Leibniz Transactions on Embedded Systems

Volume 6 | Issue 1 | May 2019

Vol. 6, Issue 1 ISSN 2199-2002 http://www.dagstuhl.de/lites

ISSN 2199-2002

Published online and open access by
the European Design and Automation Association (EDAA) / EMbedded Systems Special Interest Group (EMSIG) and Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.
Online available at
http://www.dagstuhl.de/dagpub/2199-2002.

Publication date
May 2019

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier
10.4230/LITES-v006-i001

Aims and Scope
LITES aims at the publication of high-quality scholarly articles, ensuring efficient submission, reviewing, and publishing procedures. All articles are published open access, i.e., accessible online without any costs. The rights are retained by the author(s).

LITES publishes original articles on all aspects of embedded computer systems, in particular: the design, the implementation, the verification, and the testing of embedded hardware and software systems; the theoretical foundations; single-core, multi-processor, and networked architectures and their energy consumption and predictability properties; reliability and fault tolerance; security properties; and on applications in the avionics, the automotive, the telecommunication, the medical, and the production domains.

Contact
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
LITES, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
lites@dagstuhl.de
http://www.dagstuhl.de/lites

Contents

Local Planning Semantics: A Semantics for Distributed Real-Time Systems*

Mahieddine Dellabani
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
mahieddine.dellabani@univ-grenoble-alpes.fr

Jacques Combaz
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
saddek.bensalem@univ-grenoble-alpes.fr

Saddek Bensalem
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
jacques.combaz@univ-grenoble-alpes.fr

Marius Bozga
University Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, 38000 Grenoble, France
marius.bozga@univ-grenoble-alpes.fr

Abstract

Design, implementation and verification of distributed real-time systems are acknowledged to be very hard tasks. Such systems are prone to different kinds of delay, such as execution time of actions or communication delays implied by distributed platforms. The latter increase considerably the complexity of coordinating the parallel activities of running components. Scheduling such systems must cope with those delays by proposing execution strategies ensuring global consistency while satisfying the imposed timing constraints. In this paper, we investigate a formal model for such systems as compositions of timed automata subject to multiparty interactions, and propose a semantics aiming to overcome the communication delays problem through anticipating the execution of interactions. To be effective in a distributed context, scheduling an interaction should rely on (as much as possible) local information only, namely the state of its participating components. However, as shown in this paper these information is not always sufficient and does not guarantee a safe execution of the system as it may introduce deadlocks. Moreover, delays may also affect the satisfaction of timing constraints, which also corresponds to deadlocks in the former model. Thus, we also explore methods for analyzing such deadlock situations and for computing deadlock-free scheduling strategies when possible.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory
Keywords and Phrases Distributed Real-Time Systems, Timed Automata, Formal Verification
Digital Object Identifier 10.4230/LITES-v006-i001-a001
Received 2017-10-03 **Accepted** 2018-10-08 **Published** 2019-02-18

* This work has been supported by the European Union's Horizon 2020 research and innovation programme under grant agreement #730080.

1 Introduction

Nowadays, real-time systems are ubiquitous in several application domains, and such an emergence led to an increasing need of performance: resources, availability, concurrency, etc. This expansion initiates a shift from the use of single processor based hardware platforms, to large sets of interconnected and distributed computing nodes. Moreover, it prompts the birth of a new family of systems known as *Networked Embedded Systems*, that are intrinsically distributed. Such an evolution stems from an increase in complexity of real-time software embedded on such platforms (e.g. electronic control in avionics and automotive domains [13]), and the need to integrate formerly isolated systems [24] so that they can cooperate as well as share resources improving thus functionality and reducing costs.

To deal with such complexity, the community of safety critical systems often restricts its scope to predictable systems, which are represented with domain specific models (e.g. periodic tasks, synchronous systems, time-deterministic systems) for which the range of possible executions is small enough to be easily analyzed, allowing the pre-computation of optimal control strategies. *Networked Embedded Systems* usually describe a set of real-time systems, distributed across several platforms, and interacting through a network. Because of their adaptive behavior, the standard practice when implementing such systems is not to rely on models for pre-computation of execution strategies but rather to design systems dynamically adapting at runtime to the actual context of execution. Such approaches, however, do not offer any formal guarantee of timeliness. Also, the lack of a priori knowledge on system behavior leaves also little room for static optimization.

Model-based development is one promising approach in building distributed real-time systems. First, an application model expressing a timed abstraction of the application behavior is built. This abstraction is platform independent, meaning that it does not consider any hardware specification such as communication delays or CPU(s) speed, which allows to: *(i)* model the system at early stages without any knowledge of the target platform, and *(ii)* verify the obtained model against some safety properties (functional requirements). Thereafter, the application source code, which represents the actual implementation of the system on a given platform, is automatically generated from the high level model. Then, the big challenge becomes how to verify the timing behavior of the implementation, since a lot of assumptions drop such as atomic execution of actions or timeless communication delays. In this paper, we propose a model-based approach aiming to mitigate the communication delays of distributed platforms. In this approach, systems consist of components represented as timed automata that may synchronize on particular actions to coordinate their activities. We contribute to this research field by proposing a different semantics than the usual semantics of timed automata. This semantics aims to distinguish between the decision dates for executing interactions and their actual execution dates by introducing a notion of scheduling on a semantics level. The idea behind this practice is to distinguish between the date at which interactions are executed and the date at which the execution decisions are effectively made. This will particularly help to anticipate the execution of interactions at least some delay beforehand, corresponding to the actual worst estimation of communication delays of a given platform, which will alleviate the effect of those delays on the system behavior.

This work is an extension of our work presented in [16]. We extend our previous work by *(1)* defining a more mature and realistic semantics for planning interactions. Especially, we introduce a lower bound horizon for planning interactions, that is, we impose at least a minimum delay, representing communication delays, between the effective planning of an interaction and its execution. In other words, immediate (timeless) planning is no longer allowed. Thereafter, we show that by enforcing a minimum delay between interactions planning and their executions, we may engender situational blocking situations that were nonexistent at first. We provide *(2)* a

formal characterization of such blocking situations and *(3)* suggest an execution strategy aiming to avoid the latter. Furthermore, *(4)* if no execution strategy guarantying safe executions can be found, we propose an alternative method based on real-time controller synthesis approach as well as a discussion comparing both approaches.

The rest of the paper is organized as follows. Section 2 gives preliminary definitions of timed automata with respect to multiparty interactions as well as predicates definitions needed for the rest of the paper. In Section 3, we present our extended local planning semantics, discuss its relation with the usual timed automata semantics, and give sufficient conditions that formally characterize its deadlock states. Then, Section 4 presents an execution strategy aiming to enforce the correctness of the presented approach and find a deadlock free execution scenario when possible. Additionally, Section 5 explains how the local planning semantics can be formalized as a real-time controller synthesis problem and provide, thus, an alternative method for finding an execution strategy for such semantics. We also highlight the key issues met during our reflection and discuss important modeling points when using this technique. An implementation of the proposed is approach is described in Section 6 along with experimental results conducted on real-life case studies. Finally, the related works are presented in Section 7 and the conclusion given in Section 8.

2 Timed Systems and Properties

In the framework of the present paper, components are timed automata and systems are compositions of timed automata with respect to multiparty interactions. The timed automata we use are essentially the ones from [4], however, slightly adapted to embrace a uniform notation throughout the paper.

Given a set of clock $\mathcal{X}$, a clock constraint is an expression of the form:

$$c := \mathit{true} \mid x \sim ct \mid x - y \sim ct \mid c \wedge c \mid \mathit{false},$$

with $x, y \in \mathcal{X}$, $\sim \in \{<, \leq, =, \geq, >\}$ and $ct \in \mathbb{Z}$. We denote by $\mathcal{C}(\mathcal{X})$ the set of clock constraints over $\mathcal{X}$.

▶ **Definition 1** (Component). A *component* is a tuple $B = (\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathsf{tpc})$ where $\mathcal{L}$ is a finite set of *locations*, $\ell_0 \in \mathcal{L}$ is an *initial* location, $\mathcal{A}$ a finite set of *actions*, $\mathcal{X}$ is a finite set of *clocks*, $\mathcal{T} \subseteq \mathcal{L} \times (\mathcal{A} \times \mathcal{C}(\mathcal{X}) \times 2^{\mathcal{X}}) \times \mathcal{L}$ is a set of *transitions* labeled with an action, a guard, and a set of clocks to be reset, and $\mathsf{tpc} : \mathcal{L} \to \mathcal{C}(\mathcal{X})$ assigns a *time progress condition* $\mathsf{tpc}(\ell)$ to each location $\ell \in \mathcal{L}$. Notice that time progress conditions are backward closed, that is, they are restricted to conjunctions of constraints of the form $x \leq ct$.

Throughout the paper, we assume components that are deterministic timed automata, that is, at a given location ℓ and for a given action a, there is at most one outgoing transition from ℓ labeled by a. Given a timed automaton $(\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathsf{tpc})$, we write $\ell \xrightarrow{a,g,r} \ell'$ if there exists a transition $\tau = (\ell, (a, g, r), \ell') \in \mathcal{T}$. We also denote by $\mathit{guard}(a, \ell)$ the clock constraints of the transition labeled by a and outgoing from ℓ if it exists, and *false* otherwise and we write:

$$\mathit{guard}(a, \ell) = \begin{cases} g, & \text{if } \exists \tau = (\ell, (a, g, r), \ell') \in \mathcal{T} \\ \mathit{false}, & \text{otherwise} \end{cases}$$

Before recalling the semantics of a component, we first fix some notations. Let $\mathcal{V}(\mathcal{X})$ be the set of all clock valuation functions $v : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and v_0 be the clock valuation assigning zero to all clocks. For a clock constraint c, $c(v)$ is a boolean value corresponding to the evaluation of c on v. For a valuation $v \in \mathcal{V}$ and for $\delta \in \mathbb{R}_{\geq 0}$, $v + \delta$ is the valuation satisfying $(v + \delta)(x) = v(x) + \delta$ for

any x, while for a subset of clocks r, $v[r]$ is the valuation obtained from v by resetting clocks of r, i.e., $v[r](x) = 0$ for $x \in r$, $v[r](x) = v(x)$ otherwise. We denote by $c + \delta$ the clock constraint c shifted by δ, i.e. such that $(c + \delta)(v)$ iff $c(v + \delta)$. We also consider the classical *backward* and *forward* operators [30] on clock constraints, i.e. $(\swarrow c)(v)$ iff $\exists \delta \geq 0 \ . \ c(v + \delta)$ and $(\nearrow c)(v)$ iff $\exists \delta \geq 0 \ . \ c(v - \delta)$. In what follows, we also use two variants of the backward operator considering lower bounds $l \in \mathbb{Z}_{\geq 0}$ and upper bounds $u \in \mathbb{Z}_{\geq 0} \cup \{+\infty\}$: $(\swarrow^{l} c)(v)$ iff $\exists \delta \geq l \ . \ c(v + \delta)$ and $(\swarrow_{l}^{u} c)(v)$ iff $\exists \delta \ . \ l \leq \delta \leq u \wedge c(v + \delta)$.

