# Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey

FRANCISCO J. CAZORLA, IIIA-CSIC and Barcelona Supercomputing Center
LEONIDAS KOSMIDIS and ENRICO MEZZETTI, Barcelona Supercomputing Center
CARLES HERNANDEZ, Universitat Politecnica de Valencia and Barcelona Supercomputing Center
JAUME ABELLA, Barcelona Supercomputing Center
TULLIO VARDANEGA, University of Padova

The unabated increase in the complexity of the hardware and software components of modern embedded real-time systems has given momentum to a host of research in the use of probabilistic and statistical techniques for timing analysis. In the last few years, that front of investigation has yielded a body of scientific literature vast enough to warrant some comprehensive taxonomy of motivations, strategies of application, and directions of research. This survey addresses this very need, singling out the principal techniques in the state of the art of timing analysis that employ probabilistic reasoning at some level, building a taxonomy of them, discussing their relative merit and limitations, and the relations among them. In addition to offering a comprehensive foundation to savvy probabilistic timing analysis, this article also identifies the key challenges to be addressed to consolidate the scientific soundness and industrial viability of this emerging field.

CCS Concepts: • **Computer systems organization → Embedded systems**; **Embedded software**; **Real-time systems**; • **Software and its engineering → Empirical software validation**; • **Mathematics of computing → Probabilistic representations**;

Additional Key Words and Phrases: Worst-case execution time, probabilistic analysis

**ACM Reference format:**
Francisco J. Cazorla, Leonidas Kosmidis, Enrico Mezzetti, Carles Hernandez, Jaume Abella, and Tullio Vardanega. 2019. Probabilistic Worst-Case Timing Analysis: Taxonomy and Comprehensive Survey. *ACM Comput. Surv.* 52, 1, Article 14 (February 2019), 35 pages.
https://doi.org/10.1145/3301283



This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2015-65316-P, the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 772773), and the HiPEAC Network of Excellence. Jaume Abella was partially supported by the Ministry of Economy and Competitiveness under a Ramon y Cajal postdoctoral fellowship (RYC-2013-14717). Enrico Mezzetti has been partially supported by the Spanish Ministry of Economy and Competitiveness under Juan de la Cierva-Incorporación postdoctoral fellowship No. IJCI-2016-27396.
Authors' addresses: F. J. Cazorla, L. Kosmidis, E. Mezzetti, and J. Abella, Nexus-II building, Jordi Girona, 29, 08034 - Barcelona (Spain); emails: {francisco.cazorla, leonidas.kosmidis, enrico.mezzetti, jaume.abella}@bsc.es; C. Hernandez, Camino de Vera, s/n, 46022 - Valencia (Spain); email: carherlu@upv.es; T. Vardanega, Via Trieste 63, 35121 - Padova (Italy); email: tullio.vardanega@math.unipd.it.

0360-0300/2019/02-ART14 $15.00
https://doi.org/10.1145/3301283

## 1 INTRODUCTION

Embedded real-time systems are increasingly interwoven in the control of critical elements of human life, including the health, security, and safety concerns of it [42]. The dependence of the latter on the former rests on the ability of the system software to carry out the assigned critical control functions effectively, in conformance with the applicable safety regulations. As part of that trend, software has become the main value-added vector for most embedded real-time products. In consequence of that, the nature of the critical functionalities implemented in software has changed from rather confined procedures to large, articulate, diverse, and complex algorithms that oversee multiple information flows between vast arrays of sensors and actuators. In the automotive domain, for instance, the quantity of software embedded in cars already exceeds 100 million lines of code [29], with performance requirements predicted to rise by two orders of magnitude by 2024 [17]. Similar trends occur in other application domains (cf., e.g., [43] for space).

At the processor level, the use of more advanced acceleration features is the only practical means to sustain the ever-increasing demand of guaranteed performance put forward by value-added application software. Not surprisingly, this trend has caused a sweeping transition from simple 8- and 16-bit micro-controllers to more complex processors in new-generation systems for the aerospace, automotive, and rail domains. For example, existing 32-bit processors in current automotive subsystems embed accelerator-based multicore units, like in NVIDIA DrivePX [4], RENESAS R-Car H3 [3], QUALCOMM Snapdragon 820 [2], and Intel Go [61].

Timing is one of the main non-functional concerns in embedded real-time systems. Timing analysis aims to ascertain whether software programs execute within the bounds assigned to them at specification, which normally regard duration (aka worst-case execution time, WCET) and completion (aka response time). The former type of analysis seeks to upper bound the execution duration of individual software units considered in isolation. The latter assesses whether a feasible (concurrent or parallel) compositional schedule of those software units exists that allows them to complete their work in a timely manner, that is, within assigned deadlines that respond to high-level needs of the system's functionality, after factoring in the impact of the interference among them. Arguably, the hardest challenge of WCET analysis is to comprehend the extent of execution-time variability (i.e., jitter) that software programs may exhibit when run on their target platform.

- For simple platforms, the sources of jitter (i.e., the features that contribute to the variability of the program's execution time across runs) are limited to the software program's structure and its input data. The former reflects how multiple identical runs of the program may take different durations depending on the execution path taken by the program. The latter reflects how the input data affect programs execution; for instance, determining the branch taken in the program path or the duration of jittery operations, for example, floating point operations.
- As platform complexity increases, new low-level and hard-to-catch sources of jitter emerge, which include cache utilization, bus occupancy, and parallel contention for shared resources in multicores. Those elements may have a dominant impact on the program's jitter.

The increase in complexity at software and hardware level has resulted in a relentless quest for novel timing analysis methods capable of mastering it. The hardness of that challenge has caused a surge of interest in the use of statistical and probabilistic techniques, owing to their ability to reason on (black-box) observations, which are orders of magnitude easier to obtain than (white-box) knowledge on the relevant internals of an execution. A wide range of works exist that extends from tailoring statistical techniques used in other domains to fit the timing analysis problem, to dressing hardware to better match the premises of probabilistically analysable behaviour.

Probabilistic techniques have been explored in virtually all aspects and layers of the system overall, including application programs, the operating system, the compilation system and its linker, the processor hardware, and, of course, the very fabric of the timing analysis itself.

While there is growing evidence that probabilistic and statistical techniques may help alleviate some of the limitations that hamper traditional timing analysis solutions, the proliferation of works that have appeared in the last few years is making it hard for interested individuals to understand the forming landscape of research in that area. Arguably, one pressing need for all observers of the present state of the art is to appreciate how those novel techniques relate to one another, to comprehend the assumptions on which they build, and to determine how they contribute to earning industrial acceptance for probabilistic-based timing analysis.

This is what this article aims to contribute:

(1) A comprehensive critical taxonomy of the existing works on probabilistic timing analysis, clarifying which specific problems they aim to attack, how probabilistic and statistical techniques are exploited (thereby exposing the limitations of each proposal), and where different proposals overlap or complement one another.
(2) A scrutiny of the assumptions that each surveyed technique makes on the underlying execution platform and the application timing behaviour, in the intent of clearing the ground for well-founded use of probabilistic analysis.

The remainder of this article is organized as follows: Section 2 presents the motivation and intuitions behind the application of probabilistic approaches to the timing analysis problem, and proposes a comprehensive taxonomy of the relevant state of the art. Sections from 3 to 8 survey the principal lines of work in that taxonomy. Section 9 concludes by discussing the overall status of probabilistic timing analysis, for theory and technology, and its readiness for industrial use.

## 2 PROBABILISTIC WCET ANALYSES

### 2.1 Setting the Scene

Deriving high-quality WCET estimates is universally accepted as a challenging task [112]. When facing the WCET problem, industrial practitioners screen the state of the art of solution offerings, to determine which approach best meets their needs and obligations. Ultimately, industrial choice seeks to balance the cost-effectiveness of the solution and the quality of the evidence that it can provide to attain the level of confidence required by the domain prescriptions [7].

Two dominant flavors of timing analysis exist:

— Static methods (STA, in the sequel) seek absolute theoretical rigor, at the cost of complexity in use and pessimism in the results that may inordinately increase as system design becomes more complex. The quality of their results strictly depends on the availability of accurate and sufficient information on the hardware and software internals (and in particular on their timing behaviour) of the system to analyse. For this reason, [50] notes that mathematical rigor per se, while obviously beneficial, is insufficient to assure that the provided estimate always upper-bounds actual execution times.
— Measurement-based methods (MBTA, in the sequel) follow less rigorous approaches (a vulnerability that must be addressed with utmost care), for much lower cost of use. Those methods seek to provide empirical evidence that the worst-case conditions of interest have been exercised or closely approximated in the measurement observations.

MBTA is the most commonly used technique in industry, owing to its low cost-benefit ratio. Evidence exists that measurements can be also used for functions with the highest criticality level,

for example, DAL-A in avionics [75]. Yet, STA is the preferred solution for the software functions at the highest criticality levels. To contain the costs of qualification (or certification, where required), those functions typically execute on simple processors and represent a modest fraction of the value-added software in the system overall [47].

The advent of high-performance hardware presents STA and MBTA with hard challenges. STA is especially affected in its quest for accurate timing models to compute the cost of individual processor instructions and the basic blocks of software programs. The presence of intellectual property restrictions and the vastness of hardware documentation increases the risk of considering inaccurate or incomplete timing information [7], ultimately causing STA users to depend on measurements and reverse engineering for filling the consequent information gaps [93]. MBTA suffers too, as further sources of jitter appear in hardware resources (e.g., bus occupancy, cache placement) that are difficult to study or observe in isolation, for the lack of sufficient hardware monitors or specification information.

Another problem with high-performance computing systems is that, no matter how deterministic the individual hardware components may be, the fabric of their interaction tends to grow exceedingly complex, causing an explosion in the state space on which the program's timing behaviour depends *together with* a potentially large increase in the overall jitter. The former consequence makes it difficult to determine the worst-case scenario; the latter increases pessimism.

The application of probabilistic reasoning to timing analysis aims at mitigating some of the limitations of the existing techniques.

- The probabilistic variant of MBTA, aka measurement-based probabilistic timing analysis (MBPTA), aims at easing the construction of qualification-worthy arguments that the worst-case scenarios of interest have been captured in observation runs, in a manner that lives up to the increase in hardware complexity.
- The probabilistic variant of STA (SPTA) aims at reducing the pessimism incurred by traditional WCET analysis, while also reducing the information need. SPTA builds on the notion of an execution-time profile (ETP) that describes the probabilistic execution-time distribution of individual instructions. ETPs are an attribute of static instructions at binary level as they are assumed to upper-bound all of their (dynamic) instances (i.e., the executions of that instruction during program runs).

The application of either variant of PTA requires changes to the system for it to conform with the analysis assumptions. The state of the art includes PTA works that address (i) systems where no hardware or software changes had been applied (COTS), (ii) systems that employed COTS hardware but used software modifications transparent to the application (SWRand), and (iii) systems whose processors had been modified in selected components (HWRand).

### 2.2 The Wisdom of Probabilistic Reasoning on WCET

PTA requires one to cease seeking single-valued WCET estimates, as unique definitive upper-bounds to all instances of execution-time duration, to concentrate instead on a distribution function, aka probabilistic WCET (pWCET), thus modelling the maximum probability with which a WCET bound can be exceeded. PTA is a broad paradigm, which encompasses all timing analysis methods that yield pWCET estimates regardless of the specific prerequisites that each such method may impose on the characteristics of the execution platform and its modelling, the measurement collection protocol, the existence and the modelling of dependencies, and the degree of knowledge or control required from the user. To put the PTA paradigm shift in prospective, one should appreciate that the assurance attached to the single-valued product of STA rests on the quality of

Table 1. Terms and Acronyms Used Throughout This Article

| Term | Definition | Term | Definition |
|---|---|---|---|
| MBPTA | Measurement-Based Timing Analysis | SPTA | Static Probabilistic Timing Analysis |
| PTA | MBPTA and SPTA | ETP | Execution-Time Profile |
| EVT | Extreme Value Theory | i.i.d. | Independent and Identical Distribution |
| PoT | Peak Over Thresholds | BM | Block Maxima |
| GEV | Generalized Extreme Value Theory | GPD | Generalized Pareto Distribution |
| COTS | Commercial Off the Shelf | HWRand | Time-Randomized Hardware |
| SWRand | COTS HW w/ Software Randomization | pWCET | Probabilistic WCET |
| ATD/ATS | Analysis Time Distribution/Sample | OTD | Operation Time Distribution |

the information passed to it (e.g., processor model, instruction timing, flow facts), which, however, cannot be quantified in the general case [50].

In acknowledgment of the lack of absolute certainty, no critical real-time embedded system is conceivably designed assuming absence of timing failures (i.e., task overruns). Instead, systems are designed so that no such event should ever cause the system to enter an unsafe state: If it did, that would be a single point of failure, consequent to inadequate safety-case design. Domain-specific standards require system engineering to encompass a safety process that sanctions the strategy required to mitigate the system-level risk of hardware or software malfunctions. Various mechanisms (e.g., replication, online monitoring, watchdog) are then deployed to detect and react to undesired situations, in accord with the criticality of the system part of interest.

Interestingly, probabilistic reasoning is habitually used in the design of electronic components [13], to model the appearance of certain types of hardware faults such as, for instance, random hardware faults due to particle strikes from the outer Space or other forms of electromagnetic interference. Probabilistic reasoning on WCET bounds matches that reality quite naturally.

### 2.3 A Taxonomy of PTA Works

The initial works on PTA, which date back to the early 2000s, reason theoretically on the problem of probabilistically modelling the timing behaviour, without discussing the mathematical means to derive pWCET estimates and the computing platforms on which PTA could be safely applied. About a decade later, that research area boomed, under the spin of a series of EU-funded projects and a constellation of industrial collaborations born around them. In the last few years, the number of works that address PTA in various guises has risen dramatically, making it difficult to discern the relation that they have with one another. In that vastness, several claims are made that seem to discord. Overall, the current landscape of research in that field has developed into a haphazard territory, ungainly for scientific investigators or industrial practitioners.

This section attempts to categorize all the PTA works known to date. Works with similar goals fall in the same category, although some works span multiple categories as they address several problems at once. Before drawing the taxonomy, we note that probabilistic *and* statistical reasoning take different roles in the application of PTA: probabilistic analysis reasons *a priori* on the timing behaviour that the program will exhibit during operation; statistical analysis is performed *a posteriori* on a set of execution-time measurements to check whether the hypothesis made on the program's probabilistic timing behaviour cannot be rejected. The union of those techniques (applied either to SPTA or MBPTA) is collectively referred to as PTA. Table 1 lists the main terms and acronyms used in this article.

