for 1, 5 and 10.

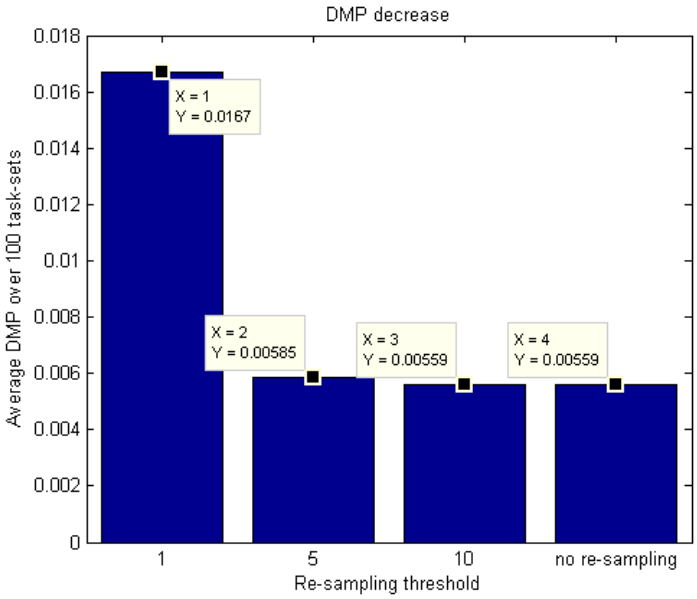


Figure 2.5: The difference in DMP when the tasks' pMIT distribution has 1, 5 and respectively 10 values. Here 1 value indicates that only the worst-case value of the pMIT distribution is considered.

Nevertheless, having a 10-value pMIT does not bring much increased precision over a 5-value pMIT, as can also be seen in Figure 2.5 where their respective DMPs are almost equal. We note that it is not necessary to have a large pMIT distribution to have a precise analysis, depending on the system under analysis just 5 values can be sufficient.

This increased precision comes at a cost, namely an increase in the analysis duration: for larger distributions the analysis needs more time to perform. In Figure 2.6 we show the analysis duration of the three cases described above. The duration of the 10 values analysis is close to that of no-re-sampling, where the duration of 5 values analysis is decreased. In this case the 5 values analysis seems to be a comfortable compromise between the duration and the gained DMP.

Precision gained by having a more detailed pWCET distribution

We performed a set of experiments to show the difference in precision when the tasks' pWCET distribution has 1000 values, 100 values, 10 values or only 1 value. A pWCET distribution with only one value is a deterministic WCET. The analysis was performed on the randomly generated task systems on which there were applied, in turn, different levels of re-sampling to the pWCET distribution. The pMIT distribution was not re-sampled.

In Figure 2.7 the difference in DMP between the four cases is depicted. Note that having 1 or 10 values in the pWCET distribution returns a task DMP equal to 1 which means that the system would be deemed unfeasible. This is not necessarily

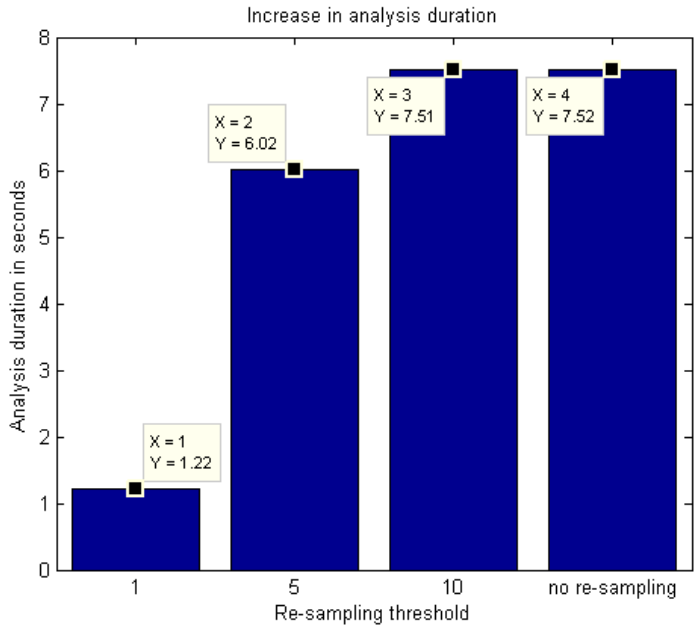


Figure 2.6: The difference in seconds of the analysis duration when the tasks' pMIT distribution has 10, 5 and respectively 1 value, i.e., only the worst-case value of the pMIT distribution.

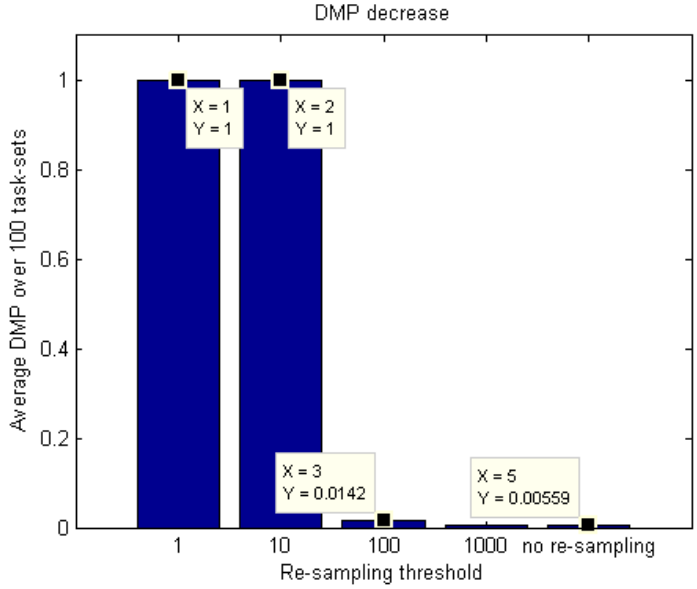


Figure 2.7: The difference in DMP when the tasks' pWCET distribution has 1000, 100, 10 and respectively 1 value, i.e., only the worst-case value of the pMIT distribution.

true, as can be seen from the bar representing the case when the pWCET distribution has 100 values. In this case the average DMP values of the analysed tasks does not surpass 0.02 which means that the systems could be feasible if they can