▶ **Definition 2** (Semantics). A component $B = (\mathcal{L}, \ell_0, \mathcal{A}, \mathcal{X}, \mathcal{T}, \mathsf{tpc})$ defines the labeled transition system (LTS) $(\mathrm{Q}, q_0, \mathcal{A} \cup \mathbb{R}_{>0}, \rightarrow)$ where:

- $\mathrm{Q} = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ denotes the *states* of B, $q_0 = (\ell_0, v_0)$ is the initial state.
- $\rightarrow \subseteq \mathrm{Q} \times (\mathcal{A} \cup \mathbb{R}_{>0}) \times \mathrm{Q}$ denotes the set of transitions between states according to the rules:
  - $(\ell, v) \xrightarrow{a} (\ell', v[r])$ if $\ell \xrightarrow{a,g,r} \ell'$ and $g(v)$ and $\mathsf{tpc}(\ell')(v[r])$ (action step).
  - $(\ell, v) \xrightarrow{\delta} (\ell, v + \delta)$ if $\mathsf{tpc}(\ell)(v + \delta)$ for $\delta \in \mathbb{R}_{>0}$ (time step).

A *run* ϱ of B is an execution sequence that alternates action steps and time steps, that is:

$$\varrho = q_0 \sigma_0 q_1 \sigma_1 q_2 \ldots, \text{ such that } q_i \in \mathrm{Q}, \ q_i \xrightarrow{\sigma_i} q_{i+1}, \text{ and } i \in \mathbb{Z}_{>0}, \sigma_i \in \mathcal{A} \cup \mathbb{R}_{>0}.$$

We say that a state (ℓ, v) is *reachable* if there is an execution sequence from the initial configuration (ℓ_0, v_0) leading to (ℓ, v). In this paper, we always assume components with *well formed guards*, that is, transitions $\ell \xrightarrow{a,g,r} \ell'$ satisfy $g(v) \Rightarrow \mathsf{tpc}(\ell)(v) \wedge \mathsf{tpc}(\ell')(v[r])$ for any $v \in \mathcal{V}$. This ensures that the reachable states always satisfy the time progress conditions, i.e. if (ℓ, v) is reachable then we have $\mathsf{tpc}(\ell)(v)$. Consequently, the action step of Definition 2 can be simplified as:

$$(\ell, v) \xrightarrow{a} (\ell', v[r]) \text{ if } \ell \xrightarrow{a,g,r} \ell' \text{ and } g(v)$$

Notice that the set of reachable states is in general infinite, but it can be partitioned into a finite number of symbolic states [30, 7, 20]. A symbolic state is defined by a pair (ℓ, ζ) where, ℓ is a location of B, and ζ is a zone, i.e. a set of clock valuations defined by a clock constraint (as defined in Definition 1). Efficient algorithms for computing symbolic states and operations on zones are fully described in [7]. Given symbolic states $\{(\ell_j, \zeta_j)\}_{j \in J}$ of B, the predicate $\mathit{Reach}(B)$ characterizing the reachable states can be expressed as:

$$\mathit{Reach}(B) = \bigvee_{j \in J} \mathsf{at}(\ell_j) \wedge \zeta_j,$$

where $\mathsf{at}(\ell_j)$ is true on states whose location is ℓ_j, and clock constraint ζ_j is straightforwardly applied to clock valuation functions of states.

We define the predicate $\mathit{Enabled}(a)$ characterizing states (ℓ, v) at which an action a is enabled, i.e. such that $(\ell, v) \xrightarrow{a} (\ell', v')$ for some (ℓ', v'). It can be formally written as:

$$\mathit{Enabled}(a) = \bigvee_{\ell \in \mathcal{L}} \mathsf{at}(\ell) \wedge \mathit{guard}(a, \ell).$$

A state (ℓ, v) is said *urgent* if time cannot progress from (ℓ, v), that is, there is no $\delta \in \mathbb{R}_{>0}$ such that $(\ell, v) \xrightarrow{\delta} (\ell, v')$. Urgent states are characterized by the predicate:

$$\mathit{Urgent}(B) = \bigvee_{\ell \in \mathcal{L}} \mathsf{at}(\ell) \wedge \mathit{urg}(\ell) \tag{1}$$

where $\mathit{urg}(\ell)$ is a clock constraint characterizing the valuations from which time cannot progress with respect to the time progress condition of ℓ, that is, it is defined by $\mathit{urg}(\ell) = \bigvee_{i=1}^{m} (x_i \geq ct_i)$ if

$\mathsf{tpc}(\ell) = \bigwedge_{i=1}^{m} x_i \leq ct_i$. Notice that due to well-formed guards, an urgent reachable state satisfies also (1) if inequalities $x_i \geq ct_i$ on clocks are replaced by equalities $x_i = ct_i$ in the expression of $urg(\ell)$.

Following [31], we require that components or systems execute forever. This is referred to as *the requirement of progress* which can be divided split into discrete and time progress captured respectively by the notion of deadlocks and timelocks.

▶ **Definition 3** (Deadlock and action-time-lock)**.** We say that a state (ℓ, v) of a component B is *deadlock* if, with respect to its semantics, no action can be executed from (ℓ, v) and from its successors, that is:

$$Deadlock(B) = \neg\Big(\exists a \in \mathcal{A} \,.\, (\ell, v) \xrightarrow{a} (\ell', v') \;\vee \exists \delta > 0 \,.\, (\ell, v) \xrightarrow{\delta} (\ell, v+\delta) \xrightarrow{a} (\ell', v')\Big).$$

Deadlock states are characterized by the following predicate:

$$Deadlock(B) = \neg\Big(\bigvee_{a\in\mathcal{A}} \swarrow \big(Enabled(a) \wedge \mathsf{tpc}(\ell)\big)\Big).$$

Because of well-formed guards this could be simplified into:

$$Deadlock(B) = \bigwedge_{a\in\mathcal{A}} \neg\big(\swarrow Enabled(a)\big).$$

A deadlock (ℓ, v) is called an *action-time-lock* when no interaction can execute nor time can progress from (ℓ, v), that is:

$$ActionTimeLock(B) = \neg\Big(\exists a \in \mathcal{A} \,.\, (\ell, v) \xrightarrow{a} (\ell', v') \;\vee\; \exists \delta > 0 \,.\, (\ell, v) \xrightarrow{\delta} (\ell, v+\delta)\Big).$$

Action-time-lock states are characterized by the following predicate:

$$ActionTimeLock(B) = \Big(\bigwedge_{a\in\mathcal{A}} \neg Enabled(a)\Big) \;\wedge\; \Big(\bigvee_{\ell\in\mathcal{L}} \mathsf{at}(\ell) \wedge urg(\ell)\Big).$$

Deadlocks are situations from which a component is stuck at a given location without being able to progress by executing an action, which must be avoided in reactive systems. Action-time-locks are modeling errors and consist in deadlocks from which time cannot progress.

We denote by $\mathsf{time}(\varrho, i)$ the total elapsed time until point i, that is, $\sum_{j<i} \sigma_j$ such as $\sigma_j \in \mathbb{R}_{>0}$. In the same way, $\mathsf{time}(\varrho)$ represents the total elapsed time during ϱ, and is defined to be the limit of $\mathsf{time}(\varrho, i)$ if the sequence converges and ∞ otherwise.

▶ **Definition 4** (Zeno runs and Timelocks)**.** Let ϱ be an infinite run such that $\mathsf{time}(\varrho) \neq \infty$. Such a run violates the time progress requirements (only a finite number of events can occur in a finite amount of time), and is called *zeno*. Given a component B, a state of B is *timelock* if all infinite runs starting form that state are zeno.

In [31], it was shown that the class of timed automata in which 1 time unit is elapsed in every structural loop, also known as *strongly non-zeno*, is non zeno. An interesting property of this class is that it preserves non-zenoness under composability. Thus, checking the requirements of progress of a given system will boil down to checking its deadlock freedom.

In our framework, components communicate by means of *multiparty interactions*. A multiparty interaction is a rendez-vous synchronization between actions of a fixed subset of components. It takes place only if all the participants agree to execute the corresponding actions. Given n

components B_i, $i = 1, \ldots, n$, with disjoint sets of actions $\mathcal{A}_i$, an interaction is a subset of actions $\alpha \subseteq \cup_{1 \leq i \leq n} \mathcal{A}_i$ containing at most one action per component, i.e. $\alpha \cap \mathcal{A}_i$ is either empty or a singleton $\{a_i\}$. That is, an interaction α can be put in the form $\{a_i\}_{i \in I}$ with $I \subseteq \{1, \ldots, n\}$ and $a_i \in \mathcal{A}_i$ for all $i \in I$. We denote by $\text{part}(\alpha)$, the set of components *participating* in α, that is, $\text{part}(\alpha) = \{B_i\}_{i \in I}$. We say that two interactions α and β are conflicting, denoted by $\alpha \# \beta$, $\text{part}(\alpha) \cap \text{part}(\beta) \neq \emptyset$.

The semantics of a composition is interpreted at runtime by evaluating enabled interactions based on current states of the system components. In a composition of n components $B_{i \in \{1,\ldots,n\}}$ synchronizing through the interaction set γ, denoted by $\gamma(B_1, \ldots, B_n)$, an action a_i can execute only as part of an interaction α such that $a_i \in \alpha$, that is, along with the execution of all other actions $a_j \in \alpha$, which corresponds to the usual notion of multiparty interaction.

▶ **Definition 5** (Standard Semantics of a Composition)**.** Given a set of components $\{B_1, \ldots, B_n\}$ and an interaction set γ. The (standard) semantics of the composition $S = \gamma(B_1, \ldots, B_n)$ w.r.t the set of interactions γ, is the LTS $(\text{Q}_g, q_0, \gamma \cup \mathbb{R}_{>0}, \rightarrow_\gamma)$ where:

- $\text{Q}_g = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ is the set of global states, where $\mathcal{L} = \mathcal{L}_1 \times \ldots \times \mathcal{L}_n$ and $\mathcal{X} = \bigcup_{i=1}^{n} \mathcal{X}_i$. We write a state $q = (\ell, v)$ where $\ell = (\ell_1, \ldots, \ell_n) \in \mathcal{L}$ is a global location and $v \in \mathcal{V}(\mathcal{X})$ is a global clock valuation. $q_0 = (\ell_0, v_0)$ is the initial state, where $\ell_0 = (\ell_0^1, \ldots, \ell_0^n)$ and v_0 is the clock valuation assigning 0 to all clocks.
- $\rightarrow_\gamma \subseteq \text{Q} \times (\gamma \cup \mathbb{R}_{>0}) \times \text{Q}$ is the set of labeled transitions defined by the rules:
  - $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$ for $\alpha = \{a_i\}_{i \in I} \in \gamma$, if $\forall i \in I\ (\ell_i, v_i) \xrightarrow{a_i} (\ell_i', v_i')$ and $\forall i \notin I\ (\ell_i, v_i) = (\ell_i', v_i')$.
  - $(\ell, v) \xrightarrow{\delta}_\gamma (\ell, v + \delta)$ for $\delta \in \mathbb{R}_{>0}$ if $\forall i \in \{1, \ldots, n\}\ \mathsf{tpc}_i(\ell_i)(v_i + \delta)$ where v_i denotes the restriction of v to clocks $\mathcal{X}_i$ of B_i.