In this article, we first classify the surveyed works according to whether they pertain to MBPTA or SPTA, and then to each of the related sub-problems. Subsequently, we single out further works of transversal interest. Figure 1 provides a pictorial representation of the proposed taxonomy.

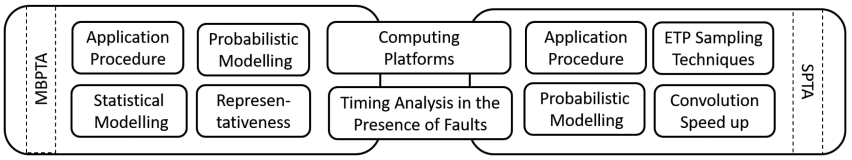



Fig. 1. The principal categories in our proposed taxonomy.

The MBPTA category, which we survey in Section 3, further breaks down as follows.

— *Probabilistic Modelling* (Section 3.2) seeks *a priori* guarantees that the time-randomized resources used in the execution platform are apt to yield a timing behaviour that allows probabilistic reasoning.
— *Statistical Modelling* (Section 3.3) discusses how to ascertain the statistical properties of the execution-time distribution of programs obtained from its runs on the target platform. This represents the *a posteriori* counterpart of probabilistic modelling.
— *Application Procedure* (Section 3.4). The works in this category present MBPTA as an orderly set of actions, and illustrate how to apply them.
— *Representativeness* (Section 4) aims at providing evidence that the execution-time observations captured for a given software unit during analysis are representative of the distribution of the program's execution times during operation. To reflect their importance, we survey works in this category in Section 4, outside of Section 3's internal hierarchy.

The SPTA category, which we survey in Section 5, further breaks down as follows (as probabilistic modelling is intrinsic to SPTA, it is not covered as a separate subcategory).

— *Application Procedure* (Section 5.2). The works in this category present the SPTA application process in general and discuss software-related challenges with it, such as path coverage. This ambit includes the SPTA correspondent of *Probabilistic Modelling* assurance, to confirm that the execution platform offers sufficient randomization to allow probabilistic reasoning. To date, SPTA research has been shown able to meet this requirement only for processor architectures considerably simpler than those amenable to MBPTA.
— *ETP Sampling Techniques* (Section 5.3). SPTA applies the convolution operator on the ETPs of individual instructions to derive their combined ETP. The number of elements in ETPs increases exponentially with the convolutions performed. Works in this category limit the number of points per ETP while ensuring that the resulting estimates do upper-bound the original non-sampled ETPs.
— *Convolution Speed-up* (Section 5.3). The works in this category propose techniques to reduce the computation overhead entailed by the use of convolution operators.

Works of transverse concern to MBPTA and SPTA fall in the following categories.

— *Computing Platforms* (Section 6). The works in this group discuss the main requirements on the design of the platform's hardware and software to which PTA can best apply.
— *Timing Analysis in the Presence of (Hardware) Faults* (Section 7). These works address the impact of faulty hardware on the program's execution time, including its WCET.

Table 2 relates the categories in our PTA taxonomy and the HWRand, SWRAnd, and COTS platform types (cf. Section 2.1) to the coverage of the corresponding topic in the state of the art.

Table 2. Coverage of the Related Work for Each PTA Taxonomy Category and Platform Type

| Platform | MBPTA | | | | SPTA | | | | Analysis in the Presence of Faults |
|---|---|---|---|---|---|---|---|---|---|
| | Application Procedure | Probabilistic Modelling | Statistic Modelling | Represen-tativeness | Application Procedure | Probabilistic Modelling | ETP Sampling | Convolution Speed-up | |
| HWRand | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SWRand | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | - |
| COTS | ✓ | - | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ |

Legend: ✓: covered; ✗: not studied; (−): not relevant.

### 2.4 Probabilistic Analysis Beyond WCET Estimation

The notion of pWCET has been also explored at higher levels of abstraction, particularly for scheduling and response time analysis: the work by Burns et al. [24] is the main reference in this field. However, since our survey focuses on the use of PTA for pWCET estimation rather than on the use of the obtained pWCET estimates, for example, for schedulability analysis, we only briefly touch upon the latter in this work.

The works on probabilistic scheduling consider tasks whose execution time follows a probabilistic distribution that can be characterized, without discussing how such a distribution is derived, which instead is of essence to PTA. Other works (e.g., [36], [72]) study system-level effects of scheduling, which have bearing on execution time and therefore on pWCET estimates, specifically the cache evictions that a task may suffer because of preemption. No other system-level effects have been studied so far in the context of PTA research.

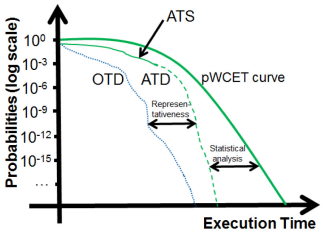
To address the scheduling problem from the PTA perspective, it is *not* strictly necessary to create a probabilistic version of it. In fact, instead of exposing the whole pWCET distribution function to the scheduling or response time analysis algorithm, one can simply select a single-valued WCET estimate for each task, at a cut-off probability that meets the safety-case requirements of the system (e.g., $10^{-15}$ per execution), and use standard scheduling approaches with them.

## 3 MBPTA

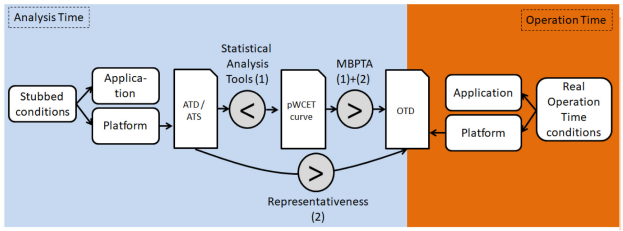
### 3.1 Introduction

Like any other measurement-based timing analysis method, MBPTA requires collecting execution-time measurements as soon as possible in the system development lifecycle, thereby avoiding the hazard of costly, late-stage regression determined by unsatisfactory analysis outcomes. Measurements are normally taken at the level of individual program units of variable granularity points in the system hierarchy, as soon as coded, after stubbing their externals. The challenge is to create analysis scenarios that, by virtue of configuration and execution procedure, expose the software program to the maximum extent of variability that may arise during operation, as a result of input received, unit state, and execution conditions (the latter including the contention incurred from co-runners on access to shared hardware resources).

The elements to consider to that effect are referred to as sources of jitter (SoJ). The distinguishing trait of MBPTA is to use analysis-time observations to derive a probabilistic bound on the program's execution time that applies to its behaviour during operation. This ability requires attaining statistical control on the sources of jitter. Figure 2(a) illustrates this notion by contrasting the analysis-time distribution (ATD) of the program's execution time, as determined from very many observations, commensurate with the degree of assurance sought, with the operation-time distribution (OTD) that will begin to emerge after final system integration, too late to serve for WCET analysis. MBPTA aims at ensuring that an ATD, with the upper-bounding characteristics shown in Figure 2(a), can be constructed off a small number of selected observation samples (ATS), and then used to derive a pWCET distribution function that upper-bounds the ATD.




(a) ATD, ATS, OTD and pWCET estimate.




(b) The space covered by MBPTA, inclusive of the representativeness and statistical concerns that are prerequisite to its application.

Fig. 2. The basics of MBPTA.

The MBPTA problem overall can thus be decomposed into two parts:

(1) To ensure that the ATD upper-bounds the OTD. This requires assuring that the analysis-time conditions are no better (for the emergence of the WCET) than those that can arise during operation. This problem is usually referred to as *representativeness*, to signify the preoccupation that the execution conditions incurred at analysis are representative of those that may occur at operation.
(2) To soundly apply statistical methods so that an affordably small sample of the ATD (a proper choice of ATS) can be drawn to derive a pWCET estimate that does upper-bound the ATD (and thus the OTD).

In the literature, the MBPTA denomination has often been used indistinctly to address any application of statistical and probabilistic approaches to measurement-based timing analysis. This misclassification matches the statistical part of the MBPTA work flow as understood in this article, but fails to capture the relevance of having a well-formed MBPTA process to guarantee a sound application of it. The latter is a crucial concern to the correct interpretation of MBPTA. Figure 2(b) provides a pictorial representation of the MBPTA proper process. Accordingly, this survey classifies current MBPTA works into three areas of concern:

(1) Probabilistic and statistical modelling (Section 3.2 and Section 3.3, respectively), where we discuss the works that address the probabilistic and statistical perspective of the MBPTA process at the high level, without entering its concrete steps of execution.
(2) MBPTA application procedure (Section 3.4), where we review the works that concentrate on procedural issues, focusing on particular stages of the MBPTA process.
(3) Representativeness (Section 4), which considers the works that study how to ensure that the platform, for its hardware or software components and the execution conditions that they allow exploring, guarantees that the ATD upper-bounds the OTD. The works in this category either address the program structure (e.g., the coverage of the program's control-flow graph) or concentrate on the execution conditions determined by the processor hardware.

### 3.2 Probabilistic Modelling

To apply probabilistic reasoning to the timing of a software system soundly, the program execution time *must* have a probabilistically characterizable behaviour. The prime means used by MBPTA research to that end, is to inject randomization into the program's timing behaviour. Different means to do so, however, fare very differently with respect to representativeness: we return to this

issue specifically in Section 4. Probabilistic modelling therefore aims to assure that the approach followed to inject randomization adequately yields probabilistic timing behaviour.

MBPTA notably differs from SPTA in the way it addresses probabilistic modelling. As we shall see in Section 5, the latter requires deriving exact or upper-bounding probabilities for each time event that is randomized (e.g., a cache miss). The former may instead follow the design and implementation prescriptions in [71] to assure that the timing events that affect the program's execution-time behaviour have a random nature (and therefore allow probabilistic reasoning on them), so that it does *not* need to know or compute their actual probabilities, for they are bound to emerge through statistically significant observations.

Overall, MBPTA's probabilistic modelling requires that randomization means are deployed to ensure that each execution time resulting from a program run has a probability of occurrence. MBPTA research has attempted to achieve this goal by randomizing the timing of individual components [71], by randomly sorting or picking from measurements [51, 81], and by adding random padding to observed measurement values [78]. Not all randomization means, however, are equally capable of providing sufficient representativeness guarantees: the user shall therefore thoroughly understand the implications of randomization means on representativeness.

### 3.3 Statistical Modelling

Broadly speaking, when statistical reasoning enters timing analysis approaches, execution-time observations are sampled according to given criteria, and then used to fit some probability distribution, which yields the pWCET estimation.

To the best of our knowledge, the vast majority of works in this area have used Extreme Value Theory (EVT), paralleling the worst-case execution-time behaviour of a program to an extreme-value probability distribution. A residual fraction of works have explored other theories such as Copulas [22] or Markov models [39, 110]. Owing to this dominance, this section concentrates on EVT and its use for the pWCET estimation problem. It is worth noting that, while MBPTA is *not* intrinsically restricted to yielding continuous distributions, the use of EVT naturally implies delivering them. Conversely, SPTA, albeit not necessarily limited to discrete distributions, often builds on discrete representations of the execution times and their individual probabilities.

*3.3.1 Basics on EVT.* EVT is a well-established branch of statistics that models the probability of occurrence of extreme events (whether maxima or minima) in a given distribution. EVT has been traditionally applied to meteorological, hydrological, insurance, and financial problems [5, 37, 45], to predict extreme behaviour or expectations, such as exceedance probability or return periods.

EVT builds on the Fisher-Tippett-Gnedenko [49] and Pickands-Balkema-deHaan [97] theorems, which stipulate that the asymptotic tail distribution of a sample of independent and identically distributed (i.i.d.) random variables converges to specific families of distributions, known as Generalised Extreme Value (GEV) and Generalized Pareto Distribution (GPD).

To model the tail (hence, the extreme) probability distribution, EVT singles out the tail values in the input sample of observations. To select those elements from the sample population and fit them to a (parametric) extreme distribution, EVT uses one of two methods, namely, Block Maxima (BM) or Peak over Threshold (PoT), briefly reviewed below. Goodness-of-fit tests or other statistical diagnostic tools eventually determine how well the obtained distribution models the population.

***Block Maxima*** *(BM).* BM filters out non-tail values by splitting the sample into smaller blocks of a given size, and then retaining only the maximum value in each block. EVT then attempts to fit the resulting set of values within the GEV family of distributions, whose parametric form is shown in Equation (1) [32, 73] (where $1 + \xi(\frac{x-\mu}{\sigma}) > 0$ must hold), which resolves into a Gumbel, Reversed Weibull, or Fréchet distribution. The parameters $\mu$, $\sigma$, and $\xi$ are known as the *location*, *scale*, and

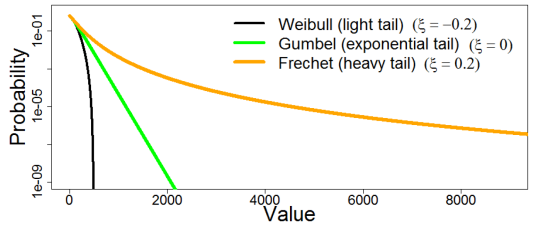



Fig. 3. Complementary cumulative distribution function for light, exponential, and heavy-tail GEV distributions, with $\xi = -0.2$, $\xi = 0$, and $\xi = 0.2$, respectively, and ($\mu = 0$, $\sigma = 100$) for all of them.

*shape*, respectively. The shape $\xi$ determines whether the resulting distribution is a Weibull (aka light or short-tailed), when $\xi < 0$, or Gumbel (aka exponential or light-tailed), when $\xi = 0$, or Fréchet (aka heavy-tailed), when $\xi > 0$.

$$G(x;\mu,\sigma,\xi) = \begin{cases} exp\left[-\left(1+\xi\frac{x-\mu}{\sigma}\right)^{-1/\xi}\right] & \xi \neq 0 \\ exp\left[-exp\left(-\frac{x-\mu}{\sigma}\right)\right] & \xi = 0. \end{cases} \tag{1}$$

Figure 3 illustrates example tail shapes: a Weibull distribution with $\xi = -0.2$, which has a steep slope and converges to a maximum value (500 in the example, not shown in the plot); a Gumbel distribution with $\xi = 0$, which yields a relatively short-tailed slope that, however, remains asymptotic (outside of the plot); and a Fréchet distribution with $\xi = 0.2$, which decreases polynomially.