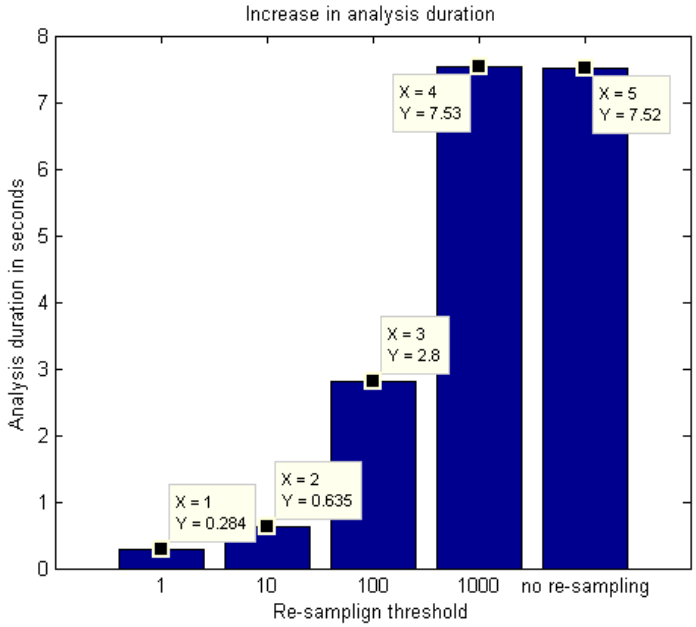


Figure 2.8: The difference in seconds of the analysis duration when the tasks' pWCET distribution has 1000, 100, 10 and respectively 1 value, i.e., only the worst-case value of the pMIT distribution.

afford a 0.02 Deadline Miss Probability for their lowest priority tasks. In conclusion we decreased the DMP fifty times by analysing the system with 100 values for the pWCET.

As in the case of re-sampling at pMIT level, the re-sampling at pWCET level also comes with an increase in the analysis duration. Figure 2.8 depicts the analysis duration of the four cases described above, with the 1000 values for the pWCET having an exponential behavior but also being the distribution that has the most important precision at DMP level. For pWCET re-sampling 100 is a compromise level that allows to obtain affordable duration and important increased DMP.

In [57] a study is performed on different re-sampling strategies and novel re-sampling strategies are proposed that introduce very little pessimism. Also, by combining pWCET re-sampling and pMIT re-sampling, the analysis duration can be decreased considerably while retaining a high level of accuracy, regardless of the system under analysis.

2.3 Measurement-Based Probabilistic Timing Analysis

The results presented in this section were published in [17].
They were obtained within the FP7 STREP PROARTIS project.

In this section we present a statistical method for estimating the probabilistic worst-case execution time of a program (task) on a given platform. This statistical

method is called Measurement-Based Probabilistic Timing Analysis and it is based on a sound application of Extreme Value Theory.

2.3.1 Extreme Value Theory for Timing Analysis

One may use mathematical methods to predict, out of a small enough number of samples, pWCET bounds for exceedance probabilities smaller than 10^{-n}, where n is a required level of confidence. We explore a mathematical method based on Extreme Value Theory (EVT). This theory estimates the probability of occurrence of extreme values, whether high or low, which are known to be rare events [21]. More precisely, EVT predicts the distribution function for the maximal (or minimal) values of a set of n observations, which are modelled with random variables. The EVT theory is analogous to Central Limit Theory [24] but instead of estimating the average, EVT estimates the extremes [28]. As in our work we are interested in the high values that bound the pWCET, we consider the EVT prediction for maximal values of a set of observations.

Theorem 7. *[28] Let $\{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$ be a sequence of i.i.d. random variables and let $\mathcal{M}_n = max\{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n\}$. If F is a non degenerate distribution function and there exists a sequence of pairs of real numbers (a_n, b_n) such that $a_n \geq 0$ and $lim_{n\to\infty} P(\frac{\mathcal{M}_n - b_n}{a_n} \leq x) = F(x)$, then F belongs to either the Gumbel, the Frechet or the Weibull family.*

The main result of EVT is provided in Theorem 7, where F denotes the common distribution function of n random variables. In our approach, random variables are used to model the execution time of programs (tasks). In order to apply Theorem 7 two main hypotheses must be verified: that the n random variables are independent and identically distributed and the existence of the sequence of real numbers (a_n, b_n).

The Gumbel, Frechet and Weibull are particular cases of the Generalised Extreme Value (GEV) distribution which has the following Cumulative Distribution Function (CDF):

$$F_\xi(x) = \begin{cases} e^{-(1+\xi\frac{x-\mu}{\sigma})^{\frac{1}{\xi}}} & \xi \neq 0 \\ e^{-e^{-\frac{x-\mu}{\sigma}}} & \xi = 0 \end{cases}$$

GEV is defined by three parameters: shape (ξ), scale (σ) and location (μ). By determining these three parameters, we prove the existence of the sequence of real numbers (a_n, b_n). If the shape parameter $\xi = 0$ then F_ξ is a Gumbel distribution, which is the distribution we use in this analysis.

When EVT is applied, we must determine its three parameters [27] and hence the appropriate CDF (Gumbel, Frechet or Weibull) is selected. To that end, a goodness-of-fit test is needed. However, not all tests are appropriate for fitting extreme values. For instance, in [40] the authors use the χ^2 test which is known to

perform incorrectly for extreme values in any classical test [27]. One test that is proven correct when fitting the Gumbel CDF is the exponential tail (ET) test [27].

EVT has previously been applied to the WCET estimation problem in [21, 40]. The main limitations of these two papers, which are shown in [34], are the following: the assumption of independent and identically distributed random variables, and fitting a continuous distribution to discrete values. We correct the application of EVT to WCET estimation by proposing solutions to these limitations. Moreover, we also provide an alternative to the χ^2 test in [40] for fitting the Gumbel CDF.

We detail now the hypotheses needed for a correct application of EVT.

Independence and identical distribution: The main hypothesis for Theorem 7 is that the sequence of random variables is independent and identically distributed. We provide here the definition of this notion. In following sections we show how to achieve this property when applying EVT to obtain pWCET estimates.

Definition 19 (Identically distributed Random Variables)**.** *A sequence of random variables is independent and identically distributed (i.i.d.) if all random variables belong to the same probability distribution and they are mutually independent.*

A sequence of (fair) die rolls where each roll is a random variable is i.i.d., since the random variables describe the same event and the outcomes are obtained from independent events (as the outcome of one roll is independent of the outcome of another roll).