To simplify notations, predicates defined on individual components B_i are straightforwardly interpreted on states (ℓ, v) of a composition $S = \gamma(B_1, \ldots, B_n)$ by considering the projection (ℓ_i, v_i) of (ℓ, v) on B_i, which is such that $\ell = (\ell_1, \ldots, \ell_n)$ and v_i is restriction of v to clocks $\mathcal{X}_i$ of B_i. For instance, $\mathsf{at}(\ell_i)$ evaluates to true on (ℓ, v) iff $\ell \in \mathcal{L}_1 \times \ldots \times \mathcal{L}_{i-1} \times \{\ell_i\} \times \mathcal{L}_{i+1} \times \ldots \times \mathcal{L}_n$. Similarly, clock constraints of components B_i are applied to clock valuation functions v of the composition by restricting v to clocks $\mathcal{X}_i$ of B_i. This allows to write the predicate $\mathit{Enabled}(\alpha)$ characterizing states (ℓ, v) from which an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$ can be executed, i.e., such that $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$, as:

$$
\begin{aligned}
\mathit{Enabled}(\alpha) = & \bigwedge_{i \in I} \mathit{Enabled}(a_i) \\
= & \bigwedge_{i \in I} \bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \wedge \mathit{guard}(a_i, \ell_i) \\
= & \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \ldots, \ell_n)}} \mathsf{at}(\ell) \wedge \bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \mathit{guard}(a_i, \ell_i).
\end{aligned}
$$

Notice that the above formulation of $\mathit{Enabled}(\alpha)$ corresponds to locations enumeration of all components participating in interaction α. In practice, we rather consider only a subset of locations $\mathcal{L}_\alpha \subseteq \mathcal{L}$, from which the execution of α is possible. This corresponds to $\prod_{i \in I} |\mathcal{L}_{a_i}|$ possible configuration, where $\mathcal{L}_{a_i} \subseteq \mathcal{L}_i$ is a subset of locations from which there exists a transition labeled by action $a_i \in \alpha$, and $|\mathcal{L}_{a_i}|$ denotes the cardinality of $\mathcal{L}_{a_i}$, which is reasonably small in pratical examples but can be (at the worst case) equal to $\prod_{i \in I} |\mathcal{L}|$.

The definitions of run, reachable states, deadlocks and action-time-locks of components are also trivially extended to composition of components. Deadlocks of a composition $S = \gamma(B_1, \ldots, B_n)$

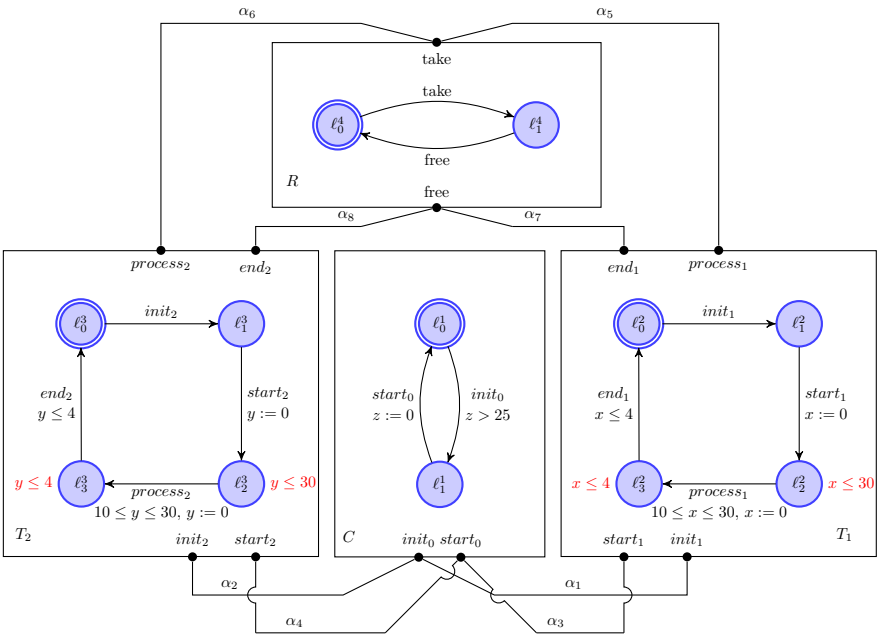


Figure 1 Task Manager.

can be characterized as follows:

$$Deadlock(S) = \bigvee_{\ell=(\ell_1,\dots\ell_n)\in\mathcal{L}} \mathsf{at}(\ell) \wedge \Big[\bigwedge_{\alpha\in\gamma} \neg \swarrow \Big(Enabled(\alpha) \wedge \bigwedge_{1\leq i\leq n} \mathsf{tpc}_i(\ell_i)\Big)\Big], \tag{2}$$

and action-time-locks by:

$$ActionTimeLock(S) = \Big(\bigwedge_{\alpha\in\gamma} \neg Enabled(\alpha)\Big) \wedge \Big(\bigvee_{1\leq i\leq n} \bigvee_{\ell_i\in\mathcal{L}_i} \mathsf{at}(\ell_i) \wedge urg(\ell_i)\Big).$$

▶ **Example 6** (Running Example). Let us consider as a running example the composition of four components C, T_1, T_2, and R of Figure 1. Component C represents a controller that initializes then releases tasks T_1 and T_2. Tasks use the shared resource R during their execution. To implement such behavior, we consider the following interactions between C, R, and T_1: $\alpha_1 = \{init_0, init_1\}$, $\alpha_3 = \{start_0, start_1\}$, $\alpha_5 = \{take, process_1\}$, $\alpha_7 = \{free, end_1\}$, and similar interactions α_2, α_4, α_6, α_8 for task T_2, as shown by connections on Figure 1. The controller is responsible for firing the execution of each task. First, it non-deterministically initializes one of the two tasks, i.e. executes α_1 or α_2, and then releases it through interaction α_3 or α_4. Tasks perform their processing independently of the controller, after being granted an access to the shared resource (α_5 or α_6). When finished, a task releases the resource (interactions α_7 or α_8) and goes back to its initial location. An example of execution sequence of the system of Figure 1 is given below, in which valuations v of clocks x, y, and z are represented as a tuples $(v(x), v(y), v(z))$:

$$((\ell_0^1,\ell_0^2,\ell_0^3,\ell_0^4),(0,0,0)) \xrightarrow{26}_\gamma ((\ell_0^1,\ell_0^2,\ell_0^3,\ell_0^4),(26,26,26)) \xrightarrow{\alpha_1}_\gamma ((\ell_1^1,\ell_1^2,\ell_0^3,\ell_0^4),(26,26,26)) \xrightarrow{\alpha_3}_\gamma$$
$$((\ell_0^1,\ell_2^2,\ell_0^3,\ell_0^4),(0,26,0)) \xrightarrow{10}_\gamma ((\ell_0^1,\ell_2^2,\ell_0^3,\ell_0^4),(10,36,10)) \xrightarrow{\alpha_5}_\gamma ((\ell_0^1,\ell_3^2,\ell_0^3,\ell_1^4),(0,36,10)) \xrightarrow{2}_\gamma$$
$$((\ell_0^1,\ell_3^2,\ell_0^3,\ell_1^4),(2,38,12)) \xrightarrow{\alpha_2}_\gamma ((\ell_1^1,\ell_3^2,\ell_1^3,\ell_1^4),(2,38,12))$$

3 Local Planning of Interactions

In the previous Section, we presented a timed automata model for representing timed systems with multiparty interactions. The semantics of such model is based on the notion of *global states*, that is, interactions executions are based not only on the state of participating components but on the states of all components of the system. Conversely, a distributed system can be seen as a collection of loosely coupled components that communicate with a scheduling layer [23, 29] which, based on a partial view of the system, is responsible of taking decisions for interactions executions and their effective execution dates. Additionally, high-level coordination primitives, such as multiparty synchronizations (interactions) are rarely built-in primitives of distributed platforms. Hence, their implementation on a distributed platform requires synchronization protocols responsible for realizing synchronizations using simpler primitives such as point-to-point messages passing as explained in [29]. This is classically implemented using one or more additional coordination component(s) observing the system state and deciding on interactions execution, which adds on a communication overhead not reflected by the semantics of Section 2.

This motivates the introduction of the *local planning semantics*. This semantics differs from the standard semantics of timed automata in two main aspects: *(i)* interactions execution is based only on partial state of the system, that is, based only on the state of components participating in the considered interaction. Thus, it allows to decide locally without monitoring the entire system. *(ii)* it distinguishes between the execution decision of an interaction (its *planning*), and the execution itself. This distinction allows us to impose a delay between the planning of an interaction and its execution. The latter is constrained by the (maximal) communication latency induced by the execution platform, which is a parameter of the semantics. It is correct in the sense that it refines (it is included in) the semantics of Section 2. However, being based on local states, planning decisions are too permissive and may introduce deadlocks when they are not compatible with the global state of the system.

3.1 Definition of the LPS

Let $S = \gamma(B_1, \cdots, B_n)$ be a composition of components $B_1, \ldots, B_n$ with disjoint set of locations, actions and clocks. We define the predicate $\mathit{Plannable}(\alpha, \delta)$ characterizing states (ℓ, v) from which an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$ is enabled in $\delta \in \mathbb{R}_{\geq 0}$ units of time (if time progresses by δ units of time), that is, such that $\mathit{Enabled}(\alpha)$ evaluates to true on state $(\ell, v + \delta)$. It is characterized by:

$$\mathit{Plannable}(\alpha, \delta) = \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \cdots, \ell_n)}} \mathsf{at}(\ell) \wedge \bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \big(\mathit{guard}(a_i, \ell_i) + \delta\big) \tag{3}$$

Notice that for an interaction α the predicate $\mathit{Plannable}(\alpha, \delta)$ depends only on states of components of $\mathrm{part}(\alpha)$, which motivates the following property.

▶ **Property 7.** *Let (ℓ, v) be a state of the composition S. For any interactions $\alpha, \beta \in \gamma$ such that, $(\ell, v) \xrightarrow{\beta}_{\gamma} (\ell', v')$ and $\mathrm{part}(\alpha) \cap \mathrm{part}(\beta) = \emptyset$, if $\mathit{Plannable}(\alpha, \delta)$ holds at state (ℓ, v) then it still holds at state (ℓ', v').*

This property derives directly from the fact that executing an interaction β does not change the states of components participating in an interaction α, provided that α and β have disjoint sets of participating components, and thus, $\mathit{Plannable}(\alpha, \delta)$ is not affected by the execution of β in this case. In the following, we say that two interactions α and β *conflicts* when they have common participating components, that is, when $\mathrm{part}(\alpha) \cap \mathrm{part}(\beta) \neq \emptyset$, and we write $\alpha \# \beta$. We denote by $\mathit{conf}(\alpha)$ the set of interactions conflicting with α, that is, $\mathit{conf}(\alpha) = \{\beta \in \gamma \mid \alpha \# \beta\}$.

▶ **Property 8.** *Let (ℓ, v) and $(\ell, v+\delta')$, with $\delta' \in \mathbb{R}_{>0}$ be two states of the composition S. For an interaction $\alpha \in \gamma$, if Plannable(α, δ) holds at state (ℓ, v) then Plannable$(\alpha, \delta - \delta')$ also holds at any state $(\ell, v+\delta')$ such that $\delta' \leq \delta$.*

This property can be found directly by expressing the predicate 3 on state $(\ell, v+\delta')$.