***Peak-over-Threshold*** *(PoT).* PoT filters out non-tail values by retaining only the observations that exceed a given threshold. EVT then attempts to fit the resulting set of values within the GPD family of distributions, Gumbel, Reversed Weibull, or Pareto, whose parametric form is shown in Equation (2), where the parameters $\mu$, $\sigma$, and $\xi$ have a similar meaning as in the GEV formulation (in fact, $\xi$ is identical), and $x > \mu$. The GPD distribution admits also a two-parameter formulation.

$$H(x;\mu,\sigma,\xi) = \begin{cases} 1-\left[1+\xi\left(\frac{x-\mu}{\sigma}\right)\right]^{-1/\xi} & \xi \neq 0 \\ 1-exp\left(-\frac{x-\mu}{\sigma}\right) & \xi = 0. \end{cases} \tag{2}$$

[32] notes that a strong correlation exists between GEV and GPD: for the same $\xi$ and similar values for $\mu$ and $\sigma$, GPD and GEV result in the same distribution.

*3.3.2 Modelling Extreme Timing Behaviour with EVT.* In spite of the remoteness of its original domains of application to the timing analysis problem, EVT has emerged as an apt tool to derive trustworthy pWCET estimates. Numerous studies and considerable effort have contributed to yielding a sound adaptation of it. From the statistical modelling perspective, among the steps that precede the application of EVT, the state-of-the-art literature distinguishes (1) the approach used for the derivation of GEV/GPD parameters, (2) the filtering of tail values from the sample, and (3) the evaluation of the quality of fit.

Edgar and Burns first proposed using EVT—the Gumbel distribution in particular—for pWCET estimation [44]. Their work uses a notion of threshold, referred to as "confidence level," to determine which distribution to use. With that, they fit the full sample of measurement observations to the given Gumbel distribution (without using BM or PoT) to ascertain that the accumulated execution time of individual tasks (assumed to be independent) does not exceed its pWCET distribution at that confidence level. The authors of [55] apply BM instead, and fit a Gumbel distribution to the resulting selection, using Chi-squared tests [96] to assess its goodness. The soundness of the resulting pWCET estimate is then assessed against a very large sample of measurement observations.

[103] compares the use of BM and PoT, also studying their sensitivity to the input parameters. Griffin and Burns [51] explain that modelling discrete processes (execution times measured in clock cycles) with continuous distributions such as Gumbel's may lead to pessimistic pWCET estimates. The cited authors also illustrate the difficulties of preserving the i.i.d. properties required for the application of EVT,[1] and propose guidelines to mitigate the corresponding risks and derive reliable pWCET estimates. Lima and Bate [78] elaborate further on the problem posed by the use of discrete data, which may impede a proper use of EVT, and suggest adding random padding to the execution-time measurements to circumvent that problem and apply EVT soundly. Yue et al. [81] address the lack of independence in the dataset, by proposing an alternative data collection method. They propose retaining only the highest values from potentially dependent execution-time measurements, repeating this process until a sufficient number of maxima are obtained that can be shown independent, so that EVT can be reliably applied to them. The same authors also suggest computing multiple pWCET estimates (with repeated applications of the proposed method) and the corresponding confidence interval, so that a given pWCET estimate at the chosen confidence level can be selected. Lima et al. [79] analyse scenarios where the whole GEV family of distributions can be applied: those where the input data come from multiple distributions (i.e., from input values that may cause execution to traverse different program paths), as well as computing platforms with or without time randomization. In order to create a source of randomness and thus obtain sufficient variability in the measurements, the authors apply random sampling across observations. This particular work does not discuss the representativeness of the studied scenarios with respect to the execution conditions expected at operation.

Abella et al. [8] employ the Coefficient of Variation method (CV) [38] to determine all GPD parameters except the shape, and compute the best (presumed) exponential distribution that fits the data, after confirming that the exponentiality assumption cannot be statistically rejected. The CV method differs from that in [34], which selects the parameters that best fit the data regardless of the distribution family, before choosing a shape parameter that matches an exponential distribution.

***Statistical Requirements for Data Samples.*** A fundamental statistical requirement in the original formulation of EVT is that the observations need to be i.i.d. [49, 51, 55]. For the purposes of pWCET estimation, satisfying the independence requirement may be impaired by data collection practices or even inherent effects of the execution platform: [22] studies the effect of dependencies across execution-time measurements of program components. In fact, it has been shown that EVT can be used to analyse stationary processes, where dependence exists across variables, but only as long as the dependencies are managed conveniently to warrant the reliability of the resulting distribution [32, 73]. Santinelli et al. [103] analyse the impact of stationary processes in pWCET estimation, and conclude that dependencies across execution-time measurements may cause pWCET underestimation. Along the same line, Melani et al. [85] single out the individual factors that may lead to dependencies, such as cache state modifications across runs and scheduling policies. That work concludes that, while those factors cause significant dependencies across measurement observations, they can be accounted for in the use of EVT via appropriate independence and correlation tests. The cited authors also suggest that when those factors combine with some random phenomena (such as, e.g., random choice of program input), the dependencies become less important. As noted earlier, [78] shows that adding a random padding to the measured values also decreases dependencies across them. Whilst this may allow preserving the base EVT assumption of i.i.d. random variables, it remains to be proven constructively whether using data padding to model a dependent process as an i.i.d. process does always lead to reliable pWCET estimates.

[1] This work does not consider stationary processes where some degree of dependence can be tolerated.

*3.3.3 Other Statistical Approaches.* Although preponderant, EVT is not the only statistical and probabilistic technique that has been used with MBTA. Bernat et al. in [22] propose a hybrid MBTA method where empirical execution-time profiles are gathered from observations taken at the level of the program's basic blocks, and then combined together to determine an end-to-end pWCET distribution. While this work predicates on the notion of pWCET distribution, it applies no predictive model to determine it: the pWCET distribution is derived by conservatively combining the execution-time profiles associated to the program's basic blocks over its control-flow graph. In this respect, this work is much closer in concept to SPTA than to MBPTA.

In a similar vein, [110] addresses the problem of determining the execution time of soft real-time systems, where probabilistic deadlines are defined, which can be missed with a given probability. The cited work focuses on application scenarios common to the robotics/image-recognition field, where the computation time depends on the complexity of the current real-world situation. In this case, the i.i.d. property does not hold. To overcome this shortcoming, the authors propose to model the timing behaviour using a hidden Markov model to represent different execution modes (associated to states in the Markov model) and valid transitions across them. In that manner, the authors infer states and transitions directly from sequences of execution-time observations, thus modelling residual dependencies accurately. At that point, the execution-time distribution for each state can be described with an independent random variable.

### 3.4 MBPTA Application Procedure

The EVT-based element of MBPTA requires its application procedure to proceed across four distinct steps, possibly iterated multiple times, while always paying attention to representativeness (which we discuss separately in Section 4).

(1) *Statistical verification*: where the set of collected observations are checked against the prerequisites for the application of EVT (i.e., i.i.d. and stationarity).
(2) *Data filtering*: the dataset is filtered (with either BM or PoT) to retain only the values that belong to the tail of the execution-time distribution of the program of interest.
(3) *Parameter selection*: where a specific distribution family is selected, along with the set of parameters that correspond to either the GEV or GPD equations.
(4) *Distribution fitting*: where the pWCET curve that arises from the above parameter choice is fitted against the data sample.

The MBPTA approaches in the state of the art differ in the specific assumptions that they make and the technique that they use to perform those steps.

A first attempt to establishing a solid and repeatable EVT-based MBPTA application procedure for multi-path programs appeared in [26, 34]. The former work [26] provides a high-level discussion of those requirements; the latter [34] presents a procedural description of the proposed application procedure. Those works build on the HWRand platform with explicit enforcement of the worst-case initial state before each measurement to meet the i.i.d. statistical requirements of EVT, while also assuring representativeness; BM is used to sample the collected observations and then the best-fit *location*, *scale*, and *shape* GEV parameters are estimated. At that point, the *shape* parameter is tested against the exponential hypothesis with the Exponential Test (ET) [41]. If passed (when $\xi \approx 0$), $\xi$ is forced to 0, to match a Gumbel distribution. The cited authors postulate that a *minimum number of observations* [34] can be incrementally determined by checking that the pWCET curve becomes *stable* (i.e., it does not change significantly when feeding further observations to the procedure). The Continuous Ranked Probability Score (CRPS) [46] is used to determine the closeness of the distributions obtained at each round of application. The proposed method is then applied to multi-path programs assuming the user is responsible for providing the

input data that cause the traversal of all the program paths of interest to the pWCET computation. All those paths are then jointly and indistinctly assumed to contribute to the sought distribution.

For both HWRand and SWRand platforms, the authors of [8] present an MBPTA approach that uses the Coefficient of Variation (MBPTA-CV), already outlined in Section 3.3.2. The authors postulate that the exponential tail can be always used to model the pWCET, backing their claim with the time-bounded nature of real-time programs of interest to WCET analysis, the characteristics of the execution platform, and the granularity level (per path) at which EVT is applied. In contrast with [34], MBPTA-CV uses PoT instead of BM, thus achieving reproducibility (i.e., yielding the same output when applied to the same data). As a further point of difference, MBPTA-CV fits the best exponential tail to the data, instead of fitting the best GEV distribution and then forcing $\xi = 0$, which may not be the best exponential fit for the data.

[53] defines a framework for the application of EVT, to ascertain the applicability of EVT to execution COTS platforms that do not employ randomization. [102] follows suit, using PoT (hence, GPD) and proposing that an array of statistical tests should be passed with a given confidence level to guarantee a *statistically reliable* application of EVT. The same confidence levels are subsequently used to sustain the *reliability* of the results, in contrast with [8, 34], which assess the quality of the pWCET distribution indirectly, as part of the parameter selection step.

[21] uses EVT to analyse the timing behaviour of highly parallel applications running on GPGPU. The MBPTA application procedure in the cited work broadly aligns with [26, 34] except that the authors' interest is more centered on the assessment of the EVT statistical requirements, owing to the looser independence conditions of their problem.

Lesage et al. [77] present an evaluation framework to assess the reliability of MBTA. The proposed approach combines timing data with compiler-generated structural information, to guide the construction of synthetic path traversals (random walks) that can be used to test the robustness of the analysis approach, especially when different coverage conditions are met. The authors' framework is then instantiated to the MBPTA context, using a maximum envelope[2] to compute a single pWCET distribution out of observations from multiple program paths. Experimental results confirm that the quality of the results is highly dependent on the attained path coverage.

### 3.5 Summary

Existing MBPTA approaches prevalently base on EVT and all share similar procedural steps. Individual approaches differ in the assumptions that they rest on and in the statistical tools that they use. At the outermost level, the research proposals are divided between those that prescribe the use of randomization in the execution platforms [8, 34] and those that assume unmodified (time-deterministic) platforms [21, 53]. Experimental evidence shows that randomized execution platforms facilitate meeting the statistical prerequisites of EVT. When the alternative approach is pursued, the method used to collect the measurements must be studied with care (cf. Section 4).

The use of BM or PoT appears to be equally valid, when used within sound methods.

Predefined governing assumptions on the timing distribution of the target program generally guide the (GEV or GPD) parameter selection, with the Gumbel (GEV) or Exponential (GPD) distribution being normally regarded as the most appropriate choice for the problem domain [8, 34, 44].

No universal consensus exists to date on how to assess the quality of the pWCET distribution derived with EVT. Some works [53, 102] use the confidence levels obtained from the assessment of statistical prerequisites, to indirectly evaluate the quality of the EVT results, which, however, postulates a strong correlation between prerequisites and results. The quality of the model is

[2]A maximum envelope of a set of distributions stands for the distribution that, for each exceedance probability in a set of input distributions, takes the maximum pWCET value across them.

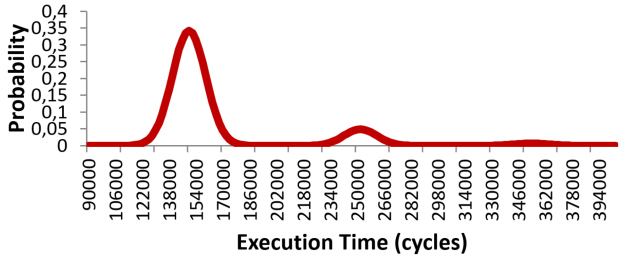



Fig. 4. An exemplary Probability Distribution Function of the execution time of a program.

typically ascribed to the goodness of the parameter selection [8, 34], where a more accurate selection is bound to yield better results. Other approaches [79] suggest the adoption of standard statistical diagnostic tools, such as the Quantile-to-Quantile [28] or the Mean-Excess plots [20]. Both solutions are complementary and not really alternative. And yet, any claim on the soundness of the MBPTA results (over and above the use of EVT) must address the representativeness concern.

## 4 REPRESENTATIVENESS

### 4.1 Introduction

For all measurement-based analysis techniques, the concern of representativeness regards the ability to assure that the execution conditions encountered when collecting the observations correlate significantly with those that can occur during operation. The umbrella term "execution conditions" refers to all the factors that may affect the timing behaviour of a program when run on a given processor platform, including, for example, its memory layout (which determines cache mapping and may thus have a large impact on the cache behaviour) and the parallel contention load on shared hardware resources in multicores.

The representativeness concern is external to EVT, as EVT treats the system of interest as a black box, without considering the system's internals when the observations are taken. Thus, the EVT projections only hold for the "world" as seen during the analysis. Whereas the events observed at different moments of the same analysis (i.e., in different runs of the same program) may contribute to one combined probability distribution, EVT has nothing to say on unobserved events.

This notion is better understood with Figure 4, which portrays for illustration purposes the *real* (hence, conceptual) probability distribution function (PDF) of the execution time that a program can take. An artifact of this kind cannot be obtained in the general case as only a (small) sample of the corresponding observations can really be taken with finite effort, so that an empirical PDF is obtained instead. The PDF presents three peaks with decreasing densities, at 150k, 250k, and 355k cycles, respectively. Each such peak is caused by a particular set of (not necessarily disjoint) execution conditions. Assume that the execution conditions that trigger the right-most peak would *not* occur in the analysis sample. In that event, EVT is unable to capture the right-most peak in the tail (which, without seeing sufficient elements of it, could be located arbitrarily far to the right of the abscissa). Applied to this example, the representativeness concern reduces to the need to assure that the analysis observations capture all the execution conditions of interest to pWCET analysis, so that EVT can factor all due peaks in its predictions.

While representativeness is acknowledged as a critical issue for EVT predictions [32], a number of works either assume it to be given or outside of scope, or conjecture the execution conditions incurred in the input sample collection *do* suffice for pWCET estimation [44, 51, 55, 78, 79, 81, 103]. [25] notes, however, that the user may often be unable to control the low-level hardware features that have bearing on representativeness.