Statistical independence between random variables Theorem 7 (as many statistical results) requires that the n random variables $\mathcal{X}_1$, $\mathcal{X}_2, \ldots, \mathcal{X}_n$ are (probabilistically) independent. Here the variables $\mathcal{X}_1$, $\mathcal{X}_2, \ldots, \mathcal{X}_n$ describe observations of the execution of task τ_i. The independence hypothesis implies that $\mathcal{X}_i$ is obtained from observed executions that are independent from those contained by $\mathcal{X}_j$, $\forall i \neq j$. Nevertheless in reality the user of this theory runs the program (task) under study several times and obtains a set of data. In this case in order to fulfill the hypothesis of independence required by Theorem 7, the user checks the (statistical) independence by applying different independence tests on those data [24]. For instance the lag test results may indicate graphically if the data are independent (see Figure 2.9, right graph) or not (see Figure 2.9, left graph).

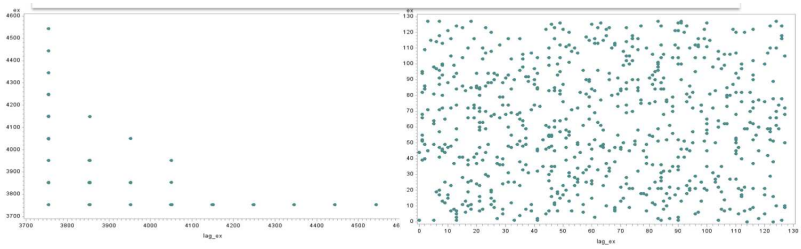


Figure 2.9: Graphical results provided by lag test on two sets of observed execution times

In conclusion EVT requires that the observed executions of a task are made

independently and this does not imply any extra requirement on the independence of different programs (or tasks), but on the process of producing and collecting the data.

Selection of an EVT distribution: In order to apply Theorem 7, parameters describing a valid EVT distribution (Gumbel, Frechet or Weibull) must be chosen. This requires an appropriate statistical test. Previous applications of EVT to WCET estimation [21, 40] indicate that the Gumbel distribution fits well the problem of WCET estimation. We validate this hypothesis by checking whether a given CDF fits the Gumbel CDF, for which we use the exponential tail (ET) test [27]. The ET test assumes that the data set is modelled by n i.i.d. random variables $\mathcal{X}_1, \cdots, \mathcal{X}_n$ described by the CDF F. The ET test compares this to the Gumbel CDF: $\mathcal{H}_0 : F = F_\xi$. If the test rejects the hypothesis $\mathcal{H}_0$, then the hypothesis $\mathcal{H}_1 : F \neq F_\xi$ must be true. The details for applying ET to our data are provided in Section 2.3.4.

2.3.2 Steps in the application of EVT

There are two main steps in the application of EVT.

Grouping: The objective of this step is to collect, from the original distribution, the values which fit the tail, and hence can be modelled with the Gumbel distribution. The values used as input for EVT are grouped into *blocks* of equal length m by applying the *block maxima* method. The maximum value observed in each block constructs a new sample, the block maximum series.

The size of the blocks determines the portion of the original distribution that is considered the tail. Larger is the block size, better data fit the tail; however, increasing the block size results in fewer blocks and thus fewer values in the final sample. In the extreme case, if a single block is used, we will have only one block maximum value corresponding to the maximum of the original distribution. Conversely, if we use as many blocks as elements in the original distribution, the distribution of the block maximum values will gather the entirety of the original distribution, rather than capturing the essence of the tail.

The theoretical basis of the block maxima method is built on the following theorem which says that if some data fit the Gumbel CDF, then the maxima of blocks of those data fit the same Gumbel CDF.

Theorem 8. *[36] [Grouping] Let $\mathcal{X}$ be a random variable. If the n variables $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ obtained from $\mathcal{X}$ fit a GEV, then the maxima of any blocks obtained from $\mathcal{X}$ fit the same GEV.*

Distribution fitting: If the ET test shows that our observations follow a Gumbel distribution, we next estimate the two remaining parameters, μ and σ using the block maxima series. Their practical estimation is obtained by applying linear regression to the QQ-plot of our data [23]. A QQ-plot (quantile plot) is a plot of the empirical quantile values of observed data against the quantiles of the standard form of a target distribution. Given that the block maxima values follow

a Gumbel distribution, then the points on the QQ-plot form a straight line [23]. The slope and the intercept of the best-fit line through these points can be used as estimators for the Gumbel μ and σ parameters, respectively.

2.3.3 Continuous vs. Discrete Functions

In [34] it is shown that using the Gumbel distribution function, or any continuous function, to approximate a discrete function of the execution time values produced by a program, could yield results that do not bound the execution time from above at all points. In [34] there are proposed different solutions to this problem by either the continuous function fitting the upper bound of the exceedance distribution function and thus overestimating the discrete function, or by adding an offset to the distribution which accounts for this effect. For instance for the single path we consider here, the maximum offset required is likely to be a small number of cycles.

In our paper we introduce an overestimation of the discrete function when applying EVT (Section 2.3.2) through the block maxima method before the distribution fitting is applied. This produces a shift of the distribution, which makes the worst-case estimate pessimistic when compared to the 'real' distribution of execution times in our architecture. In order to obtain high confidence in such a distribution, we require that the number of observations is sufficient to produce an accurate description of the program's execution time. We call this the **minimum number of observations**, and we define it as follows:

Definition 20. *For a given sequential program P and an architecture A, n is the minimum number of observations if there does not exist $m \in \mathbb{N}$ where $m > n$ such that the Gumbel CDF obtained for n observations is an upper bound for the WCET of P and also exceeds the CDF obtained for m observations.*

We obtain the *minimum number of observations* by comparing the variation in the EVT tail projection as we consider an increasing number of runs. We follow an iterative process:

1. We start by running $N_{current} + N_{delta}$ times the program under study on the target platform.
2. We then make two tail projections with EVT, one using $N_{current}$ runs and the other using $N_{current} + N_{delta}$.
3. If the difference between the two EVT distributions is below a given *difference threshold* we stop the process. If it is over the threshold, we consider all previous runs as $N_{current}$, or in other words we make $N_{current} = N_{current} + N_{delta}$, and we make N_{delta} more runs, consider the part of the sample and repeat the process from step 2.

As we keep adding more runs to $N_{current}$ the proportional effect that N_{delta} new runs has on the EVT reduces, so a convergence of the algorithm is ensured. On

the other hand, there is no guarantee that there is a strict convergence so we can have some local minima. In order to take into account this non strict convergence, we change the algorithm so that instead of stopping the process as soon as the difference between the two EVT distributions is below a given difference threshold, we stop the process when this is the case in several consecutive iterations. For the benchmarks used in this paper, considering 5 consecutive iterations below the threshold is sufficient to deal with the hysteresis introduced by local minima.