As previously explained, due to communication latencies induced by execution platforms we assume that interactions cannot be planned in δ units of time if $\delta < h_{\min}$, where $h_{\min} \in \mathbb{Z}_{\geq 0}$ is a parameter representing the minimal *planning horizon*, which should represent the upper bound communication latencies. Notice that for the sake of simplicity, we consider a global parameter $h_{\min}$ but we could also assume different parameters for each interaction. Additionally, we also consider upper bounds planning horizons $h_{\max} : \gamma \to \mathbb{Z}_{\geq 0} \cup \{+\infty\}$ for each interaction such that for any $\alpha \in \gamma$ we have $h_{\max}(\alpha) \geq h_{\min}$. We denote by $h^{\infty}_{\max}$ the upper planning horizon assigning infinity to every $h_{\max}(\alpha)$. A direct consequence of introducing the planning horizons is that every interaction α can be planned only using a horizon δ satisfying $h_{\min} \leq \delta \leq h_{\max}(\alpha)$, meaning that every component $B \in \text{part}(\alpha)$ will be blocked for a duration between $[h_{\min}, h_{\max}(\alpha)]$. Observe that while $h_{\min}$ represents the worst case estimation of the communication delays for a given platform, the parameters $h_{\max}(\alpha)$ will be used later to find a strategy that avoids deadlocks by restricting the amount of time components can be blocked for.

For an interaction α we define the predicate $\mathit{Plannable}(\alpha)$ characterizing states from which α can be planned in a delay respecting the planning horizons $h_{\min}$ and $h_{\max}(\alpha)$, that is:

$$\mathit{Plannable}(\alpha) \Leftrightarrow \exists \delta \in \mathbb{R}_{\geq 0} \;.\; h_{\min} \leq \delta \leq h_{\max}(\alpha) \;\wedge\; \mathit{Plannable}(\alpha, \delta),$$

It can be written equivalently as follows:

$$\mathit{Plannable}(\alpha) = \bigvee_{\substack{\ell \in \mathcal{L} \\ \ell = (\ell_1, \cdots, \ell_n)}} \mathsf{at}(\ell) \wedge \swarrow_{h_{\min}}^{h_{\max}(\alpha)} \Big(\bigwedge_{\substack{i \in I \\ a_i \in \alpha}} \mathit{guard}(a_i, \ell_i) \Big) \tag{4}$$

▶ **Definition 9** (Plan)**.** A plan π is a function $\pi : \gamma \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ defining relative times for executing interactions. An interaction α is planned to execute in $\pi(\alpha)$ time units only if $\pi(\alpha) < +\infty$. Plans satisfy that for any two interactions $\alpha \neq \beta$ such that $\pi(\alpha) < +\infty$ and $\pi(\beta) < +\infty$, then the interactions α and β are not conflicting (i.e. $\neg(\alpha \# \beta)$).

We denote by π_0 the plan assigning $+\infty$ to every interaction of γ, that is, $\forall \alpha \in \gamma, \pi_0(\alpha) = +\infty$. For a plan π, we consider its minimum value $\min(\pi) = \min\{\pi(\alpha) | \alpha \in \gamma\}$. We also denote by $\mathit{conf}(\pi)$ the set of interactions conflicting with the plan π, i.e. $\mathit{conf}(\pi) = \{\alpha \mid \exists \beta \# \alpha.\ \pi(\beta) < +\infty\}$, and part$(\pi)$ the set of components participating in interactions planned by π, i.e. $\text{part}(\pi) = \{B_i \mid \exists \alpha \;.\; \pi(\alpha) < +\infty \wedge B_i \in \text{part}(\alpha)\}$. Notice that since π stores relative times, whenever time progresses by δ, the value $\pi(\alpha)$ assigned by π to an interaction α should be decreased by δ until it reaches 0, meaning that α has to execute. We write $\pi - \delta$ to describe the progress of time over the plan, that is, $(\pi - \delta)(\alpha) = \pi(\alpha) - \delta$ for interactions α such that $\pi(\alpha) < +\infty$. Similarly, $\pi[\alpha \mapsto \delta]$ assigns relative time δ to α, $\alpha \notin \mathit{conf}(\pi)$, into existing plan π, i.e. $(\pi[\alpha \mapsto \delta])(\beta) = \delta$ for $\beta = \alpha$, $(\pi[\alpha \mapsto \delta])(\beta) = \pi(\beta)$ otherwise.

▶ **Definition 10** (Local Planning Semantics)**.** Given a set of components $\{B_1, \cdots, B_n\}$ and an interaction set γ, we define the local planning semantics (LPS) of the composition $\gamma(B_1, \cdots, B_n)$, as the LTS $(\mathrm{Q}_p, q_{p_0}, \sum_p, \rightsquigarrow_\gamma)$ where:

- $\mathrm{Q}_p = \mathcal{L} \times \mathcal{V}(\mathcal{X}) \times \Pi$, where $\mathcal{L}$ is the set of global locations, $\mathcal{V}(\mathcal{X})$ is the set of global clock valuation, and Π is the set of plans.
- $\sum_p = \gamma \cup \mathbb{R}_{>0} \cup (\gamma \times \mathbb{R}_{\geq 0})$, where $(\gamma \times \mathbb{R}_{\geq 0})$ defines the action of planning interactions of γ and their relative times.

- $\rightsquigarrow_\gamma \subseteq \mathrm{Q}_p \times \sum_p \times \mathrm{Q}_p$ is the set of labeled transitions defined by the rules:
  - $(\ell, v, \pi) \overset{(\alpha,\delta)}{\rightsquigarrow}_\gamma (\ell, v, \pi[\alpha \mapsto \delta])$ for $\alpha \in \gamma$, $h_{\min} \leq \delta \leq h_{\max}(\alpha)$ and $\delta \neq +\infty$ if $\alpha \notin \mathit{conf}(\pi)$ and $\mathit{Plannable}(\alpha, \delta)$ holds on (ℓ, v, π).
  - $(\ell, v, \pi) \overset{\alpha}{\rightsquigarrow}_\gamma (\ell', v', \pi[\alpha \mapsto +\infty])$ for $\alpha \in \gamma$ if $\pi(\alpha) = 0$ and $(\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v')$.
  - $(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_\gamma (\ell, v+\delta, \pi-\delta)$ for $\delta \leq \min(\pi)$, $\ell = (\ell_1, \ldots, \ell_n)$, if $\mathsf{tpc}_i(\ell_i)(v+\delta)$ for components $B_i \in \mathrm{part}(\pi)$ and $\mathsf{tpc}_i(\ell_i)(v + \delta + h_{\min})$ for components $B_i \notin \mathrm{part}(\pi)$.

Remark that in the above definition as well as in what follows, predicates defined on states $(\ell, v) \in \mathrm{Q}_g = \mathcal{L} \times \mathcal{V}(\mathcal{X})$ of the standard semantics are straightforwardly interpreted on states $(\ell, v, \pi) \in \mathrm{Q}_p$ considering the projection (ℓ, v) of (ℓ, v, π) on Q_g.

States of the LPS do not include only locations and clock valuations, but also the relative execution times of the planned interactions stored by π. Initially, no interaction is planned, that is, initial states (ℓ_0, v_0, π_0) satisfy $\pi_0 = +\infty$. Planning an interaction α to be executed at a relative time $h_{\min} \leq \delta \leq h_{\max}(\alpha)$ corresponds to the operation $\pi[\alpha \mapsto \delta]$ on the plan, which can only be done if α is not conflicting with the latter, and becomes enabled if time progresses by δ (i.e. if $\mathit{Plannable}(\alpha, \delta)$). On the other hand, time progress not only updates clock values but also the plan by decreasing the relative execution times of the planned interactions. To force the execution of planned interactions when their relative execution times reach 0, time cannot progress more than the closest relative execution times of the interactions (more than $\min(\pi)$). As for the standard semantics, time progress is limited by the time progress conditions of the components, but with the following significant difference: Components $B_i \in \mathrm{part}(\pi)$ participating in planned interactions behave as in the standard semantics, that is, time can progress until their time progress conditions expire. For components $B_i \notin \mathrm{part}(\pi)$, i.e., that are not participating in planned interactions, we take into account the minimal delay $h_{\min}$ needed for planning and then executing an interaction: in components $B_i \notin \mathrm{part}(\pi)$ time can progress only up to $h_{\min}$ time units before their time progress conditions expire. By doing so, we ensure that there always remains enough time to plan interactions involving $B_i \notin \mathrm{part}(\pi)$, if they exist, and execute them before their time progress conditions expire.

▶ **Example 11.** Let us consider the following execution sequence for example of Figure 1 under the LPS with $h_{\min} = 2$ and $h_{\max} = h^{\infty}_{\max}$.

$$
\begin{array}{lll}
((\ell_0^1, \ell_0^2, \ell_0^3, \ell_0^4), (0,0,0), +\infty) & \overset{(\alpha_1,26)}{\rightsquigarrow}_\gamma ((\ell_0^1, \ell_0^2, \ell_0^3, \ell_0^4), (0,0,0), \{\alpha_1 \mapsto 26\}) & \overset{26}{\rightsquigarrow}_\gamma \\
((\ell_0^1, \ell_0^2, \ell_0^3, \ell_0^4), (26,26,26), \{\alpha_1 \mapsto 0\}) & \overset{\alpha_1}{\rightsquigarrow}_\gamma ((\ell_1^1, \ell_1^2, \ell_0^3, \ell_0^4), (26,26,26), +\infty) & \overset{(\alpha_3,2)}{\rightsquigarrow}_\gamma \\
((\ell_1^1, \ell_1^2, \ell_0^3, \ell_0^4), (26,26,26), \{\alpha_3 \mapsto 2\}) & \overset{2}{\rightsquigarrow}_\gamma ((\ell_1^1, \ell_1^2, \ell_0^3, \ell_0^4), (28,28,28), \{\alpha_3 \mapsto 0\}) & \overset{\alpha_3}{\rightsquigarrow}_\gamma \\
((\ell_0^1, \ell_2^2, \ell_0^3, \ell_0^4), (0,28,0), +\infty) & \overset{(\alpha_2,26)}{\rightsquigarrow}_\gamma ((\ell_0^1, \ell_2^2, \ell_0^3, \ell_0^4), (0,28,0), \{\alpha_2 \mapsto 26\}) & \overset{26}{\rightsquigarrow}_\gamma \\
((\ell_0^1, \ell_2^2, \ell_0^3, \ell_0^4), (26,54,26), \{\alpha_2 \mapsto 0\}) & \overset{\alpha_2}{\rightsquigarrow}_\gamma ((\ell_1^1, \ell_2^2, \ell_1^3, \ell_0^4), (26,54,26), +\infty) & \overset{(\alpha_4,2)}{\rightsquigarrow}_\gamma \\
((\ell_1^1, \ell_2^2, \ell_1^3, \ell_0^4), (26,54,26), \{\alpha_4 \mapsto 2\}) & \overset{2}{\rightsquigarrow}_\gamma ((\ell_1^1, \ell_2^2, \ell_1^3, \ell_0^4), (28,56,28), \{\alpha_4 \mapsto 0\}) & \overset{\alpha_4}{\rightsquigarrow}_\gamma \\
((\ell_0^1, \ell_2^2, \ell_2^3, \ell_0^4), (28,0,0), +\infty) & \overset{(\alpha_6,30)}{\rightsquigarrow}_\gamma & \\
((\ell_0^1, \ell_2^2, \ell_2^3, \ell_0^4), (28,0,0), \{\alpha_6 \mapsto 30\}) &&
\end{array}
$$

This execution sequence represents a path that alternates plan actions, time progress and execution of some interactions, and leads to the action-time-lock state $((\ell_0^1, \ell_2^2, \ell_2^3, \ell_0^4), (0,0,28), \{\alpha_6 \mapsto 30\})$. In fact, the time progress condition $x \leq 30$ in component T_1, imposes the planning of interaction α_7 at the latest h_{min} units of time before it becomes urgent. However, since interaction α_6 was planned in 28 units of time, α_7 cannot be planned since it is conflicting with α_6. This execution sequence shows that a given system action-time-locks under the local planning semantics , even if it is deadlock-free in the standard semantics.