Arguably, representativeness plays a key role in the pWCET estimation process. Platform characteristics and means to account for different program paths may lead to obtaining either a single execution-time sample (hence, a single execution-time distribution) or multiple ones. In the latter case, different approaches have been followed to obtain a single pWCET estimate (i.e., a single execution-time distribution) from the multiple distributions sampled.

### 4.2 Program-Structure Representativeness or Path Analysis

The variable set of program paths that observation runs may traverse is one of the main sources of jitter in the program's execution-time behaviour, which impends on the representativeness concern. Several probabilistic approaches have been proposed to address that challenge.

*4.2.1 Applying EVT or Other Statistical Means to Multiple Program Paths.* To use MBPTA for multi-path programs, one needs to determine how (i) to apply EVT to those program paths; and (ii) to guarantee that the obtained pWCET results are valid for all the possible path traversals that may occur during the nominal operation of the analysed program.

As we noted in Section 3.3.3, [22] collects execution-time measurements for small program units (basic blocks or functions) that correspond to different execution paths, and uses their frequency of observation to construct an ETP for each such structural unit. At that point, the cited work uses appropriate mathematical operators to combine the individual ETPs into a pWCET for the whole program, in a manner that upper-bounds the execution time of any single end-to-end path traversal. The authors use convolutions to combine the ETPs of independent program units and define an extended form of *joint convolution* for units that are known to have dependencies or, conservatively, where dependencies cannot be excluded.

Other authors [34, 79] propose merging the execution-time measurements taken across multiple program paths into a single bucket, which is then passed in input to EVT. [102] shows that proceeding in that way does not work in all cases. Milutinovic et al. [92] discuss the risk of the single-bucket approach, which causes the pWCET to reflect the path traversal frequency as occurred during analysis, which the user may have great difficulty at correlating with the operational behaviour of the program. Authors further show that, when using Gumbel distributions, the single-bucket approach may lead to pWCET estimates that either are optimistic (i.e., not upper-bounding) for some, possibly all, individual program paths or exceed all of them, thus incurring unwanted pessimism. [92] recommends the multiple-bucket approach, instead, where each execution path is analysed separately, and a joint pWCET estimate (known as the *maximum envelope*) is obtained as the pWCET profile that upper-bounds the distributions computed for all individual traversed paths, and for each exceedance probability of interest. The multiple-bucket method can be adopted by all approaches in the state of the art that originally apply the single-bucket method ([34, 79, 102]).

[8] follows the multiple-bucket approach and pays special attention to the curve fitting step of the MBPTA application procedure. The authors maintain that the Exponential distribution (GPD) or Gumbel distribution (GEV) are always a safe choice for modelling the worst-case execution-time behaviour of a real-time software program, and propose an analysis procedure to be applied on a per-path basis. The authors' method seeks representativeness with a combination of upper-bounding precautions and time randomization in the execution platform.

*4.2.2 Assuring MBPTA Results for All Program Paths.* This problem is inherent to all measurement-based analysis approaches, which can only speculate on the observations made during the analysis and therefore produce results that are only valid as long as those observations are representative of the worst-case execution conditions (including worst-case path traversal). Curing this problem by requiring the user to achieve full path coverage in the analysis observations is not a tenable option in the general case. This problem, generally referred to as *path-representativeness*,

is bypassed in [34] by predicating on the user's ability to discern the *relevant* paths in a program. This assumption, however, can hardly be generalized. Other works [79, 82] suggest making random picks across all possible input data to the program, to select a random subset of program paths and associated execution conditions. This method, however, offers no absolute guarantees of coverage.

A number of methods propose to synthetically extend the degree of path-representativeness that the user-provided input data can attain in the general case [67, 114], so that the pWCET distribution obtained with MBPTA is trustworthy, even when the path traversals made during the analysis cannot be guaranteed to have included the worst-case path. [67] presents the Path Upper Bounding (PUB) technique to artificially balance the branches of all individual conditional control flow constructs in the program so that any branch is a safe upper-bound for all the other branches in the same construct. Balancing is obtained with an extended binary version of the target program (used at analysis time only), where additional core and cache access instructions are inserted as needed on the individual branches of the conditional construct. The execution-time measurements collected for the extended version of the program thus are by construction an upper bound to all possible path traversals in the original program, which is the one that is eventually deployed at operation. The degree of overestimation incurred by balancing depends on the particular structure of the program under analysis. The main drawback of PUB is that it needs a qualified compiler to generate a semantically preserving extended version of the original program. Indeed, those compiler transformations may be overly difficult for complex programs; for example, those that use "goto" or breaks in loops.

With a similar intent, Ziccardi et al. [114] propose the Extended Path Coverage (EPC) technique, a fully automated approach to artificially extend the set of measurements to achieve the same effect of full path coverage. EPC relies on the concept of *probabilistic path independence* to generate a set of *synthetic* observations that complement the set of measurements actually collected. EPC operates at the level of basic blocks whose execution-time observations are synthetically made independent from any specific path in the program, so that they can be used as building blocks to derive end-to-end measurements for unobserved paths. Feeding EVT with the union of measurements from observed and unobserved path traversals achieves the same effect as full-path coverage, without any additional burden on the user. EPC requires that the original observations assure basic block coverage: this is a realistic requirement in some application domains, certainly lighter than its full coverage alternative.

Both PUB and EPC have been developed for HWRand platforms, but in principle they might be adapted to SWRand platforms too. Instead, their direct use for time-deterministic platforms would either fail or deliver too pessimistic results (e.g., assuming that almost all cache accesses are misses).

### 4.3 Platform-Related Representativeness

The use of complex processors induces hardware-related sources of jitter that need to be accounted for in any claim of representativeness. The MBPTA research in this regard follows two trails: one that addresses time-deterministic platforms, which embed no MBPTA enabler underneath the application; the other that studies how time randomization, transparently implemented in hardware or software or both, helps meet the MBPTA application requirements.

*4.3.1 Time-Deterministic Platforms.* A reliable application of EVT for pWCET estimation requires the user to accurately understand the population of hardware events that may occur during operation [25] and determine whether and how far analysis-time observations can trigger them. Admittedly, however, enumerating all sources and all combinations of execution conditions with impact on the program's execution-time behaviour is an intractable problem in the general case.

Moreover, the cost and complexity of constructing an input dataset that triggers each and every such event is out of reach for most users. In fact, to the best of our knowledge, no work exists to date that solves this problem in a sufficiently general manner.

In [82], the authors propose random sampling across a program's input values as a means to explore different execution conditions. This approach, however, does not allow drawing solid conclusions on the coverage achieved. Interestingly, an approach of this kind may indeed produce data samples capable of passing i.i.d. tests (except for programs with low variability in execution-time behaviour), and thus lead to mechanically successful applications of EVT but that are *unfit* for pWCET computation, owing to fundamental deficiencies in representativeness.

[83] studies measurement protocols in general, that is, abstract procedures to follow for collecting execution-time measurements, without adapting them to any specific platform. This work suggests that a measurement protocol *converges* by adding more measurements to the analysis sample, given that a larger sample delivers a pWCET estimate closer to the pWCET that would have been obtained using the whole population of execution times. A measurement protocol achieves representativeness if there exists a number $k$ of execution-time observations that allows obtaining a pWCET close enough to the actual (ideal) WCET. Notably, [83] offers only a qualitative argument to support the claim of convergence, which has not been sustained with quantitative evidence.

*4.3.2 Time-Randomized Platforms.* Several works present hardware designs that aim to help achieve representativeness. All those proposals explore ways to control the jitter of certain processor resources. We briefly survey each of them in isolation, noting, however, that they can be combined opportunistically in accord with the characteristics of the platform and the needs of the application.

*Probabilistic Upper-Bounding.* Time-randomized processor architectures aim at helping the user achieve representativeness with less effort [71]. This approach attacks the hardware sources of jitter with high-enough impact to justify design change. When the events that they generate become random, then the user problem reduces to assuring that the quantity of observations made is sufficiently large for the whole spectrum of variation to manifest itself. To put it simply, for a random source of jitter with probability of appearance of $\mathcal{P}_e = 0.1$ per use, the probability of *not* observing it in $R = 1,000$ single-use runs is $\mathcal{P}_{no} = (1 - P_e)^R = (1 - 0.1)^{1000} \approx 4.3 \times 10^{-5}$. The domain regulations will then determine whether such a probabilistic assurance level is sufficient.

*Deterministic Upper-Bounding.* Processor resources whose jitter is low enough to not justify time randomization but not insignificant for the analysis, are modified to allow the user to set them to respond with their worst-case latency during analysis [71]. In that manner, the jitterless timing behaviour of those resources during analysis is assured to upper-bound their jittery behaviour during operation. A typifying example of these resources is the floating point unit (FPU), some of whose operations may take a variable response time depending on the operands. It would be unthinkable for the user to study the probability distribution of operands during operation and to figure how to reproduce it during analysis. The ability to force the FPU to always respond with worst-case jitter per operation type during analysis relieves the user from that burden. An approach of this kind has been used also to study contention effects with hardware shared resources [12, 95].

*Padding.* An alternative to touching the hardware design is to artificially enlarge the collected execution-time values before passing them to EVT [40]. The increment factor is designed, on per processor resource basis, to capture the maximum time overhead that such resource might cause in the observed use. This approach was used in [40] to upper-bound the impact of contention on access to hardware shared resources in a multicore. Unlike deterministic upper-bounding, this technique is pessimistic in that the padding may yield jitter costs higher than the absolute maximum.

*4.3.3 Cache-Centric Solutions.* [9] shows that, with time-randomized hardware, all high-impact sources of jitter, except for cache placement, can be observed sufficiently well to apply EVT to them. This is not surprising as caches are one of the processor resources with the highest impact on the program's WCET. As such, the quest for determining the worst-case behaviour of caches has attracted the attention of the WCET research community [56].

For set-associative caches, the de-facto standard in real processors, the events of interest to EVT originate when the number of program's code or data objects mapped to the same set exceeds the set capacity (the number of ways) [9]. In fact, this scenario may cause an abrupt increase in the miss ratio, and a correspondingly large inflation in the program's execution time. If events of this kind can happen with a non-negligible probability during operation, they have to be captured at least once in the observation runs performed during analysis, in order for EVT to account for them [9].

The role of caches in the representativeness problem is discussed in [101] and [88]. [69] and [70] study how that problem changes when the caches use random placement or modulo placement combined with software randomization techniques.

*Hardware-Randomized Caches.* In non-parametric random placement caches [23, 104, 109], placement considers only the address of the memory request. As a result, a given memory layout determines a single cache placement, much like with conventional deterministic caches based on modulo placement. The former and the latter therefore have the same limitation with respect to representativeness: they are exposed to highly unlikely but extremely heavy pathological cases.

Time-randomized caches [59, 64, 66], on the other hand, feature parametric placement functions, which allocate data in cache lines based on a combination of the request address and an arbitrary random number. By changing the random parameter across runs, those designs yield a different cache placement for each execution, which allows addressing the representativeness problem by sampling observations across distinct program runs.

[9] shows that, for hash random-placement caches with $S$ sets, it is possible to determine whether at least $W + 1$ program objects out of the total $K$ under consideration are mapped to the same cache set, by considering all potential mappings of the $K$ objects to the $S$ cache sets and the fraction of those mappings where at least one cache set has $W + 1$ objects allocated to it. The result of this operation can be approximated with weak compositions. A weak composition of an integer $n$ is a way of writing it as the sum of a sequence of non-negative integers. The cited work studies all the weak compositions of $K$ made of exactly $S$ parts, where at least one part is higher than $W$. From this analysis, the cited work derives the probability of the cache event of interest, $P_{ce}$, and the probability of not observing it in $R$ observation runs, $\mathcal{P}_{no} = (1 - P_{ce})^R$. One proposed solution then consists in increasing $R$ until $P_{no}$ becomes small enough to be acceptably ignored. One other solution is to repeat the analysis with a cache with a smaller number of sets, to increase the probability of occurrence of the cache event of interest to the point of capturing it with higher probability with $R$ observation runs.

[19] presents an alternative to weak compositions using the multinomial coefficient that derives exact results instead of approximations. To mitigate the non-negligible computational costs of using the multinomial coefficient, the cited work proposes Monte Carlo simulations to approximate the probability of the event of interest, with a given precision.

[90] notes that the previous solutions assume that the impact of all addresses on execution time is similar. For instance, given three addresses, $A$, $B$, and $C$, which access a direct-mapped cache, those techniques assume that mapping $A$ and $B$, $A$ and $C$, or $B$ and $C$ to the same set has the same impact on the program's execution time. This might be so when the addresses are accessed homogeneously. Yet, in the general case, not every combination of addresses—when mapped to the same set—results in an execution-time increase of the same magnitude. To address this challenge, the cited work proposes *ReVS*, a computationally intensive method to compute exactly

the probability and impact that each cache allocation can have on execution time. For sets with $W$ ways, the proposed solution takes in input the sequence of accessed addresses of the program and builds all combinations of $W + m$ addresses (with $m \geq 1$) out of the $K$ addresses in the program. The impact that each such combination can have on the miss count is determined with a cache simulator, whereas their probability is derived analytically. With this analysis, the user can assess whether $R$ observation runs are sufficient to capture the cache allocations with low probability and high impact. Otherwise, more runs are needed until the condition is satisfied.

[89] presents a less effort-intensive variant of that solution. Instead of running simulations in the cache simulator for all combinations of $W + m$ or more addresses, it uses a quantitative metric to single out the combinations of $W + m$ that may have high enough impact.

*Software-Randomized Caches.* [18] studies the difference between software- and hardware-randomized caches, for programs with homogeneously accessed objects. The authors show that, for hardware-randomized caches, the probability $\mathcal{P}_S$ of an object to be assigned to a given set for $S$ sets is $1/S$. With software-randomized caches, dependencies exist among the sets allocated to a given object and those that can be assigned to another. Accordingly, the probability of allocating an object to a given set reduces as more objects are already allocated to it. This dependency causes the probability of the cache events of interest to differ for software-randomized caches from their hardware counterpart.

### 4.4 Summary

Representativeness is a key concern for all measurement-based timing analysis approaches. Its problem space spans the Cartesian product of all distinct sources of jitter of interest, both high-level, such as program paths, and low-level, such as stateful processor resources. The MBPTA results are valid only as long as the execution conditions experienced during analysis are (conservatively) representative of those that may occur during operation.

The quest for sufficient path coverage in the analysis encompasses two contrasting trends. Some authors [67, 114] seek to achieve full path coverage synthetically, adding structural program knowledge to the postulate that the user-provided input data to the program assure basic block coverage. Other authors [79, 82] prefer to soften the requirements and propose sampling randomly from the input data space. While attractive from the user perspective [78, 79], solutions of the latter kind have fundamental weaknesses: they cannot yield solid (upper-bounding) correlation between the program paths traversed in the observation runs and the unseen ones; furthermore, owing to the black-box nature of EVT, they may produce results that are either unreliable or overly pessimistic [92, 102].