The metric we use to compare the two EVT distributions (functions) is called the continuous rank probability score defined as $CRPS = \sum_{i=0}^{+\infty}[f_{\mathcal{X}}(i) - f_{\mathcal{Y}}(i)]^2$ [7]. This test is defined for any two distribution functions $f_{\mathcal{X}}$ and $f_{\mathcal{Y}}$ as long as they operate on the same value domain. In this paper we set as difference the threshold 10^{-1}. This threshold has to be interpreted as the significance level α in hypothesis tests. The lower its value the larger is the confidence on the result. It is up to the user of our technique to set a proper value to this difference threshold. However, note that the lower the threshold the higher the required number of observations.

2.3.4 Experimental Evaluation

In this section we first present the experimental environment we used to evaluate our analysis technique. Next, we introduce SPTA as a reference probabilistic timing analysis method for single-path programs. Finally, we present the results that we obtained, in terms of pWCET estimates, for both single-path and multi-path programs.

Experimental Set-up

We run all experiments on a modified version of SoCLib [1], an open platform for virtual prototyping of multi-processors system on chip (MP-SoC). We modelled a pipelined processor featuring data and instruction caches. We used 4Kilobyte, 1024-way, 4-byte line, fully-associative data and instruction caches deploying a random-replacement policy. The latency of each instruction is fixed, except for the first stage (fetch) in which the instruction cache is accessed. The latency of the fetch stage depends on whether the access hits or misses in the instruction cache: a hit has 1-cycle latency and a miss has 100-cycle latency. After the decode stage, memory operations access the data cache so they can last 1 or 100 cycles depending on whether they miss or not. Overall, the possible latencies of a non-memory operation are (N+1, N+100) depending whether it hits or not in the instruction cache, where N if the number of cycles it takes executing the instruction (e.g. integer additions take 1 cycle). Memory operations have 3 possible latencies: 2 cycles (1+1) when it hits both instruction and data caches; 200 (100+100) when it misses in both caches; 101 (1+100) when it hits only one of both caches.

Benchmarks. We used eight benchmarks of the EEMBC Autobench suite [61] as reference for the analysis of single-path programs. EEMBC Autobench is a well-known benchmark suite that reflects the current real-world demand of some

embedded systems. We used: a2time (AT), aifirf (AI), cacheb (CA), canrdr (CN), puwmod (PU), rspeed (RS), tblook (TB) and ttsprk (TT).

For multi-path program analysis we used several Mälardarlen benchmarks [37], which are commonly used in the community to evaluate and compare different types of WCET analysis tools and methods. In particular we used: bs (BS), cnt (CNT), compress (COM), crc (CRC), insertsort (INS), qsort (QSO) and select (SEL). For each of these programs we developed several input sets that exercise different paths.

We also designed a multi-path synthetic benchmark to better understand how the multi-path program time analysis works. In the synthetic benchmark we know (1) the different paths of the program, (2) the different input set that exercise each path and (3) the longest path. Thus we may compare the pWCET estimate we obtain with our single-path time analysis when applied to the worst-case path against the pWCET estimate we obtain with our multiple-path time analysis when applied to a set of paths which includes the worst-case path.

SPTA

Static probabilistic time analysis, which was first introduced in [10], is used as a term of reference for our approach since SPTA provides a tight pWCET estimate to the actual distribution of execution times of the program. However, it does so at the cost of being extremely more onerous to execute since it operates at the level of individual processor instructions and also requires exact knowledge of the hardware model, much like classic static timing analysis.

SPTA in fact uses as input the possible latencies that every instruction may take and the probability associated with them. More details on the description of this method may be found in [10].

2.3.5 Results for Single-Path Programs

Data fits the Gumbel distribution. We start with the hypothesis that our execution time observations fit the Gumbel distribution: $\mathcal{H}_0 : F = F_\xi$. To that end we use the ET test [27].

When applying the ET test, we check a relation that is proved equivalent to the hypothesis $\mathcal{H}_0$ in [27]. This relation verifies that a certain parameter $\hat{q}_{ET,n}$ calculable for any set of data belongs to an imposed interval. This interval is also proved calculable for any set of data. For all our data we observed that $\hat{q}_{ET,n}$ belongs to the expected interval.

Independent and identically distributed observations. We start by checking that the data are identically distributed. We apply the two-sample Kolmogorov-Smirnov [24] test in which any two sample sets are compared. The Kolmogorov-Smirnov (KS) statistic quantifies a distance between the empirical distribution functions of two samples. The *null hypothesis* $\mathcal{H}_0$ is that the both samples are identically distributed and the *alternative hypothesis* $\mathcal{H}_1$ is that the samples are not identically distributed. For our experiments we use a *significance level* $\alpha = 0.05$, which is a

Table 2.1: p-value for the identical distribution test for different values of m for a sample of N=10,000 observations. P-value for the independence test

	AT	AI	CA	CN	PU	RS	TB	TT
m	Identical distribution test (p-value)							
100	0.34	0.11	0.13	0.41	0.99	0.67	0.77	0.89
500	0.36	0.81	0.24	0.62	0.55	0.84	0.21	0.57
1000	0.16	0.75	0.91	0.88	0.71	0.16	0.92	0.36
N	Independence test (p-value)							
10000	0. 66	0.81	0.92	0.84	0.69	0.73	0.76	0.81

common choice in this type of test. The outcome of the test is called the *p-value*. If the p-value is higher than α the null hypothesis cannot be rejected, which means that both samples are identically distributed.

Table 2.1 shows the p-value obtained by applying the KS test to the execution times obtained for a sample of 10,000 observations. From this *original sample* we create two *smaller samples* of m elements. For this experiment we use three values for m: 100, 500 and 1000. We populate the smaller samples by randomly taking elements from the original sample, which ensures that the smaller samples maintain the same statistical properties as the original [24]. In our case, the property of interest is the distribution. Then we apply the two-sample KS test to the two smaller samples. In Table 2.1 we observe that in all cases the p-value is higher than 0.05 which indicates that data are identically distributed.

In order to prove that samples are independent we use the *runs test* for randomness [7]. This test is used to check whether a series of binary events can be considered to be randomly distributed, and hence independent. In this test, the null hypothesis is that the data are randomly distributed (independent) and the alternative hypothesis is that the data are not randomly distributed (not independent).

A run is defined as a sequence of identical events, preceded or succeeded by different events or no events. The runs test used here applies to binomial distributions only. For example, in a sequence 0110111, we have 4 runs $(0, 11, 0, 111)$. Our data are continuous, a cut-point must be chosen by the user so that the data are transformed into a binary sample. The average is chosen as our cut-point. If the execution time is lower than the average, we obtain 0 and in the opposite case 1.