3.2 Properties of the LPS

We use weak simulation to compare models of the standard semantics and the local planning semantics by considering the planning transitions unobservable. As shown in Example 11, the LPS does not preserve the deadlock freedom property of our system. Nevertheless, the following proves weak simulation relations between the two semantics.

▶ **Lemma 12.** *Given a reachable state (ℓ, v, π) of the LPS. If for $\alpha \in \gamma$, $\pi(\alpha) < +\infty \Rightarrow$ Plannable$(\alpha, \pi(\alpha))$.*

▶ **Proposition 13.** *An interaction can execute from a state (ℓ, v, π) in the LPS semantics only if it can execute from (ℓ, v) in the standard semantics, that is:*

$$\forall \alpha \in \gamma.(\ell, v, \pi) \overset{\alpha}{\rightsquigarrow}_\gamma (\ell', v', \pi') \Rightarrow (\ell, v) \xrightarrow{\alpha}_\gamma (\ell', v').$$

Proposition 13 is a consequence of Lemma 12: an interaction α can execute in the local planning semantics only if $\pi(\alpha) = 0$ (see Definition 9). That is, a state (ℓ, v, π) of the LPS from which α can execute satisfies *Plannable*$(\alpha, 0)$ or equivalently *Enabled*(α), which demonstrates that α can execute from (ℓ, v) in the standard semantics.

▶ **Proposition 14.** *Time can progress by δ at a state (ℓ, v, π) in the local planning semantics only if time can progress by δ at (ℓ, v) in the standard semantics, that is:*

$$\forall \delta \in \mathbb{R}_{>0}.(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_\gamma (\ell', v', \pi') \Rightarrow (\ell, v) \xrightarrow{\delta}_\gamma (\ell', v').$$

Proposition 14 is a direct consequence of the definition of time progress in the local planning semantics which is a restriction of the one in the standard semantics.

▶ **Corollary 15.** *If a state (ℓ, v, π) is reachable in the local planning semantics, then the state (ℓ, v) is reachable in the standard semantics.*

Corollary 15 is obtained from Propositions 13 and 14 and the fact that planning transitions (labeled by (α, δ)) affect only the plan π in states (ℓ, v, π) of the LPS.

▶ **Definition 16** (Weak Simulation)**.** A weak simulation over $A = (\mathrm{Q}_A, q_{A_0} \sum \cup \{\beta\}, \rightarrow_A)$ and $B = (\mathrm{Q}_B, q_{B_0} \sum \cup \{\beta\}, \rightarrow_B)$, where β actions represent silent (unobservable) action, is a relation $R \subseteq \mathrm{Q}_A \times \mathrm{Q}_B$ such that:

- $\forall (q, r) \in R, a \in \sum .q \xrightarrow{a}_A q' \implies \exists r' : (q', r') \in R \wedge r \xrightarrow{\beta^* a \beta^*}_B r'$ and,
- $\forall (q, r) \in R : q \xrightarrow{\beta}_A q' \implies \exists r' : (q', r') \in R \wedge r \xrightarrow{\beta^*} r'$.

B simulates A, denoted by $A \sqsubseteq_R B$, means that B can do everything A does.

The definition of weak simulation is based on the unobservability of $\beta-$transitions. In our case, $\beta-$transitions corresponds to planning transitions. Let LTS_g and LTS_p be respectively the underlying labeled transition system of the standard semantics and the local planning semantics respectively.

▶ **Corollary 17.** *$LTS_p \sqsubseteq_R LTS_g$ with $R = \{((q, \pi); q) \in \mathrm{Q}_p \times \mathrm{Q}_g\}$.*

Corollary 17 corresponds to a notion of correctness of the local planning semantics: any execution in the LPS corresponds to an execution in the standard semantics. In addition, if interactions are allowed to be planned with relative execution times of 0 (i.e. $h_{\min} = 0$) then timeless planning of interactions becomes possible. Thus, the planning semantics simulates the standards semantics in that case.

▶ **Corollary 18.** *$LTS_g \sqsubseteq_{R'} LTS_p$ with $R' = \{(q;(q,\pi)) \in \mathrm{Q}_g \times \mathrm{Q}_p \mid h_{\min} = 0\}$.*

However, this is no longer true in general if $h_{\min} > 0$ which means that not all execution sequences of the standard semantics are preserved by the local planning semantics.

▶ **Corollary 19.** *If LTS_g is zeno runs free then LTS_p is too.*

Corollary 19 states that the LPS does not introduce any zenoness behavior if the standard semantics is free from the latter. It is a direct consequence of Corollary 17 and the fact that it is not possible to have infinite sequences of planning transitions without interaction execution (γ is finite and planning times are bounded).

▶ **Proposition 20.** *If every reachable state of LTS_g is not a deadlock, then a reachable state of LTS_p is not deadlock if and only if it is not an action-time-lock.*

Proof of Propostion 20. We prove Proposition 20 by contradiction. Let us assume that the system under the standard (resp. local planning) semantics is deadlock free (resp. action-time-lock-free). Let (ℓ, v, π) be a reachable deadlock state of the LPS. We have:

$$\nexists \sigma \in \gamma \cup (\gamma \times \mathbb{R}_{\geq 0}), \exists \delta.\ (\ell, v, \pi) \overset{\sigma}{\rightsquigarrow}_{>\gamma} (\ell', v', \pi') \vee (\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{>\gamma} (\ell, v+\delta, \pi-\delta) \overset{\sigma}{\rightsquigarrow}_{>\gamma} (\ell', v', \pi')$$

We denote by $wait(\ell, v, \pi)$ the set of allowed waiting times at state (ℓ, v, π), that is:

$$wait(\ell, v, \pi) = \{0\} \cup \{\delta \in \mathbb{R}_{>0} | (\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{>\gamma} (\ell, v+\delta, \pi-\delta)\}$$

We also put $\max(wait(\ell, v, \pi))$ to denote the maximal waiting time at state (ℓ, v, π). Notice that $\max(wait(\ell, v, \pi))$ may not be defined in some cases. In fact, we are not interested in its actual existence but rather in the fact that it is bounded ($< +\infty$) or not.

▶ **Lemma 21.** *Let (ℓ, v, π) be a reachable state of the local planning semantics. For $k \in \mathbb{R}_{\geq 0}$, such that $k = \max(wait(\ell, v, \pi))$, we have the following properties:*
P1 *If $k < +\infty$ then $(\ell, v, \pi) \overset{k}{\rightsquigarrow}_{>\gamma} (\ell, v+k, \pi-k) \wedge wait(\ell, v+k, \pi-k) = \{0\}$*
P2 *If $\pi \neq \pi_0$ then $k \leq \min(\pi)$*

We distinguish 2 cases:

Case 1: no interaction is planned (i.e. $\pi = \pi_0$)

By definition of the LPS, it is clear that for $\pi = \pi_0$, there is no interaction to execute from (ℓ, v, π) or any of its successor $(\ell, v+\delta, \pi-\delta)$.

1. $wait(\ell, v, \pi) = \{0\}$:
   This means that time progress is not allowed at state (ℓ, v, π). We also have $\nexists \sigma \in (\gamma \times \mathbb{R}_{\geq 0}).(\ell, v, \pi) \overset{\sigma}{\rightsquigarrow}_{>\gamma} (\ell', v', \pi')$ (deadlock assumption). We can conclude that (ℓ, v, π) is a reachable action-time-lock state, which contradicts the assumption that the system under the local planning semantics is action-time-lock-free.
2. $wait(\ell, v, \pi) \neq \{0\}$:
   a. $\max(wait(\ell, v, \pi)) = +\infty$:

▶ **Lemma 22.** *Let (ℓ, v, π) be a reachable state of the local planning semantics . If $\forall \delta \in \mathbb{R}_{>0}.\ (\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{>\gamma} (\ell, v+\delta, \pi-\delta) \wedge \neg Plannable(\alpha)$ at (ℓ, v, π), then we have $\neg Enabled(\alpha)$ at $(\ell, v+\delta, \pi-\delta)$ with $\delta \geq h_{\min}$.*

By P1 of Lemma 21 we can deduce that $\exists \delta \geq h_{\min}$ such that $(\ell, v, \pi) \overset{\delta}{\rightsquigarrow}_{>\gamma} (\ell, v+\delta, \pi-\delta)$. We also have from the deadlock assumption and Lemma 22: $\bigwedge_{\alpha\in\gamma} \neg \mathit{Enabled}(\alpha)$. Finally, since the state $(\ell, v+\delta, \pi-\delta)$ is reachable in the standard semantics, and by evaluating the deadlock characterization 2 on state $(\ell, v+\delta, \pi-\delta)$, we can conclude that the system under the standard semantics deadlocks, which contradicts the assumption of deadlock freedom of the system under the standard semantics.

b. $\max(\mathit{wait}(\ell, v, \pi)) < +\infty$:
Considering that $k = \max(\mathit{wait}(\ell, v, \pi))$, then we have by P1 of Lemma 21: $(\ell, v, \pi) \overset{k}{\rightsquigarrow}_{>\gamma} (\ell, v+k, \pi-k) \wedge \mathit{wait}(\ell, v+k, \pi-k) = \{0\}$. Using the deadlock assumption we have: $\bigwedge_{\alpha\in\gamma} \neg \mathit{Plannable}(\alpha)$ at state $(\ell, v+k, \pi-k)$. Since the system cannot progress beyond this state $(\mathit{wait}(\ell, v+k, \pi-k) = \{0\})$, we can conclude that $(\ell, v+k, \pi-k)$ is a reachable action-time-lock state, which contradicts the assumption that the system under the local planning semantics is action-time-lock-free.

Case 2: at least an interaction is planned (i.e. $\pi \neq \pi_0$)

Considering that $k = \max(\mathit{wait}(\ell, v, \pi))$, since $\pi \neq +\infty$, we have by 2 of Lemma 21: $k < +\infty \wedge k \leq \min \pi$. Using the deadlock assumption we can infer that $k < \min \pi$, since no execution is possible from (ℓ, v, π) or any of its successors. This means that $(\ell, v+k, \pi-k)$ is a reachable action-time-lock state, which contradicts the assumption that the system under the LPS is action-time-lock-free. ◀

4 Enforcing Deadlock-Free Planning

As explained in previous section, the local planning semantics is based on local conditions for planning interactions and may exhibit deadlocks even when the system is deadlock-free with the standard semantics. Such deadlocks are partly due to the fact that planning an interaction may block, in addition to the participating components, extra components whose timing constraints are not considered by these local conditions. In this section, we investigate simple execution strategies that only restrict the horizon used for planning interactions with upper bounds. By reducing the period of time during which components are blocked, they tend to remove deadlocks from the reachable states. In what follows, we consider a composition of components $S = \gamma(B_1, \cdots, B_n)$ such that it is deadlock-free in the standard semantics.