The same problem, only grander, presents itself for the processor resources, in the need to assure that the effects of the execution conditions incurred in the observation runs upper-bound those that may occur during operation. Expecting the user to achieve full control of the low-level factors of interest is evidently untenable, and, again, random sampling from the program's input space as proposed in [82] offers only a manner to possibly incur enough variability to satisfy EVT mechanically, but not to warrant a reliable solution for MBPTA. Low-cost hardware modifications, transparent to the application, have been studied extensively to help in building arguments of representativeness [9, 89, 90]. To become more industrially viable they have to be extended to COTS processors, which is still a work in progress to date.

## 5 SPTA

### 5.1 Introduction

Probabilistic reasoning has also entered the world of static timing analysis, which builds *a priori* models of worst-case behaviour. [35] is an early instance of that union, which does not require

any form of platform randomization, thus falling into the category of classic static timing analysis. The cited work assumes that probabilities of occurrence can be attached to the outcomes of the conditional branches in a program. By propagating that information along the abstract syntax tree of the program, and assuming that the WCET cost of individual program units be statically known, the authors compute *a priori* traversal frequencies for individual runs and consequently for execution-time values.

Conversely, the static analysis techniques that assume (require) some degree of platform randomization fall under the umbrella of Static Probabilistic Timing Analysis (SPTA), which aims to yield probability distributions for the program's execution time. The presence of a randomized cache is the fundamental assumption at the basis of SPTA research and the fundamental enabler to its state-of-the-art solutions. In randomized caches, a random eviction occurs in response to either a cache access (*evict-on-access*) [26] or a cache miss (*evict-on-miss*) [64].

SPTA solutions construct ETPs, that is, discrete representations of the *probability mass function* of the time cost of program units, which were considered for the first time in [22]. ETPs can be defined at various levels of abstraction and granularity of program execution, from individual processor instructions in the program binary (*static* view) or their executed instance (*dynamic* view, which carries contextual information akin to *loop unrolling* and *call-context* in classic static timing analysis [112]), to groups thereof. ETPs for static instructions provide a safe over-approximation over all possible executions. ETPs for dynamic instructions, instead, allow constructing a tighter bound to the actual timing behaviour. Representing the timing behaviour of single instructions as an ETP requires a precise micro-architectural model of the execution platform, a common need to static timing analysis [112]. Whether explicitly or implicitly, all current SPTA solutions work at the level of static instructions, which is deemed more robust against the lack of precise timing information, and thus less demanding to the user. Unless explicitly stated, therefore, we assume this choice by default in the sequel.

For a given (static) instruction $x$, the ETP describes the latencies $\{et_{x_i}\}$ that its execution can incur, and the associated probabilities $\{\mathcal{P}_{x_i}\}$ of occurrence. Accordingly, the ETP for instruction $x$ can be defined as follows:

$$ETP(x) = \begin{pmatrix} et_{x_1} & et_{x_2} & \cdots & et_{x_n} \\ \mathcal{P}_{x_1} & \mathcal{P}_{x_2} & \cdots & \mathcal{P}_{x_n} \end{pmatrix}, \sum_{i=1}^{n} \mathcal{P}_{x_i} = 1.$$

SPTA postulates that the ETPs of individual instructions can be composed to build up the ETP of any sequence of instructions up to the entire program. The *independence* of the ETPs of individual instructions in a sequence is guaranteed by either making assumptions on guarantees offered by the underlying hardware platform [26] or explicitly removing potential dependencies in the modelling of individual ETPs (e.g., by considering lower bounds to the probability of hitting in cache). On the assumption of independence, SPTA computes the ETP of an instruction sequence by applying *discrete convolution* ($\otimes$) to their individual ETPs. More formally, let $\mathcal{X}$ and $\mathcal{Y}$ denote the random variables that describe the execution time of instructions $x$ and $y$, respectively. Their convolution $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ is defined as

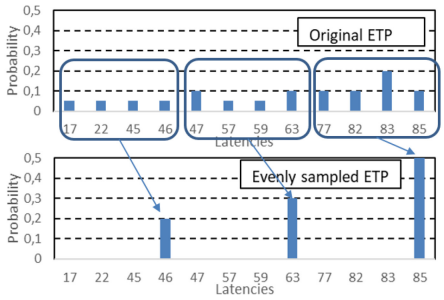
$$\mathcal{P}\{\mathcal{Z} = z\} = \sum_{k=0}^{k=+\infty} \mathcal{P}\{\mathcal{X} = k\} \times \mathcal{P}\{\mathcal{Y} = z - k\}.$$

Figure 5(a) shows an example convolution of two ETPs. The remainder of this section illustrates the SPTA application procedure and discusses how its computational cost can be mitigated.

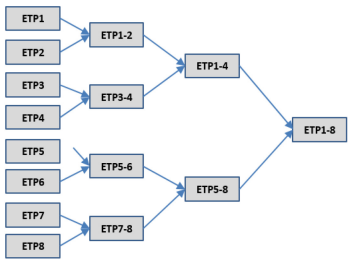
$$ETP(x) = \begin{pmatrix} 1 & 10 \\ 0.9 & 0.1 \end{pmatrix} \quad ETP(y) = \begin{pmatrix} 2 & 10 \\ 0.5 & 0.5 \end{pmatrix}$$

$$ETP(x) \otimes ETP(y) = \begin{pmatrix} 3 & 11 & 12 & 20 \\ 0.45 & 0.45 & 0.05 & 0.05 \end{pmatrix}$$

(a)




(b)




(c)

Fig. 5. SPTA basic concepts: (a) convolution; (b) sampling; (c) inter-ETP parallelization.

### 5.2 SPTA Application Procedure

The concept of ETPs and convolutions was first used by Zhou in [113], where the latencies of individual instructions are determined by the cache hit and miss latency for code and data. The hit and miss probabilities in a random cache relate to the number of memory accesses that occur between two subsequent accesses to the same memory block (aka *reuse distance*): this value provides an upper bound to the number of potentially evicting events. However, [15] shows that the eviction probabilities across instructions are not independent in `evict-on-miss` random caches, and consequently, the prerequisites for convolution do not hold. By using convolution on those dependent ETPs, Zhou obtains a probabilistic approximation of the program's execution time, which is *not* a pWCET estimate, for it is not guaranteed to upper-bound the real execution-time distribution. Hence, Zhou's approach cannot be regarded as an instance of SPTA. While such model requires only reuse distances for fully associative caches, it needs precise addresses and the corresponding cache mapping for set-associative and direct-mapped ones. In fact, such constraint also holds for all SPTA techniques considered next.

The first definition of a proper SPTA approach was presented by Cazorla et al. in [26]. The proposed solution addresses single-path programs, assuming *evict-on-access* random caches, where cache hits do cause evictions. In that work, the notion of reuse distance is applied to conservatively build ETPs that are assuredly independent and can be convolved, so that the result is a true pWCET estimate. In particular, the said authors use reuse distances to estimate lower bounds to the probability of hit of each access. By using *evict-on-access* random replacement, whether memory accesses in between two consecutive accesses to the same memory block hit or miss is irrelevant since they cause a constant number of evictions, which facilitates the construction of independent ETPs, as needed for convolution.

Subsequent works address other cache policies [36], multi-path programs [76], and attempt to improve on the analysis precision [15]. [36] extends SPTA to *evict-on-miss* random replacement caches, where eviction occurs only in the event of cache misses. [15] discusses the conditions for a correct application of SPTA and the optimality of the equations used to approximate or statically

bound the probability of hit and miss in random replacement caches. The authors of that work stress the importance of using lower bounds to hit probabilities as a means to render the ETPs of individual instructions independent. The cited work also proposes an enhanced analysis to tighten the bounds on the miss probability. Instead of working with reuse distances considered in isolation (i.e., for individual instructions), the proposed approach uses additional knowledge on the reuse distance of instruction sequences.

[14] studies the tightness of different mathematical formulations of SPTA solutions. [101] questions the viability of applying SPTA to randomized caches, on the grounds that it cannot possibly obtain results as good as those produced by deterministic timing analysis using caches with modulo placement and LRU replacement. [88] responds to that objection by noting that traditional approaches have great difficulties at working with global stack distances, across instruction sequences, as studied in [15].

Lesage et al. [76] extend the multi-path approach in [36] by leveraging the improvements in the upper bound to the cache miss probability proposed in [15]. The cited work assumes a single-level, fully associative cache with *evict-on-miss* random replacement. The same technique could in principle apply to set-associative caches, yet facing serious scalability challenges.

### 5.3 Speeding Up Convolutions

The number of elements in an ETP increases exponentially with the convolutions performed to compute it. The ETP that results from the convolution of two ETPs with sizes $m$ and $n$ ranges from $m + n - 1$ to $m \times n$ elements. Sampling techniques aim to keep that number under control, thus also reducing the quantity of computations required to compute them [84, 100].

Re-sampling manipulates the values in an ETP so that the resulting ETP has fewer elements and upper-bounds the original one, to avoid the risk of underestimation. Borrowing the definitions from [80], and noting that an ETP describes a random variable, we can see re-sampling as follows. Let $X$ and $Y$ be two random variables. $Y$ upper-bounds $X$, denoted $Y \geq X$, if $\mathcal{P}(Y \leq D) \leq \mathcal{P}(X \leq D) \forall D$.

[100] selects $k$ elements from the original $n$-element ETP and assigns the probability of the residual $n - k$ elements to the largest value of the resulting ETP. The amount of pessimism incurred by re-sampling depends on the value chosen for $k$, which is determined by the amount of time and memory available for convolutions and ETPs, respectively. Maxim et al. in [84] propose three specific re-sampling techniques. A first basic method selects equally distanced values in the original ETP and associates them with the sum of the probabilities of the ETP elements between them. This method preserves the shape of the distribution but intensifies the peaks of it. Figure 5(b) shows an example of sampling that reduces a 12-point ETP to a 3-point one. A second method attempts to minimize the number of elements in the resulting ETP, using a re-sampling pass, to force exactly the same distance among values in each operand of the ETP. In this manner, the number of distinct values produced by the convolution gets closer to the theoretical minimum. The third and more sophisticated technique collapses multiple ETP entries while seeking to minimize the pessimism incurred in aggregating them.

Other techniques study how to reduce the computational requirements of SPTA. [91] proposes precision-preserving optimizations using parallelization techniques: parallelizing the convolution procedure itself and performing multiple convolutions in parallel, as depicted, for example, in Figure 5(c). [91] presents a *discretization* technique similar to re-sampling, which reduces computational complexity at the cost of additional pessimism in the analysis. Discretization flattens the probabilities in the ETPs by rounding up the probability of the highest latency, and rounding down that of the lowest latency. In that manner, initially different ETPs eventually equalize, which yields

computational gain as the convolution of two identical ETPs can be performed much faster than the convolution of different ones.

### 5.4 Summary

Research on SPTA has made important advances lately, on methodological foundations and application procedure. Yet, numerous challenges, which we enumerate below, remain to be solved before it can become a viable alternative for industrial users.

***Computational Complexity***. Despite the considerable improvements achieved with re-sampling techniques as proposed in [84, 91, 100], the computation of sequences of convolutions still has a major computational impact. There is evident tension between performing convolutions over large sets of ETP, each of which with a non-negligible number of entries, and the risk of untenable pessimism incurred by re-sampling.

***Requirements***. While probabilistic analysis is resilient to occasional lack of information (e.g., particular addresses of given memory accesses), the quality of SPTA bounds in term of tightness largely depends on the quality and the completeness of the information available to it, much like classic static timing analysis in general. SPTA needs to know the latencies of each and every hardware operation, whose availability and reliability are not certain [7]. Moreover, as observed in [15], current SPTA formulations can only guarantee reasonably tight results if they exploit a large amount of information on execution history, which is difficult to ensure and costly to maintain. Solutions have been proposed to that end, which aim to reduce (i.e., compress) the size of the needed state information [52].

## 6 PLATFORM

### 6.1 Introduction

We now survey the PTA-related works that focus on the execution platform and propose hardware designs or system and application software solutions to facilitate the use and improve the effectiveness of PTA. To the best of our knowledge, no hardware or software support at the platform level has been proposed for the purposes of SPTA. SPTA assumes hardware architectures that feature a simple core with instruction timings that can be accurately derived, and single-level fully associative or set-associative caches, with random replacement to enable probabilistic reasoning [26]. When set-associative caches are used, SPTA works further assume that all program addresses are known so that it is possible to determine the precise cache set to which each address is mapped. To date, SPTA has not been extended to multicore contention analysis.

Instead, numerous MBPTA works propose hardware designs to improve the quality or the cost of the pWCET product. The techniques that they propose mostly seek to assure representativeness, which we discussed in Section 4. As a positive side effect, those solutions also help create execution-time distributions that favor statistical analysis, for example, by creating less discrete distributions or facilitating the probabilistic modelling of execution times with i.i.d. random variables.

MBPTA has also been applied to COTS processors. The works in this scope fit into two distinct sets: those that propose MBPTA-specific software solutions (e.g., in the compilation of the program) or SWRand (cf. Table 1); and those that use COTS architectures with no special provisions for MBPTA. The latter works fall short in representativeness, though, as discussed in Section 4.

### 6.2 MBPTA-Supportive Hardware Designs

The works in this area concentrate on the processor resources that cause jitter in the program's execution time and that are hard to model for timing analysis. Three processor resources have been

studied extensively: caches, floating point units, and the mechanisms to govern parallel contention on access to hardware shared resources in multicores. One common goal of those studies, which we surveyed in Section 4.3.2 from the standpoint of representativeness, is to propose modified designs for those resources that allow their jitter to be upper-bounded either probabilistically or deterministically [71].

*6.2.1 Cache Memories.* Caches accelerate the program's access to memory, thereby reducing its WCET, at the cost of more complex timing analysis. Complexity increases because the addresses at which the program's code and data are located—its memory layout, which determines the program's cache layout, that is, the cache sets to which the program's data and code map—are an indirect consequence of multiple (asynchronous) factors of system development. Different cache layouts can cause significant variations to the program's execution time, owing to the possibly (and normally) large imbalance between the cost of a cache miss over a hit. Even the smallest difference in the order in which the program's object files are linked together may affect the cache layout and thus impact execution time. The presence or absence of environmental variables and directives in the program's code, which one would consider irrelevant to this problem, can displace the addresses of all program objects, changing its memory layout and consequently its cache layout. Incremental integration, which is a convenient practice in industrial software development, is another source of variations of cache layouts, regardless of the logical independence between the old and the new software modules. [87] studies this problem outside of PTA.