The expectation of the number of runs R is given by: $E(R) = 2mn/N$, where m is the number of events of type 1, and n the number of events of type 2, and N is the total sample size. The variance of the number of runs R is given by $V(R) = \frac{2mn \times (2mn-N)}{N^2 \times (N-1)}$.

Let r be the number of runs measured in the sample. It is known that when m or n tend to infinity then $Z = \frac{r-E(R)}{\sqrt{V(R)}} \rightarrow N(0,1)$, where $N(0,1)$ is the standard normal distribution [7]. This means that the test statistic, Z, is asymptotically normally distributed. From the value of Z, we compute the p-value with a significance level $\alpha = 5\%$. If the p-value is higher than alpha the null hypothesis cannot be rejected,

which means that data are randomly distributed.

The last row in Table 2.1 shows the p-value computed for each EEMBC benchmark with 10,000 runs. We observe that for all the benchmarks under consideration the runs we collect from them are independent.

Minimum Number of Observations Figure 2.10 shows the variation of CRPS as we add more runs using $N_{delta} = 50$. In order to obtain the minimum number of

Table 2.2: Minimum number of observations per EEMBC Autobench benchmark

AT	AI	CA	CN	PU	RS	TB	TT
300	650	400	450	500	300	500	300

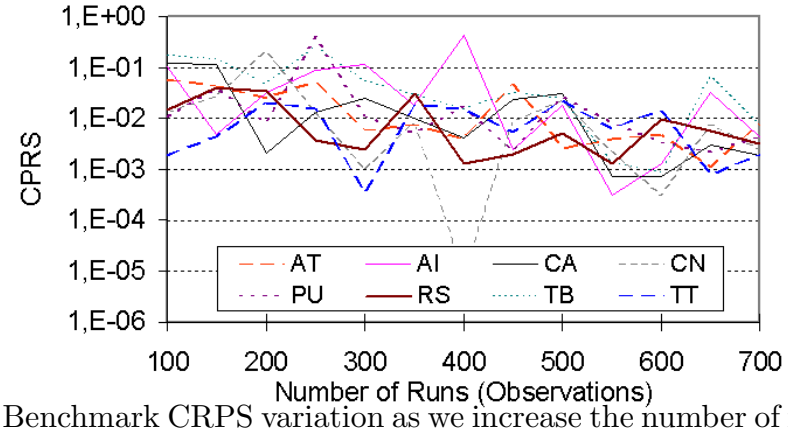
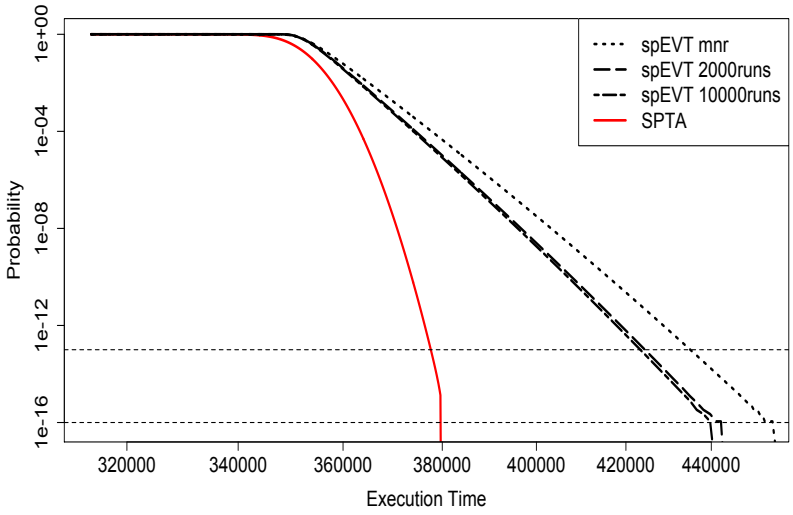


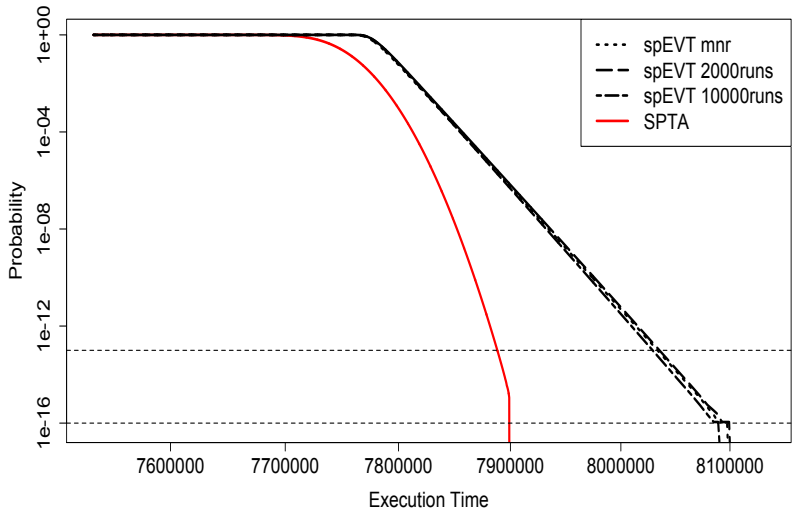
Figure 2.10: Benchmark CRPS variation as we increase the number of runs. $N_{delta} = 50$ and threshold= 0.1

observations we use the algorithm presented in Section 2.3.3, which focuses on the convergence of the EVT tail projection as more runs are considered, taking CRPS as the convergence metric. We set the threshold to 0.1 and N_{delta} to 50. We observe that although CRPS decreases, it is not monotonically decreasing. For this reason our convergence algorithm takes into account small variations. Table 2.2 shows the minimum number of runs required for each EEMBC benchmark. On average we need 425 runs per benchmark. The average time required to run an EEMBC benchmark on our simulation infrastructure is about 1 minute, so the total time required was $425 \times 1 \times 8/60 = 57$ hours. This is equivalent to less than one hour of simulations on a 64-node cluster, which is a fairly common asset in industrial production systems. It is worth noting that for building Figure 2.10 we used 700 runs, which provides satisfactory convergence for a threshold of 10^{-1}. However, to achieve convergence at lower CRPS values, e.g. 10^{-3} we would need to increase the number of observations.

pWCET estimates. Figure 2.11 shows the pWCET estimates provided by MBPTA for two EEMBC benchmarks, *rspeed* and *aifirf* (due to space constraints we cannot show pWCET estimates graphically for all benchmarks). In Figure 2.11 the x-axis shows the pWCET estimate and the y-axis shows the associated probabilities.