▶ **Corollary 23** (Sufficient Condition for Deadlock-freedom). *If a reachable state of the planning semantics is not an action-time-lock then it is not deadlock.*

Corollary 23 is a direct consequence of Proposition 20. It affirms that for systems that are initially deadlock-free under the standard semantics, it is sufficient to prove action-time-lock-freedom of the LPS to prove its deadlock-freedom.

▶ **Proposition 24.** *A reachable state (ℓ, v, π) of the local planning semantics is an action-time-lock if and only if:*

$$\pi > 0 \ \wedge \bigwedge_{\alpha \notin conf(\pi)} \neg \mathit{Plannable}(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}).$$

The above proposition derives directly from the definition of action-time-locks on a state of the local planning semantics. As shown in Example 11, the local planning semantics may introduce deadlocks. The source of deadlocks is twofold: *(i)* due to communication delays, consecutive execution in a component are separated by at least $h_{\min}$ units of time which may be incompatible

with its timings constraints, and *(ii)* conditions for planning interactions are too permissive as they only take into account timing constraints of participating components whereas they may block additional components, namely the ones participating in conflicting interactions. In the rest of the paper, we study how to generate planning strategies for preserving deadlock-freedom by restricting the planning transitions of the LPS so that deadlock states become unreachable. Such a strategy may not exist when timing constraints cannot accommodate with the communication delays $h_{\min}$.

From Corollary 23, action-time-lock-freedom is a sufficient condition for deadlock-freedom of the LPS. By Proposition 24, a state (ℓ, v, π) is an action-time-lock in the local planning semantics if and only if no time progress is allowed nor planning or execution of interactions from (ℓ, v, π), that is:

$$\pi > 0 \ \wedge \bigwedge_{\alpha \in \gamma \setminus conf(\pi)} \neg Plannable(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \ \wedge (urg(\ell_i) + h_{\min}).$$

The above predicate characterizes the fact that no interaction can be executed or planned, nor time can progress in component $B_i \notin \mathrm{part}(\pi)$. Consequently, we deduce that a necessary condition of action-time-lock is the existence of a component $B_i \notin \mathrm{part}(\pi)$ such that time cannot progress in B_i and B_i cannot be planned in an interaction, that is:

$$\bigwedge_{\alpha \in \gamma(B_i) \setminus conf(\pi)} \Big(\neg Plannable(\alpha) \ \wedge \bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (urg(\ell_i) + h_{\min}) \Big).$$

where $\gamma(B_i)$ denotes the subset of interactions in which B_i participates, that is, $\gamma(B_i) = \{\beta \in \gamma \mid B_i \in \mathrm{part}(\beta)\}$. Notice that the above expression strongly depends on the plan π, which is difficult to characterize in practice. The following theorem proposes sufficient plan-independent condition characterizing action-time-lock states of the LPS .

▶ **Theorem 25.** *Let ϕ be the following predicate:*

$$\bigvee_{1 \le i \le n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (urg(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg Plannable(\alpha) \ \vee \bigvee_{\substack{\beta \in conf(\alpha) \\ B_i \notin \mathrm{part}(\beta)}} \widetilde{Plannable(\beta)} \Big) \Big].$$

where $\widetilde{Plannable(\beta)}$ is the predicate defined as follows:

$$\widetilde{Plannable(\beta)} \Leftrightarrow \exists \delta > 0 \ . \ \delta \le h_{\max}(\beta) \ \wedge \ Plannable(\beta, \delta).$$

We prove that a reachable action-time-lock state (ℓ, v, π) satisfies ϕ.

Proof of Theorem 25. A reachable action-time-lock state of the LPS satisfies:

$$\pi > 0 \ \wedge \bigwedge_{\alpha \in \gamma(B_i) \setminus conf(\pi)} \Big(\neg Plannable(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \mathrm{part}(\pi)}} \mathsf{at}(\ell_i) \ \wedge (urg(\ell_i) + h_{\min}) \Big).$$

In order to approximate the above formula, we distinguish two cases:

Case 1: no interaction is planned (i.e. $\pi = \pi_0$)

From $\pi = +\infty$ we deduce directly that there exists an urgent component B_i such that no interaction α involving B_i can be planned, that is:

$$\bigvee_{1 \le i \le n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (urg(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \neg Plannable(\alpha) \Big]. \tag{1}$$

Case 2: at least an interaction is planned (i.e. $\pi \neq \pi_0$)

In this case, there exists an urgent component $B_i \notin \text{part}(\pi)$ such that no interaction α involving B_i can be planned, either because it conflicts with a planned interaction β ($0 < \pi(\beta) < +\infty$) or because $\mathit{Plannable}(\alpha)$ is not satisfied, that is $\exists \beta \in \pi, \exists B_i \notin \text{part}(\beta)$ satisfying:

$$(0 < \pi(\beta) < +\infty) \ \wedge \bigwedge_{\alpha \in \gamma(B_i) \setminus conf(\beta)} \neg \mathit{Plannable}(\alpha) \ \wedge \bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \text{part}(\beta)}} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}).$$

or equivalently $\exists \beta \in \pi, \exists B_i \notin \text{part}(\beta)$ satisfying:

$$\bigvee_{\substack{\ell_i \in \mathcal{L}_i \\ B_i \notin \text{part}(\beta)}} \mathsf{at}(\ell_i) \wedge (\mathit{urg}(\ell_i) + h_{\min}) \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \vee \big(\beta \in \mathit{conf}(\alpha) \wedge (0 < \pi(\beta) < +\infty)\big)\Big).$$

By noticing that we have the following implication between quantifiers $\exists y, \forall x. Q(x,y) \implies \forall x, \exists y. Q(x,y)$, we can deduce that the above condition implies:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \ \vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \text{part}(\beta)}} 0 < \pi(\beta) < +\infty \Big) \Big].$$

As $\pi > 0$, and if we consider only reachable action-time-locks, we have $0 < \pi(\beta) \leq h_{\max}(\beta)$, and by Lemma 12 we have $\mathit{Plannable}(\beta, \pi(\beta))$. That is, β satisfies $\mathit{Plannable}(\beta)$ in which the lower bound $h_{\min}$ is replaced by the strict lower bound 0, i.e. $\widetilde{\mathit{Plannable}(\beta)}$. Then, the above expression becomes:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \ \vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \text{part}(\beta)}} \widetilde{\mathit{Plannable}(\beta)} \Big) \Big]. \quad (2)$$

By remarking that Expression 1 implies Expression 2, we can conclude that an action-time-lock of the local planning semantics satisfies:

$$\bigvee_{1 \leq i \leq n} \Big[\bigvee_{\ell_i \in \mathcal{L}_i} \mathsf{at}(\ell_i) \ \wedge (\mathit{urg}(\ell_i) + h_{\min}) \ \wedge \bigwedge_{\alpha \in \gamma(B_i)} \Big(\neg \mathit{Plannable}(\alpha) \ \vee \bigvee_{\substack{\beta \in \mathit{conf}(\alpha) \\ B_i \notin \text{part}(\beta)}} \widetilde{\mathit{Plannable}(\beta)} \Big) \Big]. \ \blacktriangleleft$$

Notice that due to the monotony of ϕ on upper bound horizons, we obtain the following lemma:

▶ **Lemma 26.** *If LTS_p is action-time-lock free for the upper bound horizons function $h_{\max}$, then it is action-time-lock free for any upper bound horizon function $h'_{\max} \leq h_{\max}$.*

In order to attest that planning interactions does not introduce deadlocks, we use an SMT solver to check the satisfiability of ϕ. As explained earlier, a given system is deadlock-free under the restricted LPS if $\mathit{Reach}(LTS_p) \wedge \phi$ is unsatisfiable. Since $\mathit{Reach}(LTS_p) \subseteq \mathit{Reach}(LTS_g)$ (Corollary 17), we can verify the above on $\mathit{Reach}(LTS_g)$. Effectively, we do not compute $\mathit{Reach}(LTS_g)$ to avoid the combinatorial explosion problem, inherent to composition of timed automata. In fact, we rather build an over-approximation, $\widetilde{\mathit{Reach}(LTS_g)}$, of the latter, and use it during our verification. Finding a strategy granting action-time-lock-freedom is based on the idea of restricting the upper bound horizon function $h_{\max}$. In fact, since $h_{\min}$ is a parameter that is dependent of the communication latency of a given execution platform, it cannot be tuned. Instead, initially for each interaction $\alpha \in \gamma$, $h_{\max}(\alpha) = +\infty$. Thereafter, due to the monotony of ϕ (Lemma 26) on upper horizons, this parameter will be refined, that is, its maximum will be decreased until finding a function $h_{\max}$ for which $\widetilde{\mathit{Reach}(LTS_g)} \wedge \phi$ is unsatisfiable or until reaching the upper horizon function $h^{h_{\min}}_{\max}$ for which $h_{\max}(\alpha) = h_{\min}$ for every $\alpha \in \gamma$ and such that $\widetilde{\mathit{Reach}(LTS_g)} \wedge \phi$ is satisfiable.

5 Planning Semantics as Real-Time Controller Synthesis

In Section 5, we presented a method that provides execution strategies by restricting the upper bounds planning horizon for each interaction. This strategy aims to preserve the deadlock-freedom property of a given system under the local planning semantics without imposing further scheduling constraints. This approach relies on the verification of a given expression on over-approximations of the reachable states of the initial semantics. Thus, it may give false-positive results due to *(i)* the nature of expression to check (sufficient condition) and *(ii)* the over-approximation of the reachable states of the LPS using over-approximations of the reachable states of the standard semantics (Corollary 17).

In such cases, an alternative is to tackle the problem as a real-time controller synthesis problem. Real-time controller synthesis is a common method used to extract an execution strategy from formal specifications satisfying certain properties. Usually, these properties express the reachability (resp. non-reachability) of a set of winning states (resp. bad states). In case of planning interactions with bounded horizons, the idea is to restrict the transition relation so that all the remaining behaviors do not lead to states where a component is urgent and no possible execution including this component may occur. This can be formalized as a reachability game for a timed game automaton [12], where the main idea consists in trying to find an execution strategy guaranteeing that a given set of namely *bad states* of the system are never reached.

In order to apply this approach, it is required to encode the planning of interactions and their effects on the system, that is, *(i)* encode interactions planning as synchronizations between components, *(ii)* reserve the components of the planned interactions until their chosen execution date, i.e, keep track of the planned interactions and their execution dates, and *(iii)* characterize the set of bad states. Thereafter, tools such as UPPAAL-Tiga [6] can be used to find an execution strategy of the planning semantics avoiding the set of bad states, that is, deadlock states. Expressing the planning problem as a real-time controller synthesis problem is not an easy task. Hereinafter, we discuss the different issues met during the formalization process and provide suggestions for solving them.