The works that study MBPTA-supportive cache designs propose random placement in place of deterministic modulo-based policies. Random placement first appeared in [104]. The cited work proposes using a pseudo-random hash function in high-performance processors to distribute the data across cache lines and thus make the cache performance less sensitive to different placements compared to traditional modulo placement. [109] bases on the same idea but explores different pseudo-random hashing functions and provides a more complete evaluation based on superscalar out-of-order processor architectures. The cited work uses simulation to show that the proposed solution reduces conflict misses. In skewed associative caches [23], each way uses a distinct hash function for randomized placement across banks. That work shows that this solution reduces conflict misses for programs that process large matrices.

A common trait of those randomization solutions is that their placement function uses solely the address of the access. Hence, for a given memory layout, only a single placement exists for all runs of that program, which renders its effect similar to conventional deterministic architectures based on modulo placement. For this reason, these non-parametric randomized cache designs cannot be employed with MBPTA unless software randomization is used on top of them.

The time-randomized caches proposed in [59, 64] use parametric placement functions to allocate data in cache lines based on the request address *and* an arbitrary random number. It is thus sufficient to change the latter parameter (the random number) across program runs, to break the dependence between the position in memory where an object is placed, that is, its address, and the cache set to which it is mapped. This solution facilitates providing probabilistic assurance of the coverage of cache layouts that result in high execution times, aka cache risk patterns [86]. Not surprisingly, the risk of pathological cache layouts is one of the principal impediments to the unrestricted use of caches in real-time systems.

Research on random caches for real-time systems starts with [99], whose authors provide initial evidence of how randomized replacement allows quantifying the risk of pathological behaviour, which cannot be assured with deterministic cache policies. The cited work also shows that the average performance of random-replacement caches is acceptable. Motivated by this initial study,

several other solutions were proposed for fully time-randomized cache designs, deploying random replacement *and* random placement.

Hash-based Parametric Random Placement [64] deploys a parametric hash function that combines the requested memory address and a random number. That function delivers the index bits used to determine the set to which the accessed address is mapped. The hash function combines all the address bits, except for the cache-line offset bits, via a set of rotator blocks and XOR gates, and ensures that all address to set mappings have equal probability. Every time the seed is changed (e.g., when a task starts or ends), the cache contents are flushed. Accordingly, each (random) placement of addresses is valid for a whole execution of the program of interest: the addresses mapped to the same set compete for space during the whole run. With hash-based random placement, all addresses are mapped to sets independently. Hence, addresses that would otherwise perfectly coexist in a given cache way, have a non-zero probability to map to the same set, which increases the miss rate and negatively affects execution time. [16] presents the first FPGA implementation and evaluation of the cache design proposed in [64], using the Mersenne Twister as random generator instead of the Multiply With Carry, for its better statistical properties. The evaluation considers the propagation delay and the area required on that FPGA board.

Random Modulo [59] keeps the locality advantages of modulo placement but breaks the dependence between memory location and cache placement. To that end, Random Modulo prevents conflicts between cache lines that map to the same cache way: by randomly permuting the random seed with the address tag bits, two addresses with identical tag bits and different index bits are necessarily mapped to distinct cache sets. Hence, two addresses that with modulo placement would belong in the same cache way but different cache sets, are also prevented from mapping to the same set with Random Modulo. This solution avoids pessimistic scenarios where several addresses in the same cache way map to the same set. Similarly to [16], [59] presents an FPGA implementation of the proposed Random Modulo design and synthesizes it with an ASIC cell library. An evaluation based on the same criteria used in [64] shows how the performance of Random Modulo exceeds that of hash-based parametric random placement.

Multi-level time-randomized caches have also been studied. The first attempt to analyse them [65] considers hash random placement and shows how the events in a multilevel cache have a probabilistic nature, hence fitting the premises of MBPTA. The authors analyse several L1-L2 inclusion policies (inclusive caches, exclusive caches, and non-inclusion control) and write-miss policies (write through and write back).

[106] studies hash random placement for non-partitioned last-level caches. Cache partitioning is the preferred solution for shared caches, as it protects the data updates of one core from the risk of eviction by another core, thereby enabling per-core cache analysis. The cited work shows that the probabilistic behaviour of random caches allows upper-bounding the impact that a task running on one core may have on tasks assigned to other cores. This result is achieved by controlling the frequency at which tasks are allowed to perform evictions on shared caches, thus sparing the need to partition.

*6.2.2 Arbitration Policies.* Shared hardware resources normally comprise an arbiter that orchestrates concurrent accesses as a means to prevent or regulate conflicts. In the scope of real-time systems, the arbiter must implement an assignment policy that yields a known upper-bound to the longest suffered delay. Where classic analysis seeks a deterministic single-valued bound (the longest possible duration), MBPTA can use both deterministic and probabilistic bounds. While a deterministic bound asserts that the probability that a request is delayed longer than the stipulated amount is zero, probabilistic bounds provide a set of values, each attached to an exceedance

probability. To facilitate the latter, arbitration policies are meant to incorporate some form of randomization in the grant assignment process.

The most basic randomized policy is known as *lottery* [74] where, at every (slot) round of duration $L$, the grant is assigned randomly to one of $N_r$ requestors. Hence, the probability for a requestor to be granted access in the first round is $(1/N_r)^1 \times ((N_r - 1)/N_r)^{1-1} = 1/N_r$. Likewise, the probability of being granted access in the second round is $(1/N_r)^1 \times ((N_r - 1)/N_r)^{(2-1)}$, which becomes $(1/N_r)^1 \times ((N_r - 1)/N_r)^{(3-1)}$ in the third round, and so forth. Lottery is an MBPTA-supportive policy thanks to the probabilistic nature of the guarantees that it offers of granting access in a specific number of cycles, and to the fact that this assurance holds at analysis and at operation. It, however, suffers the risk that the probability of not being granted access is never null (albeit very small) after several rounds of arbitration.

*Random permutations* for buses [63] and tree networks-on-chip [105] is an alternative randomized policy that is also MBPTA-supportive. Unlike lottery, it guarantees that the probability of a request not being granted access in a given number of arbitration slots does fall to zero. This is assured by the use of an arbitration window of $N_r$ slots, which contains the permutations of the IDs of all requestors. The order in which access is granted follows the permutation window, which changes randomly every time it is exhausted.

Other policies that allow providing a deterministic bound to the longest wait time for a request owing to arbitration have been shown to be MBPTA-supportive. For example, this is the case for round-robin [63] and TDMA [94]. The cited work shows that random arbitration solutions outperform deterministic alternatives either in terms of pWCET estimates or average performance.

*6.2.3 Pseudo-Random Number Generation.* Pseudo-random number generators provide the random bits needed by random placement, replacement, and arbitration policies [10]. While the sequence of bits that they generate is not truly random, the cited work presents an implementation that provides an output long enough to prevent repetitions for durations commensurate with task run periods, thus preventing potential correlation of events.

### 6.3 MBPTA-Supportive Software Solutions

While HWRand platforms support MBPTA with low-complexity designs, some of which have been implemented on FPGAs, it will take some time before they can enter the processor market for good. Software support for MBPTA hence aims at providing a short-term solution to apply MBPTA to COTS processors. To date, the works in this area have concentrated on COTS caches.

All the proposed software randomization solutions leverage the fixed relation between the main memory position and cache placement in deterministic systems, as well as between the binary and main memory layout. In addition, all those solutions use random padding to displace programs' memory objects, and also reorder them across program runs or program compilations. The various software randomisation solutions differ in the way they employ randomisation, their intrusiveness on the compilation tool chain, and their friendliness to the certification process.

Dynamic Software Randomisation [69] (DSR) uses self-modifying code during program initialization (hence, at every program run), to randomly allocate code and global data, including the stack frame, whose size is decided at runtime. The DSR runtime has been ported to PowerPC [69] and SPARCv8 [33]. That work improves the original implementation by reducing its memory footprint and bounding the execution-time effect of its runtime operation. DSR has been tested with avionics [111] and aerospace case study applications [33], which have increased its TRL.

However, the DSR self-modifying code does conflict with automotive practices and technologies that allocate read-only code and data to flash memories. Moreover, the use of pointers and dynamic objects is restricted by safety standards, for example, ISO26262 [62]. Static Software

Randomisation [70] (SSR) addresses those issues. Unlike DSR, SSR operates in a static manner by performing a random reallocation of object across distinct binaries of the same program. Selecting one executable from the set of binaries generated by SSR allows reasoning in probabilistic terms about the coverage of cache layouts. The static nature of SSR makes the compiled software more amenable to certification. SSR has been verified on top of real hardware platforms like AURIX [68].

## 7 TIMING ANALYSIS IN THE PRESENCE OF FAULTS

### 7.1 Introduction

Various authors have studied the impact of (hardware) faults on program's execution time, leading to probabilistic WCET estimates.

[57] analyses the impact of random and independent permanent faults disabling lines in modulo-placement and LRU-replacement instruction caches, using traditional static timing analysis (STA). The cited work studies the faulty cache maps that may result from a given failure rate in SRAM bit cells, and the degradation that this may cause to WCET estimates, associating the latter to the probability of occurrence of each cache map. This work shows that, for specific failure rates in SRAM bit cells, faults may occur in locations that affect execution time (hence, the program's WCET) with a probability sufficiently high to deserve attention. The probabilistic WCET estimates that result from this analysis therefore relate specifically to the pathology of faulty maps. Hardy et al. [58] extend their earlier work, proposing hardware techniques to mitigate the impact of permanent faults in instruction caches. They show that the cache sets whose lines are all faulty are the major contributors to WCET degradation, and they propose mitigation techniques that use either a hardened cache way, so that each cache set has at least one fault-free line, or a 1-entry shared buffer for cache sets with all lines faulty. Their results show that those solutions trade off differently, for hardware cost and WCET improvement, against non-hardened instruction caches.

Other works focus on time-randomized caches. Chen et al. [31] consider fully associative caches with random replacement policies[3] and model cache behaviour with Markov chains, where each state corresponds to a different set of cache contents. With exponential cost, such Markov chains allow modelling precisely and statically the probabilistic execution time of the analysed programs, thus yielding costly but highly accurate pWCET outcomes. This form of SPTA is further optimized by limiting the set of addresses considered, thus trading accuracy for speed of computation. Building on this Markov model, the authors consider the impact of transient and permanent faults on the pWCET estimates. Transient faults change the current state in the Markov model—with a probability related to the failure rate of a SRAM bit cell—by removing the contents of one cache line. That transition happens on trigger from hardware fault detection mechanisms such as parity checks. For permanent faults instead, the work describes the program's timing behaviour with $N + 1$ Markov models, where $N$ is the number of cache lines. Each model corresponds to a number $n \in 0\,..\,N$ of faulty cache lines. Upon a permanent fault, the content of the affected cache line is marked erroneous and the system transitions from its current Markov model with $N - i$ fault-free lines, to the model with $N - i - 1$ ones.

[30] extends this line of work to account for the delay occurring between fault detection and the diagnosis of its permanent nature. The cited authors acknowledge that transient and permanent faults yield different latencies, and propose considering a fault as transient until diagnosed otherwise. They further show that, if permanent faults occur sufficiently often, as could occur in harsh environment conditions (e.g., space), the corresponding latencies need to be accounted for, which requires quick diagnostic mechanisms, to contain the inflation of pWCET estimates.

[3]The cited authors note that their work also applies to set-associative caches with deterministic placement, by studying each cache set separately.

Slijepcevic et al. [107] study the impact of permanent and transient (hardware) faults, including the overhead of fault detection, correction, classification, and recovery, in the context of MBPTA. The cited authors assume processors that embed time-randomized caches, with random placement and replacement policies, and show that the random nature of fault location, paired to random placement lead to simple fault models. While time-deterministic caches require considering all potential fault maps (whose cardinality grows exponentially with the number of faults), time-randomized caches only require considering the maximum fault count, regardless of the fault location. Moreover, the probabilistic nature of the fault-free platform (with time-randomized behaviour) and the fault occurrence allow considering permanent faults in all cache memories at once, independently of the complexity of the cache hierarchy. The cited work supports its claims on a multi-level cache hierarchy, with a unified L2 cache for instructions and data, and TLBs. Experimental evidence shows that pWCET estimates degrade slowly as the fault count increases, since pathological alignments of faults and address placement occur with decreasing probabilities. The authors also provide hints on how to account for faults in components other than cache memories as part of the pWCET estimates, by decreasing the operating frequency when needed [107].

Höfig [60] studies the impact of faulty sensors in WCET estimates. The cited work analyses the impact that such faults may have on program's execution time, and proposes a failure-dependent timing analysis that considers fault handling as determined by the applicable safety requirements. The authors assume that worst-case response time bounds are known for all system components as well as fault probabilities for all sensors. At that point, the safety mechanisms in place and their associated execution-time cost are characterised as probabilistic additions to the known WCET, thus leading to pWCET estimates. Interestingly, this approach can be combined opportunistically with any technique that delivers any type of WCET estimate, whether probabilistic or not.

## 8 MISCELLANEA

### 8.1 Introduction

A few other significant lines of work in the state of the art on PTA escape the taxonomy presented in Section 2. We discuss them here under three headings: case studies, which determine the viability and performance of PTA solutions in application scenarios; comparative assessments, which study how PTA fares with respect to traditional STA; and argumentative analyses, which assess the fitness for use of PTA in certification-regulated domains.

### 8.2 Case Studies

The works in this category use either real processor boards with real-world applications, or abstract hardware models or architectural simulators with benchmark suites [54, 98]. All PTA case studies employ MBPTA: to the best of our knowledge, in fact, no evaluation of SPTA has been performed to date in any of the above settings.

[68] uses an Automotive Cruise Control System (ACCS), automatically generated from a Simulink model and targeted to the AURIX TC277 processor, to show that static software randomization (cf. Section 6.3) enables MBPTA to capture the effect of cache jitter on the pWCET.

[111] evaluates the use of dynamic software randomization (cf. Section 6.3) with MBPTA for single-core and multicore variants of an MBPTA-supportive processor simulator [63, 69] running two avionics applications (one for acquisition and maintenance of flight-control data, the other for estimating the center of gravity position of the aircraft). [33] targets an FPGA platform based on LEON3 with a mixed-criticality space application, controlling an active optics instrument for space telescopes, to evaluate the performance of DSR (cf. Section 6.3) implemented within the LLVM compiler and integrated with an industrial-quality RTOS. The cited work shows that DSR

enables MBPTA at the cost of a modest performance penalty. Interestingly, the pWCET estimates computed in those experiments are tighter than the typical 20% margin considered in current industrial practice over the high water mark execution time.