(a) rspeed



(b) aifirf

Figure 2.11: Single-path pWCET estimate for two benchmarks

For this experiment we use a processor setup with instruction cache only. The reason for this choice is that the application of SPTA requires the computation of execution time profiles that are hard to obtain for the setup with instruction and data caches.

SPTA and MBPTA can be projected to arbitrarily low probabilities. For the sake of illustration, we set our range of probabilities of interest to lie in the interval $[10^{-13}, 10^{-16}]$. This is based on the observation that for commercial airborne systems at the highest integrity level, DAL-A, the maximum allowed failure rate in a system component is 10^{-9} per hour of operation [2]. The highest frequency

Table 2.3: MBPTA pWCET estimates for EEMBC benchmarks w.r.t. SPTA

probability	AI	AT	CA	CN	PU	RS	TB	TT
10^{-13}	9%	5%	6%	5%	5%	2%	2%	6%
10^{-16}	15%	7%	8%	7%	7%	3%	3%	9%

at which a task can be released is current systems can be assumed to be at 10 milliseconds (10^2 activations per second). Hence, to translate the cited failure rate requirement into the equivalent of at least 10^9 hours of continuous operation (i.e., $10^2 \times 60 \times 60 \times 10^9 = 3.6 \times 10^{13}$), the pWCET of that task should have an exceedance probability in the region of 10^{-13}, which falls in the lower end of our probability range.

We present pWCET estimates for our single-path (sp) MBPTA technique when using the calculated minimum number of runs (*mnr*), 2000 and 10000 runs. For comparison purposes we show the exact execution time estimate we obtain with SPTA. We observe that, for every WCET value, we can compute its exceedance probability. The probabilistic WCET function is smooth, that is, it does not present abrupt changes. Moreover the MBPTA curve is an upper-bound to the SPTA curve. The pessimism introduced increases as the probability decreases: for probability 10^{-13} it is 2% and 9% respectively for *aifir* and *rspeed*, increasing to 3% and 15% respectively for a 10^{-16} probability. We also observe that the pWCET estimates provided by MBPTA have little variation for numbers of runs higher than the minimum number of runs: observed variations are smaller than 3% for all benchmarks for 10^{-13}.

Table 2.3 shows the pWCET estimates provided by our MBPTA technique with respect to the estimates provided by SPTA for the selected EEMBC Autobench benchmarks. We observe that for the 10^{-13} confidence level, the overestimation is less than 9% and for 10^{-16} it is less than 15% for all benchmarks, which is quite acceptable.

2.3.6 Results for Multi-Path Programs

pWCET estimates for the synthetic benchmark.

We use for the case of multi-path programs the synthetic benchmark presented in Section 2.3.4. This is a multi-path program for which we know the worst-case path, which is significantly longer than any other path. As a reference for a comparison we take the pWCET estimate that we obtain with our single-path technique when applied to the worst-case path. We compare it with the pWCET estimates that we obtain when we apply the multi-path technique to different sets of paths in which we include the worst-case path.

Figure 2.12 shows the pWCET estimate when varying the number of runs for the worst path and for the non-worst paths. We label each estimate as *xxx-yyy* where *xxx* is the number of runs we consider of non worst paths and *yyy* the number of runs from the worst case path. For every path we make at least the minimum

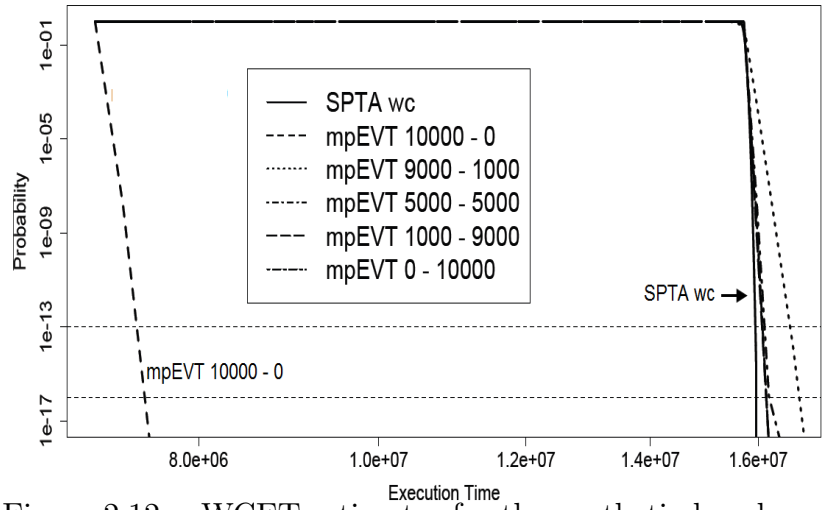


Figure 2.12: pWCET estimates for the synthetic benchmark

Table 2.4: MBPTA pWCET estimates for the Malardalen benchmarks

probability	BS	CNT	COM	CRC	INS	QSO	SEL
10^{-13}	7%	2%	4%	1%	9%	7%	5%
10^{-16}	8%	2%	5%	1%	11%	8%	6%

number of observations.

We compare all estimates with the result we obtain with SPTA for the worst-case path (*SPTA wc*). First, and most importantly, we observe that all multi-path estimates that consider the worst path are an upper bound for the SPTA. In the particular case of the 10000-0 estimate, we consider no runs from the worst path in MBPTA. This case illustrates that pWCET estimates for untested paths cannot be made. The other pWCET estimates that consider the worst path are all an upper bound for the SPTA and have a small variation between them, which indicates that the frequency at which the worst-case path is exercised with the given input data set does not affect the provided WCET estimate.

pWCET estimates for real benchmarks. As in the previous experiments, the data passed all Gumbel fitting and i.i.d tests. We compare the pWCET estimate when we consider 10,000 runs from the worst-case path (wc) against the case in which we consider 10,000 runs from all paths including the worst-case path (all). Figure 2.13 shows the results for the *select* benchmark. Again we observe that as soon as the worst-case path is considered as part of the data set, MBPTA provides results comparable to considering only the worst-case path. Table 2.4 shows the pessimism of the latter over the former, which is always less than 11%.