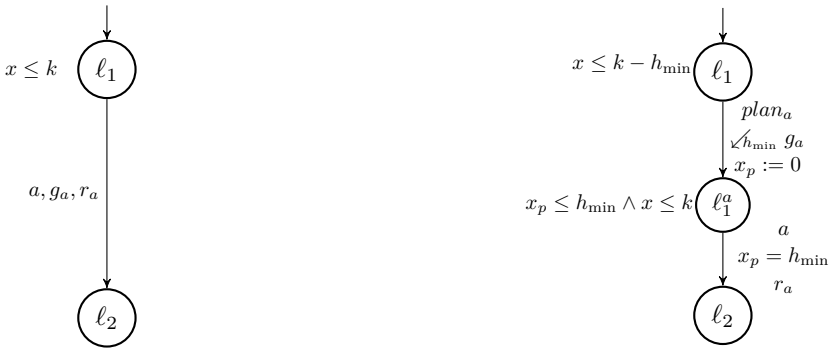
5.1 Planning Zones

From expression 4, we can see that the clocks values for planning an interaction α are calculated at a global level, that is, by applying the $\swarrow_{h_{\min}}^{h_{\max}(\alpha)}$ on the conjunction of its participating actions timing constraints. Notice that for a timing constraint $g = g_1 \wedge g_2$, we have:

$$\swarrow_{h_{\min}}^{h_{\min}} g = \swarrow_{h_{\min}}^{h_{\min}} (g_1 \wedge g_2) \implies \swarrow_{h_{\min}}^{h_{\min}} g_1 \wedge \swarrow_{h_{\min}}^{h_{\min}} g_2 \tag{5}$$

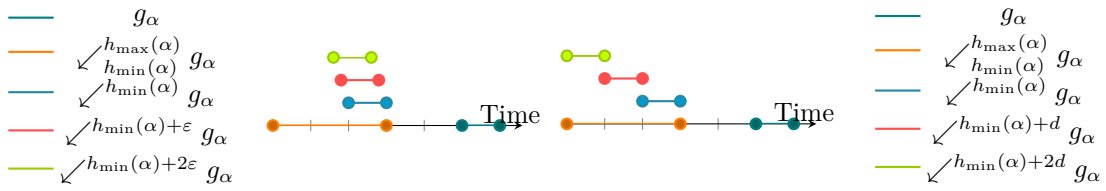
The above expression bears out the fact that planning states must be encoded on the composition of the system model and not on individual components. Particularly, expression 5 points out the fact that encoding the planning on transitions of individual components will induce additional behavior ($\swarrow_{h_{\min}}^{h_{\min}} (g_1 \wedge g_2) \implies \swarrow_{h_{\min}}^{h_{\min}} g_1 \wedge \swarrow_{h_{\min}}^{h_{\min}} g_2$). This represents the first drawback of this method since building the composition may be tedious especially for big scale systems. Therefore, a simple solution to avoid computing the composition is to consider models with interactions having timing constraints on up to one of their participating actions, that is, given an interaction $\alpha = \{a_i\}_{i \in I} \in \gamma$, we have $g_\alpha = true$ or $g_\alpha = g_{a_i}$, with $g_{a_j} = true$ for $j \in I, j \neq i$. In fact, considering interactions including up to one action with timing constraints, will allow to encode the planning on individual components that, additionally to the defined synchronizations (interactions), will also synchronize their planning actions.

The idea is to split each transition of the initial model into two transitions: *(1)* a planning transition, followed by *(2)* an execution transition after the plan transition being performed.



(a) Part of a timed automaton. **(b)** Planning encoding.

Figure 2 Planning as a Timed Automaton.



An interaction guard g_α and its planning intervals.

Discretized planning intervals for g_α.

Figure 3 Discretizing Planning Horizons for Interaction.

For an interaction $\alpha \in \gamma$, the choice of the planning horizon, that is, the duration for which components participating in α will be blocked for until their execution, will be encoded on the execution transition of the component whose action $a_i \in \alpha$ and $g_\alpha = g_{a_i}$. Otherwise, if $g_\alpha = true$ this choice is made arbitrarily. Consequently, this component will be equipped with a clock x_p that will be used to track the planning dates. Finally, time progress conditions must also be translated to enforce planning at the latest $h_{\min}$ units of time before their expiry. Figure 2 depicts an overview of such transformation for $\delta = h_{\min}$ horizon:

5.2 Infinite Planning Transitions

Effectively, in order to encode the planning in timed automata, horizons values must be integer. Moreover, due to the dense time nature of the planning intervals (relative planning date for each interactionα are in $[h_{\min}, h_{\max}(\alpha)]$), we end up with an infinity of plan transitions, especially when not restricted upper bound planning horizons, i.e., $h_{\max} = h^{\infty}_{\max}$. Consequently, the first thing to do is to restrict for each interaction $\alpha \in \gamma$ the upper bound planning horizon $h_{\max}(\alpha)$.Thereafter, we propose to discretize the planning horizons in order to obtain finite values in $\mathbb{Z}_{>0}$ (Figure 3). In what follows, we denote by $Disc : \gamma \longrightarrow \mathcal{D}$ the discretized horizon function defining for each interaction its respective discretized planning horizons $\mathcal{D} \subset \mathbb{Z}_{>0}$.

▶ **Definition 27** (Planning Timed Automaton)**.** Given n timed components $\mathcal{B}_i = (\mathcal{L}_i, \ell_0^i, \mathcal{A}_i, \mathcal{T}_i, \mathcal{X}_i, \mathcal{I}_i)$ synchronizing through the interaction set γ such that, for each interaction $\alpha \in \gamma$, the guard of α is equal to the guard of one of its included actions. We define the corresponding planning model as the composition of the n timed automata $\mathcal{B}_i^p = (\mathcal{L}_i^p, \ell_0, \mathcal{A}_i \cup \mathcal{P}_i, \mathcal{T}_i^p, \mathcal{X}_i \cup \{x_i^p\}, \mathcal{I}_i^p)$, w.r.t the

interaction set $\gamma \cup \mathcal{P}$, where:

- $\mathcal{P}_i = \cup_{a \in \mathcal{A}_i}\ p_a$ is the set of *Planning Actions*
- $\mathcal{P} = \{p_\alpha = \{p_{a_i}\}_{i \in I} | \alpha \in \gamma \wedge \alpha = \{a_i\}_{i \in I}\}$ is the set of *Planning Interactions*
- x_i^p is a *Tracking Clock* for interactions execution in each component
- $\mathcal{L}_i^p = (\mathcal{L}_i \cup \mathcal{L}_{i_p})$ is the set of control locations, where $\mathcal{L}_{i_p}$ is the set of locations following planning actions
- $\mathcal{T}_i^p$ is such that for each $(\ell_i, a_i, g_i, r_i, \ell_i') \in \mathcal{T}_i$, $a_i \in \alpha$ and for each $\delta \in Disc(\alpha)$:
  - if $g_\alpha \neq true$ we have:

    Planning transitions: $\begin{cases} \ell_i \xrightarrow{p_{a_i}, true, \emptyset} \ell_{a_i}, \text{ if } g = true \\ \ell_i \xrightarrow{p_{a_i}, \swarrow^\delta g_i, r(x_i^p)} \ell_{a_i}^\delta, \text{ otherwise} \end{cases}$

    Execution transitions: $\begin{cases} \ell_{a_i} \xrightarrow{a, true, r_i} \ell_i', \text{ if } g = true \\ \ell_{a_i}^\delta \xrightarrow{a, g_a \wedge x_i^p = \delta, r_i} \ell_i', \text{ otherwise} \end{cases}$

    where $\ell_a, \ell_{a_i}^\delta \in \mathcal{L}_{i_p}$.

  - if $g_\alpha = true$, we choose one action $b \in \alpha$:

    Planning transitions: $\begin{cases} \ell_i \xrightarrow{p_{a_i}, true, \emptyset} \ell_{a_i}, \text{ if } a \neq b \\ \ell_i \xrightarrow{p_{a_i}, true, r(x_i^p)} \ell_{a_i}^\delta, \text{ otherwise} \end{cases}$

    Execution transitions: $\begin{cases} \ell_{a_i} \xrightarrow{a_i, true, r_i} \ell_i', \text{ if } a \neq b \\ \ell_{a_i}^\delta \xrightarrow{a_i, g_i \wedge x_i^p = \delta, r_i} \ell_i', \text{ otherwise} \end{cases}$

- $\mathcal{I}_i^p$ is the set of *Location Invariants* , such that $\forall \ell_i^p \in \mathcal{L}_i^p$, we have:

  $\mathcal{I}_i^p(\ell_i^p) = \begin{cases} \mathsf{tpc}(\ell_i) - h_{\min}, \text{ if } \ell_i^p = \ell_i \in \mathcal{L}_i \\ x_i^p \leq \delta \wedge \mathsf{tpc}(\ell_i), \text{ if } \ell_i^p = \ell_{a_i}^\delta \in \mathcal{L}_{i_p} \text{ such that } \ell_i \in \mathcal{L}_i \wedge \ell_i \xrightarrow{p_{a_i}} \ell_{a_i}^\delta, \end{cases}$

For a composition $\gamma(B_1, \cdots, B_n)$, let $LTS_{p'} = (\mathrm{Q}_{p'}, \gamma' \cup \mathbb{R}_{>0}, \longrightarrow_{\gamma'})$, where $\gamma' = \gamma \cup \mathcal{P}$, be the corresponding labeled transition system of its planning model under the standard semantics.

▶ **Theorem 28.** *$LTS_{p'} \sqsubseteq_{R'} LTS_g$ where R' is the relation defined as follows: For $q^p = (\ell^p, v^p) \in \mathrm{Q}_{p'}$ and $q^g = (\ell^g, v^g) \in \mathrm{Q}_g$, such that $(q^p, q^g) \in R'$, we have:*

- *$\ell^p = (\ell_1^p, \cdots, \ell_n^p)$, $\ell_g = (\ell_1^g, \cdots, \ell_n^g)$:*

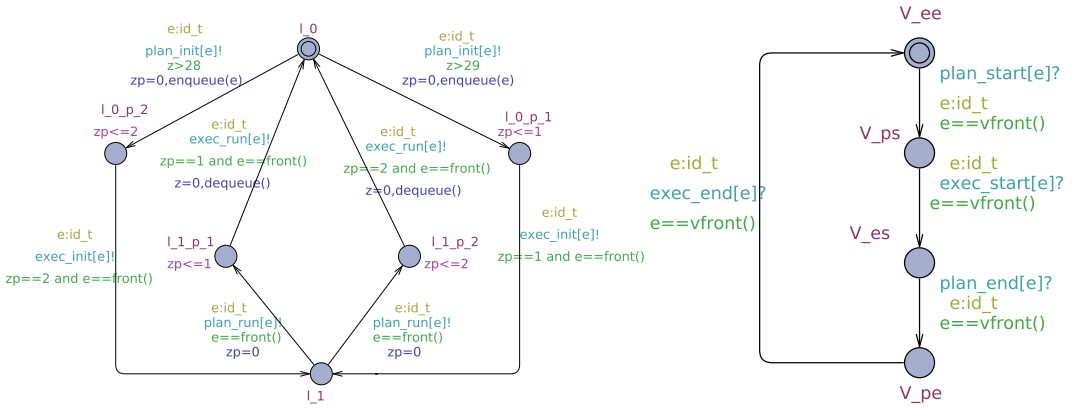
$$\forall i \in \{1, \cdots, n\},\ \ell_i^g = \begin{cases} \ell_i^p, \text{ if } \ell_i^p \in \mathcal{L}_i, \\ \ell_i, \text{ if } \ell_i^p \in \mathcal{L}_{i_p} \text{ with } \ell_i \xrightarrow{a,g,r} \ell_i^p \in \mathcal{T}_i^p \wedge \ell_i \in \mathcal{L}_i, \end{cases}$$