Fernandez et al. [48] use a LEON3 made MBPTA-supportive, with the FPU forced to work at its highest latency, and deploying time-randomized caches and TLBs, as presented in [59], to run a Thrust Vector Control Application developed by the European Space Agency. The cited work shows how the modified hardware allowed MBPTA to derive sound pWCET estimates.

Lima and Bates in [78] apply EVT to measurement observations collected on a Rolls-Royce Digital Engine Controller application, using the single-bucket approach discussed in Section 4.2.1. The cited work presents the Indirect Estimation in Statistical Time Analysis (IESTA) technique, which the authors propose as an alternative to time-randomization applied at hardware or software level, to enable the use of EVT with otherwise overly discrete or poorly analysable datasets. IESTA uses an artificial random variable to inject timing variability into the observation measurements collected during the analysis, before applying EVT. The authors report that IESTA allowed them to derive statistically sound models of the application's timing behaviour.

### 8.3 Comparisons

Making a fair and sound comparison of different timing analysis techniques is a hard challenge as individual techniques often build upon very different (and possibly even antagonistic) assumptions. Yet, some works made some inroad in that direction.

[6] compares MBPTA, STA, and SPTA on a simple processor setup where instructions have fixed latency except for the jitter caused by the instruction cache, which is deterministic for STA and time-randomized for SPTA and MBPTA. The quantitative results (a qualitative comparison is also performed) of that study show that STA performs better when the program fits in cache whereas MBPTA gets better results when conflict misses arise.

[7] compares the requirements set by STA, SPTA, and MBPTA on the procedure needed to obtain high-confidence WCET estimates for multicore processors. The cited work ranks those techniques according to the affordability of satisfying their use requirements in industrial-use scenarios.

### 8.4 Certification

To support the penetration of MBPTA in the industrial practice of application domains subject to rigorous qualification or certification, some reasoned argument ought to be constructed to explain why the method is suitable. Interestingly, however, the question of how to construct an explicit argument for the WCET problem in general lacks an established literature. Implicit argumentations are more frequent, often in the form of underlying assumptions and associated context vocabulary.

Industrial practice is normally very specific in the validation program that development and verification methods alike have to undergo before being deemed fit for use. In the face of standard prescriptions, fitness for use requires compliance. The CAST guidelines [27] aim to help the certification authorities in the avionics domain to understand what is being proposed as *alternative means of compliance*: the objectives met, the rationale for using the alternative means, its engineering and safety adequacy, without prescribing a way to generate or document this information. A plausible argument pattern would map its objectives to those of the applicable standard, showing that the method is trustworthy as well as feasible to implement (for usability, automation, or user competency). To this very end, [108] constructs a model certification argument to instantiate for specific validation programs, which follows an explicit argumentation approach that seeks maximum clarity in its claims, shows where probabilities and confidence apply, and suggests how to transfer the argument between domains.

Another angle of interest in this regard arises with mixed-criticality systems, which integrate applications at different safety assurance levels, into multicore embedded platforms, in the hope of reaping performance, cost, size, weight, and power benefits. When safety assurance is at stake, the pursuit of determinism at all levels of execution behaviour is often perceived as a major factor of simplification in the process of gathering the prescribed certification evidence [1]. Yet, that determinism is compromised in most modern COTS multicore platforms, where the quest for higher average performance is a far greater concern than determinism. [11] shows that the use of MBPTA may fit, for compliance to domain regulations and resource efficacy, in the definition of a safety concept for a mixed-criticality multicore system equipped with hardware protection mechanisms for use in the automotive domain. Regarding the compliance concern, the authors report the positive independent assessment of that safety concept by an independent certification body. For the efficacy concern, the cited work shows how MBPTA helps mitigate the untenable pessimism that may arise from the use of traditional WCET analysis or the (potentially unsound) addition of conservative margins to the high water mark values obtained by observation.

[11]'s safety concept sees MBPTA as a competitive solution to prevent resource overprovisioning and ensure independence among mixed-criticality applications. The pseudo-random number generator plays a central role in processors that support MBPTA. One interesting ramification of the certification-enabling works cited in this section was to assess whether a design exists for such a device that meets the safety requirements in IEC-61508 for no less than SIL 3, and warrants seamless integration in a real-world multicore processor. This concern was addressed in [10].

## 9 CONCLUSIONS AND OUTLOOK

### 9.1 Status

The unprecedented rise in guaranteed performance needs in critical real-time embedded systems promotes the adoption of high-performance processors that feature deep cache hierarchies and multicore execution even in otherwise conservative industrial domains. While transitioning to complex hardware platforms has been acknowledged as the only cost-effective manner to meet the emerging performance requirements, it also pushes existing timing analysis techniques to their limits, for complexity and effectiveness. In this context, probabilistic/statistical timing analysis methods have recently emerged as a promising paradigm to overcome the limitations of standard deterministic timing analysis approaches and to deliver sustainable, reliable, and industrial-quality WCET estimates. The rapid rise of interest in those techniques has motivated the production of a vast body of literature in the state of the art. Unfortunately, however, those works often have a very diverse range of goals and assumptions that are difficult to tell apart, thus impeding a clear understanding of the proposed problem-and-solution landscape.

This survey originates from the need to fill that gap, with particular focus on the worst-case timing analysis of critical real-time systems. To this end, this work covers the state of the art of probabilistic timing analyses, from their theoretical basis, to their evaluation and assessment against industrial applications. The result shows that several, substantially different approaches were previously collected under the indistinct umbrella of probabilistic timing analysis.

This work shows that much progress has been made towards the consolidation of PTA, not only in foundational terms, but also to make it viable for industrial use and to quantitatively and qualitatively assess the performance of its solutions.

### 9.2 PTA Limitations and Research Directions

To conclude this survey, we single out directions of future work on PTA which should help this research area to further its maturation and extend its reach into broader acknowledgment and use.

*Advancing SPTA.* Whereas the static and dynamic (measurement-based) families of PTA techniques were born together and were seen as equally disruptive, the largest proportion of the subsequent research efforts has concentrated on the measurement-based variant.

As a consequence, SPTA does not seem to have reached yet the level of maturity required to warrant low complexity of application and viable use prerequisites. In that respect, the research on SPTA has large margins for improvement. In order to maintain its attractiveness, it is desirable that further research efforts be directed to improve the computational complexity of SPTA and to extend it to multi-level cache hierarchies and more realistic processor architectures.

*Measurement-Based Methods, Platform Requirements, and Representativeness.* In contrast with SPTA, measurement-based methods have been vastly investigated in the last few years, leading to the publication of diverse approaches with different assumptions, procedures, and degrees of formalization, using statistical tools from Extreme Value Theory as a common trait of most of them.

A clear distinction among MBPTA approaches derives from the assumptions that they make on the hardware or software of the execution platform. Employing MBPTA (with EVT) on deterministic platforms, while not necessarily antagonistic to the EVT hypotheses, challenges the provision of confirmatory arguments, which afflicts measurement-based analysis domain for representativeness with respect to the coverage attained of program paths and execution conditions. MBPTA-supportive platforms address this problem specifically, in the form of real hardware implementation or software-level solutions, which allow the user to reason on representativeness in probabilistic terms. However, most existing platforms lack adequate hardware support, while software-level support has been shown to pose additional use challenges: these limitations prevent applying those techniques to several current platforms and applications.

The applicability of EVT per se has also been questioned as it may not be possible to derive satisfactory results on all classes of software programs. How to intercept those situations and whether there exists an alternative approach to circumvent them thus are interesting research questions to further investigate.

*Industrialization.* Insisting on industrialization is a necessary driver for the consolidation of PTA as the industrial concerns with the timing analysis problem were the origin to probabilistic approaches in the first place. Meeting the certification requirements of the real-time critical domains is a mandatory prerequisite in this respect. This line of work has been already initiated for certifiable railway applications and needs to be further extended to other industrial domains.

## REFERENCES

[1] 2013. Advanced Cockpit for Reduction of Stress and Workload. Retrieved on May, 2017 from http://www.across-fp7.eu/.

[2] 2016. QUALCOMM Snapdragon 820 Automotive Processor. Retrieved on May, 2017 from https://www.qualcomm.com/products/snapdragon/processors/820-automotive.

[3] 2016. RENESAS R-Car H3. Retrieved on May, 2017 from https://www.renesas.com/en-us/solutions/automotive/products/rcar-h3.html.

[4] 2017. NVIDIA DRIVE PX. Scalable Supercomputer for Autonomous Driving. Retrieved from http://www.nvidia.com/object/drive-px.html.

[5] H. Abarbanel, S. Koonin, H. Levine, G. MacDonald, and O. Rothaus. 1992. *Statistics of Extreme Events with Application to Climate.* MITRE CORP MCLEAN VA JASON PROGRAM OFFICE. http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA247342.

[6] J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla. 2014. On the comparison of deterministic and probabilistic WCET estimation techniques. In *Euromicro Conference on Real-Time Systems (ECRTS'14).*

[7] Jaume Abella, Carles Hernández, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Perez, Enrico Mezzetti, and Tullio Vardanega. 2015. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES'15).*

[8] J. Abella, M. Padilla, J. Del Castillo, and F. J. Cazorla. 2017. Measurement-based worst-case execution time estimation using the coefficient of variation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 2, 4 (2017).
[9] J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla. 2014. Heart of gold: Making the improbable happen to increase confidence in MBPTA. In *Euromicro Conference on Real-Time Systems (ECRTS'14)*.
[10] I. Agirre, M. Azkarate-askasua, C. Hernández, J. Abella, J. Perez, T. Vardanega, and F. J. Cazorla. 2015. IEC-61508 SIL 3 compliant pseudo-random number generators for probabilistic timing analysis. In *2015 Euromicro Conference on Digital System Design (DSD'15)*.
[11] I. Agirre, M. Azkarate-askasua, A. Larrucea, J. Perez, T. Vardanega, and F. J. Cazorla. 2016. Automotive safety concept definition for mixed-criticality integration on a COTS multicore. In *Computer Safety, Reliability, and Security—SAFECOMP Workshops, ASSURE, DECSoS, SASSUR, and TIPS*.
[12] B. Akesson, A. Hansson, and K. Goossens. 2009. Composable resource sharing based on latency-rate servers. In *12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools (DSD'11)*.
[13] M. A. Alam, K. Roy, and C. Augustine. 2011. Reliability and process-variation aware design of integrated circuits. In *Reliability Physics Symposium (IRPS'11))*.
[14] S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. 2015. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems* 51, 1 (2015).
[15] Sebastian Altmeyer and Robert I. Davis. 2014. On the correctness, optimality and precision of static probabilistic timing analysis. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*.
[16] H. Anwar, C. Chen, and G. Beltrame. 2015. A probabilistically analysable cache implementation on FPGA. In *International New Circuits and Systems Conference (NEWCAS'15)*.
[17] ARM. 2015. *ARM Expects Vehicle Compute Performance to Increase 100x in Next Decade*. Technical Report. ARM. https://www.arm.com/about/newsroom/arm-expects-vehicle-compute-performance-to-increase-100x-in-next-decade.php.
[18] P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella, and F. J. Cazorla. 2016. A confidence assessment of WCET estimates for software time randomized caches. In *International Conference on Industrial Informatics (INDIN'16)*.
[19] P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella, and F. J. Cazorla. 2016. Modelling the confidence of timing analysis for time randomised caches. In *Symposium on Industrial Embedded Systems (SIES'16)*.
[20] G. Benktander and C. O. Segerdahl. 1960. On the analytical representation of claim distributions with special reference to excess of loss reinsurance. In *XVIth International Congress of Actuaries*.
[21] K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar. 2014. WCET measurement-based and extreme value theory characterisation of CUDA kernels. In *Conference on Real-Time Networks and Systems (RTNS'14)*.
[22] G. Bernat, A. Colin, and S. M. Petters. 2002. WCET analysis of probabilistic hard real-time system. In *Real-Time Systems Symposium (RTSS'02)*.
[23] F. Bodin and A. Seznec. 1997. Skewed associativity improves program performance and enhances predictability. *IEEE Transactions on Computers* 46, 5 (1997).
[24] A. Burns, G. Bernat, and I. Broster. 2003. A probabilistic framework for schedulability analysis. In *Embedded Software*, Rajeev Alur and Insup Lee (Eds.). Springer, Berlin, Berlin, 1–15.
[25] F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella. 2013. Upper-bounding program execution time with extreme value theory. In *13th International Workshop on Worst-Case Execution Time Analysis (WCET'13)*.
[26] F. J. Cazorla, et al. 2013. PROARTIS: Probabilistically analyzable real-time systems. *ACM Transactions on Embedded Computing Systems* 12, 2s (2013).
[27] Certification Authorities Software Team. 2016. *CAST-32A Multi-core Processors*.
[28] J. M. Chambers. 1983. *Graphical Methods for Data Analysis*. Wadsworth, Belmont, CA.
[29] Robert N. Charette. 2009. This car runs on code. In *IEEE Spectrum (online)*. http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.
[30] C. Chen, J. Panerati, and G. Beltrame. 2016. Effects of online fault detection mechanisms on probabilistic timing analysis. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT'16)*.
[31] C. Chen, L. Santinelli, J. Hugues, and G. Beltrame. 2016. Static probabilistic timing analysis in presence of faults. In *Symposium on Industrial Embedded Systems (SIES'16)*.
[32] S. Coles. 2001. *An Introduction to Statistical Modelling of Extreme Values*. Springer.
[33] F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla. 2017. Dynamic software randomisation: Lessons learned from an aerospace case study. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*.
[34] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. 2012. Measurement-based probabilistic timing analysis for multi-path programs. In *Euromicro Conference on Real-Time Systems (ECRTS'12)*.