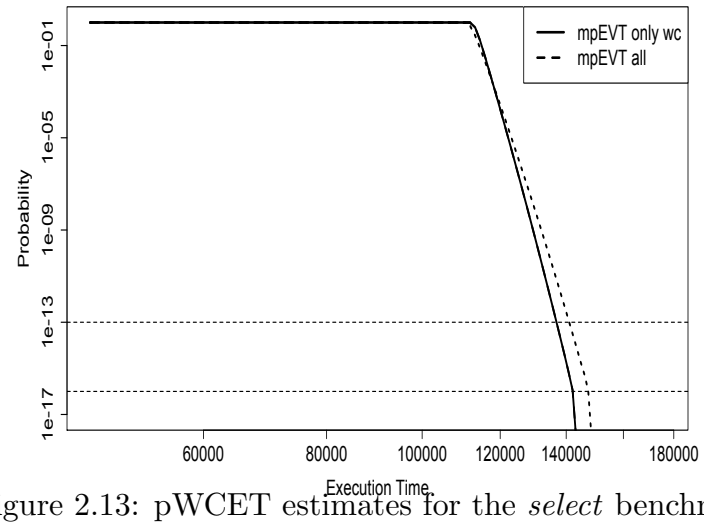


Figure 2.13: pWCET estimates for the *select* benchmark

Chapter 3

Perspectives

The arrival of complex architectures (e.g., use of caches, many-cores) increases the timing variability of the programs and thus the worst-case values are larger. For instance the utilization of many-cores implies significant increase in complexity and non-uniform memory model means huge difference between average cases and worst-cases. In this context deterministic worst-case approaches propose solutions based on over-dimensioned architectures that not all real-time systems may afford. Existing results indicate that worst-case values have a low probability of appearance (10^{-45}) if compared to certification standards (10^{-9} for the highest level of safety in avionics). Any solution taking into account this information will decrease importantly the over-dimensioning of the architectures. Probabilistic approaches provide such solutions by associating to the possible values of a parameter its probability of appearance.

These solutions are, generally, accepted as possible for soft real-time systems, e.g., media processing or wireless networks. Proposing probabilistic approaches for hard real-time systems is feasible by proving that **the probabilistic real-time approaches are as safe as deterministic real-time approaches.** In our opinion this proof should be done by solving the following challenges:

- **[C1]** the definition of the "good" statistical properties for a real-time system. The challenge [C1] is the most important as [C2] and [C3] can not be solved without a solution to [C1]. This challenge is also the most difficult as it is at the border between Computer Science and Mathematics. Nevertheless it is not impossible as several research fields have faced this challenge previously and solutions were provided (cryptography, biology, medicine).

  For real-time systems "attacking" this challenge should be done in two parts. First a definition for the *relevant sample* should be provided. For instance, for the estimation of the pWCET of a program we need to provide the definition of relevant sample with respect to the input data for the program when measurement-based approaches are proposed. This relevant sample would al-

low to estimate the probability distribution of the worst-case execution time of the program.

By providing a method for obtaining a relevant sample, then we may move to the second part of this challenge that is the definition of *random real-time*. Currently our community uses (pseudo-)random generators from other research fields (especially from cryptography) in order to generate testing data.

- **[C2]** proposition of a complete methodology for many-core platforms. Current results indicate that multiprocessor platforms have no future for real-time systems because of energy restrictions [19] and the same considerations seem to push to use rapidly many-cores [6]. This challenge requires the most important effort among the three challenges. Nevertheless it is the one taking less risks as the current results need to be extended to these platforms. Based on different requirements from different industries we may expect to have different solutions for each industry.

- **[C3]** certification of probabilistic real-time methods. This challenge needs a realistic strategy of collaboration with the industry. Leading such activities is at the border between research and transfer and it is better to be led by industrial partners. Nevertheless the research needs to provide the arguments to convince the community and the certification authorities.

Besides the safety aspects, the proof of safeness for probabilistic approaches has to offer means to respond to the increasing complexity of systems that often integrate components for which the real-time designer does not have the intellectual property. Statistical methods by their property of black boxes allow to provide solutions for this context.

The results synthesised in this thesis provide the basis for all three challenges enumerated earlier. For instance, defining the relevant sample with respect to the pWCET will use the pWCET estimation as measure on the universe of possible input data. Two sets of input data may be compared with respect to the pWCET estimates that they provide.

When moving to many-cores platforms any pWCET estimation needs to take into account an appropriate definition of what is a WCET. Issues like predictability should be studied before advancing on this problem.

The certification of probabilistic methods should pass by the validation of these methods. For instance for the same set of execution times a MBPTA approach must provide as result the same (or acceptable close) probability distribution for repeated applications of the MBPTA.

Bibliography

[1] SoCLib. `http://www.soclib.fr/trac/dev`.

[2] Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. *ARP4761*, 2001.

[3] L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *the 19th IEEE Real-Time Systems Symposium (RTSS98)*, pages 4–13, 1998.

[4] L. Abeni and Buttazzo G. QoS guarantee using probabilistic deadlines. In *the 11th Euromicro Conference on Real-Time Systems (ECRTS99)*, 1999.

[5] T.P. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *the 10th International Conference on Principles of Distributed Systems (OPODIS2007)*, 2007.

[6] S. Borkar and A. Chen. The future of microprocessors. *Communications of ACM*, 54(5), 2011.

[7] J. Bradley. *Distribution-Free Statistical Tests*. Prentice-Hall, 1968.

[8] I. Broster and A. Burns. Applying random arrival models to fixed priority analysis. In *the WiP session of the 25th IEEE Real-Time Systems Symposium (RTSS04)*, 2004.

[9] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. *A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms*. Springer, 2005.

[10] F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. D. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. Proartis: Probabilistically analyzable real-time systems. *ACM Trans. Embedded Comput. Syst.*, 12(2s):94–114, 2013.

[11] S. Cho, S. Lee, S. Ahn, and K. Lin. Efficient real-time scheduling for multiprocessor systems. *IEICE Trans. Commun.*, E85-B(12):2859–2867, 2002.

[12] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. *the 11th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2006)*, 2006.

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

[14] L. Cucu-Grosjean. Probabilistic real-time schedulability analysis: from uniprocessor to multiprocessor when the execution times are uncertain. *RR-INRIA*, 2009.

[15] L. Cucu-Grosjean and J. Goossens. Predictability of fixed-job priority schedulers on heterogeneous multiprocessor real-time systems. *Information Processing Letters*, 110(10):399–402, 2010.

[16] L. Cucu-Grosjean and J. Goossens. Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms. *Journal of Systems Architecture - Embedded Systems Design*, 57(5):561–569, 2011.

[17] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzeti, E. Quinones, and F.J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *the 24th Euromicro Conference on Real-time Systems (ECRTS)*, 2012.

[18] L. David and I. Puaut. Static determination of probabilistic execution times. In *the Euromicro Conference on Real-Time Systems(ECRTS)*, 2004.

[19] R. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv.*, 43(4):35, 2011.

[20] J.L Díaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, López J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In *the 23rd IEEE Real-Time Systems Symposium (RTSS02)*, 2002.

[21] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *the 22nd IEEE Real-Time Systems Symposium (RTSS01)*, 2001.

[22] B. Efron and R. Tibshirani. An introduction to the bootstrap. *Chapman & Hall/CRC Monographs on Statistics & Applied Probability*, 1994.

[23] D. Faranda, V. Lucarini, G. Turchetti, and S. Vaienti. Numerical convergence of the block-maxima approach to the generalized extreme value distribution. Technical Report arXiv:1103.0889, Cornell University Library, March 2011.

[24] W. Feller. *An introduction to Probability Theory and Its Applications.* Wiley, 1996.

[25] M.K. Gardner and J.W. Lui. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS99)*, pages 44–58, 1999.

[26] M.K. Gardner and J.W. Lui. Analyzing stochastic fixed-priority real-time systems. In *the 5th International Conference on Tools and Algorithms for the Construction and Analusis of Systems*, 1999.

[27] M. Garrido and J. Diebolt. The ET test, a goodness-of-fit test for the distribution tail. In *Methodology, Practice and Inference, Second International Conference on Mathematical Methods in Reliability*, 2000.

[28] B.V. Gnedenko. Sur la distribution limite du terme maximum d'une seris aleatoire. *Annals of Mathematics*, 44:423–453, 1943.

[29] J. Goossens. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints.* PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 1999.

[30] J. Goossens. Scheduling of offset free systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 24(2):239–258, 2003.

[31] J. Goossens and R. Devillers. The non-optimality of the monotonic assignments for hard real-time offset free systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 13(2):107–126, 1997.

[32] J. Goossens and R. Devillers. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *the 6th International Conference on Real-time Computing Systems and Applications*, 1999.

[33] J. Goossens, S. Funk, and S. Baruah. EDF scheduling on multiprocessors: some (perhaps) counterintuitive observations. In *the 8th International Conference on Real-Time Computing Systems and Applications*, 2002.

[34] D. Griffin and A. Burns. Realism in Statistical Analysis of Worst Case Execution Times. In *the 10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, 2010.

[35] E. Grolleau, J. Goossens, and L. Cucu-Grosjean. On the periodic behavior of real-time schedulers on identical multiprocessor platforms. *CoRR*, abs/1305.3849, 2013.

[36] E. Gumbel. *Statistics of Extremes.* Columbia University Press, 1958.

[37] J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET benchmarks – past, present and future. In *the International Workshop on Worst-case Execution-time Analysis*, 2010.

[38] R. Ha and J.W.S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *the 14th IEEE International Conference on Distributed Computing Systems (OPODIS94)*, 1994.

[39] S. Han and M. Park. Predictability of least laxity first scheduling algorithm on multiprocessor real-time systems. In Springer, editor, *Emerging Directions in Embedded and Ubiquitous Computing*, volume 4097, pages 755–764, 2006.

[40] J. Hansen, S Hissam, and G. A. Moreno. Statistical-based wcet estimation and validation. In *the 9th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2009.

[41] Jeffery P. Hansen, John P. Lehoczky, Haifeng Zhu, and Ragunathan Rajkumar. Quantized edf scheduling in a stochastic environment. In *IPDPS*, 2002.

[42] D. Hardy and I. Puaut. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. In *the 21st International Conference on Real-Time and Network Systems (RTNS) RTNS*, 2013.

[43] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[44] G. Kaczynski, L. Lo Bello, and T. Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *the 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA2007)*, 2007.

[45] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium (RTSS)*, 1990.

[46] J.P. Lehoczky. Real-time queueing theory. In *the 10th IEEE Real-Time Systems Symposium (RTSS96)*, pages 186–195, 1996.

[47] C.L. Liu. Scheduling algorithms for multiprocessors in a hard real-time environment. *JPL Space Programs Summary 37-60(II)*, pages 28–31, 1969.

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

[49] J. López, J. Díaz, J. Entrialgo, and D. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems*, pages 180–207, 2008.

[50] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of real-time embedded systems. In *IEEE Real-Time Systems Symposium (RTSS)*, 2012.

[51] S. Manolache, P. Eles, and Z. Peng. Schedulability analysis of applications with stochastic task execution times. *ACM Trans. Embedded Comput. Syst.*, 3(4):706–735, 2004.

[52] S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Trans. Embedded Comput. Syst.*, 7(2), 2008.

[53] Cristian Maxim, Adriana Gogonel, Dorin Maxim, and Liliana Cucu. Estimation of Probabilistic Minimum Inter-arrival Times Using Extreme Value Theory. In *the 6th Junior Researcher Workshop on Real-Time Computing (JRWRTC)*, 2012.

[54] D. Maxim. *Probabilistic Real-Time Systems*. PhD thesis, University of Lorraine, 2013.

[55] D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. Davis. Optimal priority assignments for probabilistic real-time systems. In *the 19th International Conference on Real-Time and Network Systems*, 2011.

[56] D. Maxim and L. Cucu-Grosjean. Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *IEEE Real-Time Systems Symposium (RTSS 2013)*, 2013.

[57] D. Maxim, M. Houston, L. Santinelli, G. Bernat, Robert I. Davis, and L. Cucu-Grosjean. Re-sampling for statistical timing analysis of real-time systems. In *the 20th International Conference on Real-Time and Network Systems (RTNS)*, 2012.

[58] A.K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983.

[59] L. Palopoli, L. Abeni, D. Fontanelli, and N. Manica. An analytical bound for probabilistic deadlines. In *the 24th Eumicro Conference on Real-time Systems (ECRTS2012)*, 2012.

[60] L. Palopoli, D. Fontanelli, Nicola Manica, and L. Abeni. An analytical bound for probabilistic deadlines. In *the 24th Euromicro Conference on Real-time Systems (ECRTS)*, 2012.

[61] J. Poovey. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University, 2007.

[62] K. Refaat and P.-E. Hladik. Efficient stochastic analysis of real-time systems via random sampling. In *the 22nd Euromicro Conference on Real-Time Systems (ECRTS 2010)*, pages 175–183, 2010.

[63] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *the 2nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS95)*, pages 164–174, 1995.

[64] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 1995.

[65] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quinones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *the 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2013.

[66] L. Yue, I. Bate, T. Nolte, and L. Cucu-Grosjean. A new way about using statistical analysis of worst-case execution times. *ACM SIGBED Review*, September 2011.

[67] H. Zhu, Jeffery P. Hansen, J. Lehoczky, and R. Rajkumar. Optimal partitioning for quantized edf scheduling. In *IEEE Real-time Systems Symposium (RTSS)*, 2002.