[35] L. David and I. Puaut. 2004. Static determination of probabilistic execution times. In *Euromicro Conference on Real-Time Systems (ECRTS'04)*.
[36] Robert I. Davis, Luca Santinelli, Sebastian Altmeyer, Claire Maiza, and Liliana Cucu-Grosjean. 2013. Analysis of probabilistic cache related pre-emption delays. In *Euromicro Conference on Real-Time Systems (ECRTS'13)*.
[37] J. T. de Oliveira. 1984. Statistical extremes and applications. In *NATO ASI Series, Mathematical and Physical Sciences, Vol. 131*. Springer.
[38] J. del Castillo and M. Padilla. 2016. Modelling extreme values by the residual coefficient of variation. *Statistics and Operations Research Transactions (SORT)* 40, 2 (2016), 303–320. ISSN 1696-2281.
[39] F. Despaux, Y. Song, and A. Lahmadi. 2015. Extracting Markov chain models from protocol execution traces for end to end delay evaluation in wireless sensor networks. In *World Conference on Factory Communication Systems (WFCS'15)*.
[40] E. Diaz, M. Fernandez, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and F. J. Cazorla. 2017. MC2: Multicore and cache analysis via deterministic and probabilistic jitter bounding. In *Ada Europe, 22nd International Conference on Reliable Software Technologies*.
[41] J. Diebolt, M. Garrido, and S. Girard. 2007. A goodness-of-fit test for the distribution tail. In *Topics in Extreme Values, (Chapter 5)*, M. Ahsanullah and S. N. U. A. Kirmani (Eds.). Nova Science, New York.
[42] M. Duranton, K. De Bosschere, C. Gamrat, J. Maebe, H. Munk, and O. Zendra. 2017. *HiPEAC Vision 2017*.
[43] Daniel L. Dvorak. 2009. *NASA Study on Flight Software Complexity. Final Report*. Technical Report. NASA.
[44] S. Edgar and A. Burns. 2001. Statistical analysis of WCET for scheduling. In *Real-Time Systems Symposium (RTSS'01)*.
[45] P. Embrechts, T. Mikosch, and C. Klüppelberg. 1997. *Modelling Extremal Events: For Insurance and Finance*. Springer-Verlag, London, UK.
[46] E. S. Epstein. 1969. A scoring system for probability forecasts of ranked categories. *Journal of Applied Meteorology* 8, 6 (1969).
[47] Rolf Ernst and Marco Di Natale. 2016. Mixed criticality systems—A history of misconceptions? *IEEE Design & Test* 33, 5 (2016), 65–74.
[48] M. Fernández, D. Morales, L. Kosmidis, A. Bardizbanyan, I. Broster, C. Hernández, E. Quiñones, J. Abella, F. J. Cazorla, P. Machado, and L. Fossati. 2017. Probabilistic timing analysis on time-randomized platforms for the space domain. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*.
[49] R. A. Fisher and L. H. Tippett. 1928. Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Mathematical Proceedings of the Cambridge Philosophical Society* 24, 2 (1928).
[50] Patrick J. Graydon and Iain Bate. 2014. Realistic safety cases for the timing of systems. *The Computer Journal* 57, 5 (2014).
[51] D. Griffin and A. Burns. 2010. Realism in statistical analysis of worst case execution times. In *10th International Workshop on Worst-Case Execution Time Analysis (WCET'10)*.
[52] D. Griffin, B. Lesage, A. Burns, and R. I. Davis. 2014. Static probabilistic timing analysis of random replacement caches using lossy compression. In *International Conference on Real-Time Networks and Systems (RTNS'14)*.
[53] F. Guet, L. Santinelli, and J. Morio. 2016. On the reliability of the probabilistic worst-case execution time estimates. In *8th European Congress on Embedded Real Time Software and Systems (ERTS'16)*.
[54] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. 2010. The Mälardalen WCET benchmarks—Past, present and future. In *10th International Workshop on Worst-Case Execution Time Analysis*.
[55] J. P. Hansen, S. A. Hissam, and G. A. Moreno. 2009. Statistical-based WCET estimation and validation. In *Workshop on Worst-Case Execution Time (WCET) Analysis*.
[56] D. Hardy and I. Puaut. 2008. WCET analysis of multi-level non-inclusive set-associative instruction caches. In *The 29th IEEE Real-Time Systems Symposium (RTSS'08)*.
[57] D. Hardy and I. Puaut. 2015. Static probabilistic worst case execution time estimation for architectures with faulty instruction caches. *Real-Time Systems* 51, 2 (2015).
[58] D. Hardy, I. Puaut, and Y. Sazeides. 2016. Probabilistic WCET estimation in presence of hardware for mitigating the impact of permanent faults. In *Design, Automation Test in Europe Conference Exhibition (DATE'16)*.
[59] Carles Hernández, Jaume Abella, Andrea Gianarro, Jan Andersson, and Francisco J. Cazorla. 2016. Random modulo: A new processor cache design for real-time critical systems. In *Design Automation Conference (DAC'16)*.
[60] K. Höfig. 2012. *Failure-Dependent Timing Analysis—A New Methodology for Probabilistic Worst-Case Execution Time Analysis*. Springer, Berlin, 61–75.
[61] Intel. 2016. Intel GO Automated Driving Solution Product Brief. Retrieved on May, 2017 from https://www.intel.es/content/dam/www/public/us/en/documents/platform-briefs/go-automated-accelerated-product-brief.pdf.
[62] International Organization for Standardization. 2009. *ISO/DIS 26262. Road Vehicles—Functional Safety*.
[63] Javier Jalle, Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, and Francisco J. Cazorla. 2014. Bus designs for time-probabilistic multicore processors. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'14)*.

[64] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. 2013. A cache design for probabilistically analysable real-time systems. In *Design, Automation and Test in Europe (DATE'13)*.
[65] Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, and Francisco J. Cazorla. 2013. Multi-level unified caches for probabilistically time analysable real-time systems. In *Real-Time Systems Symposium (RTSS'13)*.
[66] L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. 2014. Efficient cache designs for probabilistically analysable real-time systems. *IEEE Transactions on Computers* 63, 12.
[67] L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla. 2014. PUB: Path upper-bounding for measurement-based probabilistic timing analysis. In *Euromicro Conference on Real-Time Systems (ECRTS'14)*.
[68] L. Kosmidis, D. Compagnin, D. Morales, E. Mezzetti, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. 2016. Measurement-based timing analysis of the AURIX caches. In *16th International Workshop on Worst-Case Execution Time Analysis (WCET'16)*.
[69] L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. D. Berger, and F. J. Cazorla. 2013. Probabilistic timing analysis on conventional cache designs. In *Design, Automation and Test in Europe (DATE'13)*.
[70] L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla. 2014. Containing timing-related certification cost in automotive systems deploying complex hardware. In *Design Automation Conference (DAC'14)*.
[71] L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla. 2014. Measurement-based probabilistic timing analysis and its impact on processor architecture. In *Euromicro Conference on Digital System Design (DSD'14)*.
[72] Leonidas Kosmidis, Eduardo Quiñones, Jaume Abella, Tullio Vardanega, and Francisco J. Cazorla. 2013. Achieving timing composability with measurement-based probabilistic timing analysis. In *ISORC*. IEEE Computer Society.
[73] S. Kotz and S. Nadarajah. 2000. *Extreme Value Distributions: Theory and Applications*. World Scientific.
[74] Kanishka Lahiri, Anand Raghunathan, and Ganesh Lakshminarayana. 2006. The LOTTERYBUS on-chip communication architecture. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 6.
[75] Stephen Law and Iain Bate. 2016. Achieving appropriate test coverage for reliable measurement-based timing analysis. In *Euromicro Conference on Real-Time Systems (ECRTS'16)*.
[76] Benjamin Lesage, David Griffin, Sebastian Altmeyer, and Robert I. Davis. 2015. Static probabilistic timing analysis for multi-path programs. In *2015 IEEE Real-Time Systems Symposium (RTSS'15)*.
[77] B. Lesage, D. Griffin, F. Soboczenski, I. Bate, and R. I. Davis. 2015. A framework for the evaluation of measurement-based timing analyses. In *Real Time and Networks Systems (RTNS'15)*.
[78] G. Lima and I. Bate. 2017. Valid application of EVT in timing analysis by randomising execution time measurements. In *Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*.
[79] G. Lima, D. Dias, and E. Barros. 2016. Extreme value theory for estimating task execution time bounds: A careful look. In *Euromicro Conference on Real-Time Systems (ECRTS'16)*.
[80] José María López, José Luis Díaz, Joaquín Entrialgo, and Daniel García. 2008. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Systems* 40, 2 (2008).
[81] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. 2011. A new way about using statistical analysis of worst-case execution times. *SIGBED Review* 8, 3 (2011).
[82] Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. 2012. A statistical response-time analysis of real-time embedded systems. In *33rd Real-Time Systems Symposium*.
[83] C. Maxim, A. Gogonel, I. Asavoae, M. Asavoae, L. Cucu-Grosjean, and W. Talaboulma. 2016. Reproducibility and representativity—Mandatory properties for the compositionality of measurement-based WCET estimation approaches. In *9th International Workshop on Compositional Theory and Technology for Real-Time Embedded System (CRTS'16)*.
[84] Dorin Maxim, Mike Houston, Luca Santinelli, Guillem Bernat, Robert I. Davis, and Liliana Cucu-Grosjean. 2012. Re-sampling for statistical timing analysis of real-time systems. In *Real-Time and Network Systems (RTNS'12)*.
[85] A. Melani, E. Noulard, and L. Santinelli. 2013. Learning from probabilities: Dependences within real-time systems. In *Emerging Technologies Factory Automation (ETFA'13)*.
[86] E. Mezzetti, N. Holsti, A. Colin, G. Bernat, and T. Vardanega. 2008. Attacking the sources of unpredictability in the instruction cache behavior. In *Real-Time and Network Systems (RTNS'08)*.
[87] Enrico Mezzetti and Tullio Vardanega. 2013. A rapid cache-aware procedure positioning optimization to favor incremental development. In *Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*.
[88] Enrico Mezzetti, Marco Ziccardi, Tullio Vardanega, Jaume Abella, Eduardo Quiñones, and Francisco J. Cazorla. 2015. Randomized caches can be pretty useful to hard real-time systems. *Leibniz Transactions on Embedded Systems* 2, 1 (2015).
[89] S. Milutinovic, J. Abella, I. Agirre, M. Azkarate-Askasua, E. Mezzetti, T. Vardanega, and F. J. Cazorla. 2017. Software time reliability in the presence of cache memories. In *Ada Europe, 22nd International Conference on Reliable Software Technologies*.

[90] S. Milutinovic, J. Abella, and F. J. Cazorla. 2016. Modelling probabilistic cache representativeness in the presence of arbitrary access patterns. In *19th IEEE International Symposium on Real-Time Distributed Computing (ISORC'16)*.
[91] Suzana Milutinovic, Jaume Abella, Damien Hardy, Eduardo Quiñones, Isabelle Puaut, and Francisco J. Cazorla. 2015. Speeding up static probabilistic timing analysis. In *Architecture of Computing Systems - ARCS*.
[92] S. Milutinovic, E. Mezzetti, J. Abella, T. Vardanega, and F. J. Cazorla. 2017. On uses of extreme value theory fit for industrial-quality WCET analysis. In *Symposium on Industrial Embedded Systems (SIES'17)*.
[93] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt. 2014. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *Euromicro Conference on Real-Time Systems*.
[94] M. Panic, J. Abella, C. Hernández, E. Quiñones, T. Ungerer, and F. J. Cazorla. 2015. Enabling TDMA arbitration in the context of MBPTA. In *Euromicro Conference on Digital System Design (DSD'15)*.
[95] Marco Paolieri, Eduardo Quiñones, Francisco J. Cazorla, Guillem Bernat, and Mateo Valero. 2009. Hardware support for WCET analysis of hard real-time multicore systems. In *International Symposium on Computer Architecture (ISCA'09)*.
[96] K. Pearson. 1900. X. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine* 50, 302 (1900), 157–175. DOI: https://doi.org/10.1080/14786440009463897
[97] J. Pickands. 1975. Statistical inference using extreme order statistics. *The Annals of Statistics* 3, 1 (1975), 119–131.
[98] Jason Poovey, et al. 2007. *Characterization of the EEMBC Benchmark Suite*. North Carolina State University.
[99] E. Quiñones, E. D. Berger, G. Bernat, and F. J. Cazorla. 2009. Using randomized caches in probabilistic real-time systems. In *Euromicro Conference on Real-Time Systems (ECRTS'09)*.
[100] Khaled S. Refaat and Pierre-Emmanuel Hladik. 2010. Efficient stochastic analysis of real-time systems via random sampling. In *Euromicro Conference on Real-Time Systems (ECRTS'10)*.
[101] Jan Reineke. 2014. Randomized caches considered harmful in hard real-time systems. *Leibniz Transactions on Embedded Systems* 1, 1 (2014).
[102] L. Santinelli, F. Guet, and J. Morio. 2017. Revising measurement-based probabilistic timing analysis. In *Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*.
[103] L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. 2014. On the sustainability of the extreme value theory for WCET estimation. In *14th International Workshop on Worst-Case Execution Time Analysis*, Vol. 39.
[104] M. Schlansker, R. Shaw, and S. Sivaramakrishnan. 1993. *Randomization and Associativity in the Design of Placement-insensitive Caches*. Technical Report. HP Report, HPL-93-41.
[105] M. Slijepcevic, M. Fernández, C. Hernández, J. Abella, E. Quiñones, and F. J. Cazorla. 2016. pTNoC: Probabilistically time-analyzable tree-based NoC for mixed-criticality systems. In *Euromicro Conference on Digital System Design*.
[106] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. 2014. Time-analysable non-partitioned shared caches for real-time multicore systems. In *Design Automation Conference (DAC'14)*.
[107] M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. 2014. Timing verification of fault-tolerant chips for safety-critical applications in harsh environments. *IEEE Micro* 34, 6 (2014).
[108] Z. R. Stephenson, J. Abella, and T. Vardanega. 2013. Supporting industrial use of probabilistic timing analysis with explicit argumentation. In *11th IEEE International Conference on Industrial Informatics (INDIN'13)*.
[109] N. Topham and A. Gonzalez. 1999. Randomized cache placement for eliminating conflicts. *IEEE Transactions on Computers* 48, 2 (1999).
[110] B. Villalba Frias, L. Palopoli, L. Abeni, and D. Fontanelli. 2017. Probabilistic real-time guarantees: There is life beyond the i.i.d. assumption. In *Real-Time and Embedded Technology and Applications Symposium (RTAS'17)*.
[111] F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovin, Z. R. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetti, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. 2015. Timing analysis of an avionics case study on complex hardware/software platforms. In *Design, Automation & Test in Europe Conference & Exhibition (DATE'15)*.
[112] Reinhard Wilhelm, et al. 2008. The worst-case execution time problem: Overview of methods and survey of tools. *ACM TECS* 7, 3 (2008).
[113] S. Zhou. 2010. *An Efficient Simulation Algorithm for Cache of Random Replacement Policy*. Springer, Berlin.
[114] M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. 2015. EPC: Extended path coverage for measurement-based probabilistic timing analysis. In *Real-Time Systems Symposium (RTSS'15)*.

Received July 2017; revised August 2018; accepted October 2018