  Notice that for the case where $\ell_i^p \in \mathcal{L}_{i_p}$, ℓ_i is unique by construction of the planning model.
- *$v^g = equ(v^p)$, where $equ(v^p)$ is the projection of v^p on clocks of v^g*

Proof of Theorem 28. To prove that $LTS_{p'} \sqsubseteq_{R'} LTS_g$, we need to prove that:

1. $\forall (q^p, q^g) \in R', \sigma \in \gamma \cup \mathbb{R}_{>0}$ such that $q^p \xrightarrow{\sigma}_{\gamma'} q'^p \Rightarrow \exists q'^g . (q'^p, q'^g) \in R' \wedge q^g \xrightarrow{\sigma}_{\gamma} q'^g$
2. $\forall (q^p, q^g) \in R', p_\alpha \in \mathcal{P}$ such that $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p \Rightarrow (q'^p, q^g) \in R'$

1. a. Suppose that $(q^p, q^g) \in R', \sigma = \alpha \in \gamma$ and $q^p \xrightarrow{\alpha}_{\gamma'} q'^p$ with $q'^p = ((\ell'^p_1, \cdots, \ell'^p_n), v'^p)$. We have: $q^p \xrightarrow{\alpha}_{\gamma'} q'^p \Rightarrow g_\alpha$ is *true*, and for $\alpha = \{a_i\}_{i \in \mathcal{I}}$, by construction of the planning automaton, we have: $\ell_i^g \xrightarrow{a_i, g_i, r_i} \ell'^g_i$ such that $\ell'^g_i = \ell'^p_i$. Moreover, since the same clocks are reset by the execution of α in both models, we deduce that $v'^g = equ(v'^p)$. By remarking that the state of components not participating in α remains the same, we conclude that $\exists q'^g$ such that $q^g \xrightarrow{\alpha}_{\gamma} q'^g \wedge (q'^p, q'^g) \in R'$.



(a) Controller Component. **(b)** Variable Component.

(c) Task Component.

Figure 4 Planning Automata for the Task Manager Example.

b. Suppose that $(q^p, q^g) \in R', \sigma \in \mathbb{R}_{>0}$ and $q^p \xrightarrow{\sigma}_{\gamma'} q'^p$. For $q_i^p = (\ell_i^p, v_i^p)$, we define $\mathcal{I}_g$ the set of indexes such that $\ell_i^p \in \mathcal{L}_i$, and $\mathcal{I}_p$ the set of indexes such that $\ell_i^p \in \mathcal{L}_{p_i}$.
- $\forall i \in \mathcal{I}_g . \ell_i^p = \ell_i^g \wedge q_i^p \xrightarrow{\sigma} q'^p_i \Rightarrow q_i^g \xrightarrow{\sigma} q'^g_i$. This implication is a direct result of the planning model definition since: $\sigma \leq \mathcal{I}(\ell_i^p) \leq \mathsf{tpc}(\ell_i^g) - h_{\min}$.
- $\forall i \in \mathcal{I}_p . \ell_i^g = \ell_i$ such that $\ell_i^p \in \mathcal{L}_{i_p}$ with $\ell_i \xrightarrow{a,g,r} \ell_i^p \in \mathcal{T}_i^p \wedge \ell_i \in \mathcal{L}_i$. Thus $q_i^p \xrightarrow{\sigma} q'^p_i \Rightarrow q_i^g \xrightarrow{\sigma} q'^g_i$, since $\mathcal{I}(\ell_i^p) \implies \mathsf{tpc}(\ell_i^g)$.

We conclude that $\exists q'^g$ such that $q^g \xrightarrow{\sigma}_{\gamma} q'^g \wedge (q'^p, q'^g) \in R'$.

2. Suppose that $(q^p, q^g) \in R'$ and $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p$, with $p_\alpha \in \mathcal{P}$ and $q'^p = ((\ell'^p_1, \cdots, \ell'^p_n), v'^p)$. We have: $q^p \xrightarrow{p_\alpha}_{\gamma'} q'^p \Rightarrow$ for $\alpha = \{a_i\}_{i \in \mathcal{I}}$ $\ell_i^g = \ell_i^p \wedge \ell_i^g \xrightarrow{p_{a_i}} \ell'^p_i$. Moreover, since planning actions reset only the clocks x_i^p for tracking execution time, we can deduce that $(q'^p, q^g) \in R'$. ◀

Once interactions planning encoded, one last thing to do is to add the set of bad states to each planning automaton (if needed) and find a strategy to avoid those states. Figure 4 depicts the corresponding planning automata for example of Figure 1 with respect to Definition 27. Locations suffixed by p, correspond to locations following planning actions, whereas locations ending with *err* define the bad states, that is, states with urgent time progress condition(s) and no possible execution removing the urgency. In this example, for each interaction $\alpha \in \gamma$, we chose $\mathcal{D}(\alpha) = \{1, 2\}$. Notice that for this example, we consider that all actions are controllable actions

```
State:  ( Controller.l_1_p_1 Task(0).l_0 Task(1).l_1_p Task(2).l_3_p2
Task(3).l_0 Task(4).l_0 Task(5).l_0 Task(6).l_0 Task(7).l_0 Task(8).l_0
Task(9).l_0 Task(10).l_0 Task(11).l_0 Task(12).l_0 Task(13).l_0 Task(14).l_0
Task(15).l_0 Task(16).l_0 Task(17).l_0 Task(18).l_0 Task(19).l_0 Var.V_pe
) vlist[0]=2 vlist[1]=0 vlist[2]=0 vlist[3]=0 vlist[4]=0 vlist[5]=0
vlist[6]=0 vlist[7]=0 vlist[8]=0 vlist[9]=0 vlist[10]=0 vlist[11]=0
vlist[12]=0 vlist[13]=0 vlist[14]=0 vlist[15]=0 vlist[16]=0 vlist[17]=0
vlist[18]=0 vlist[19]=0 vlen=1 Controller.list[0]=1 Controller.list[1]=0
Controller.list[2]=0 Controller.list[3]=0 Controller.list[4]=0
Controller.list[5]=0 Controller.list[6]=0 Controller.list[7]=0
Controller.list[8]=0 Controller.list[9]=0 Controller.list[10]=0
Controller.list[11]=0 Controller.list[12]=0 Controller.list[13]=0
Controller.list[14]=0 Controller.list[15]=0 Controller.list[16]=0
Controller.list[17]=0 Controller.list[18]=0 Controller.list[19]=0
Controller.len=1
When you are in (Controller.zp==1 && Task(2).yp<=2), take transition
Controller.l_1_p_1-> Controller.l_0 { zp == 1 && 1 == front(), exec_run[1]!,
z := 0, dequeue() } Task(1).l_1_p->Task(1).l_2 { 1, exec_run[id]?, y := 0,
venqueue(id) } When you are in (Task(2).yp==2 && Controller.zp<=1), take
transition Task(2).l_3_p2-> Task(2).l_0 { yp == 2, exec_end[id]!, vdequeue()
} Var.V_pe->Var.V_ee { 2 == vfront(), exec_end[2]?, 1 }
```

Figure 5 Sample of the Output Strategy from UPPAAL-Tiga.

since it is a closed system in the sense that there is no interaction with the environment.

We performed the verification on the Task Manager examples with 20 tasks. The winning condition being a safety condition: avoid all *"err"* locations. This was translated into the following property:

$$\text{control: A[ ] forall (i : int[0,N-1]) not (Task(i).l\_2\_err or Task(i).l\_3\_err)} \tag{6}$$

The property of interest was successfully verified. Additionally, we were also able to synthesize all wining actions of all states using the command line of UPPAAL-Tiga. A sample of the resulting output is provided below Figures 5. Notice that the average execution time[1] for verifying Property 6 is 0.1141 seconds (0.6534 seconds when requesting the generation of a strategy).

5.3 Discussion

In this section, we explained how the problem of planning interactions can be formalized into a real-time controller synthesis approach. However, this approach has some drawbacks. In order to encode planning of interactions in components as timed automata, this approach restricts its scope to discretized horizon values which results in having less control over the planning dates of interactions, and leads in case of a high number of discretized values, to an explosion in the number of planning transitions. Unfortunately, we do not have an immediate solution for this problem. In fact, it is user dependent since one user may just want to have a ASAP execution for a given interaction, for instance because the components involved in this interaction are often

[1] The experiments have been conducted on a HP machine with Ubuntu 16.04, an Intel® Core™ i5-4300U processor of frequency 1.90GHz×4, and 7.7GiB memory.

requested, and in that case the practice will be to always plan with $h_{\min}$. In other cases, the user may want to plan an interaction with flexible amount time. Additionally, this approach considers only a class of systems where interactions have timing constraints on up to one of their participating components action. Otherwise, the planning should be encoded on the composition, which represents a tedious work because of the state space explosion problem. Nevertheless, this approach differs form the usual scheduler synthesis approach since it is not performed on the regular semantics of timed automata. Particularly, here we are interested in avoiding bad states of the planning semantics (states that verify the expression of Theorem 25). Consequently, unless finding an automatic general method for generating such complex expression in the query language accepted by such tools, and without ignoring that finding a strategy avoiding those states may be hard in term of computational complexity, our real-time controller synthesis approach seems more straightforward and much simpler but it comes with some feasibility restriction.

6 Implementation and Experiments

6.1 Implementation

For our experiments, we used the BIP framework [5] as a modeling language to define systems and their synchronizations. BIP (Behavior, Interaction, Priority), is a component-based framework with a rigorous semantics that allows to model systems as a set of atomic components coordinating their behaviors through multiparty interactions. The BIP framework provides a rich set of tools that allows to model, verify and execute systems. The BIP toolbox is structured in different categories (see Figure 6):

1. This category includes *translation of various language or modeling paradigm* that allows the automatic generation of BIP models as well as the front-end of the BIP compiler.
2. The middle-end of the BIP compiler consists of several modules that allows model transformation (from BIP to BIP) and performance optimization (flattening and distributed real-time filter). Particularly, the distributed real-time filter provides an intermediate model transformation (Send/Receive transformation [29]) that aims to reduce the gap between high level models and their actual implementations. Additionally, in association with the RTD-Finder tool it provides analyses allowing performance evaluation [15] as well as the actual analyses for the approach presented in Section 4. Note that the identity filter is the default filter that given a BIP model return the same BIP model.
3. The BIP back-end consists of code generator that generates the actual C++ corresponding to the actual BIP model yield by the middle-end. The engine based code generator produces simulation code that incorporates the BIP simulation engine. The Distributed code generator generates form Send/Receive models C++ code for distributed platform.
4. Verification of safety properties or properties allowing to tune a given system in order to obtain better performances are achieved using the RTD-Finder [27] and the SBIP tools [26].

The proposed method has been implemented in the distributed real-time middle-end filter of the BIP compiler. It aiming to generate information that could be used by the back-end during the code generation phase. The presented approach requires a substantial knowledge of the system model, since the satisfiability verification of $\overbrace{Reach(LTS_g)} \wedge \phi$, needs a deep analysis of the system, in order to generate the predicates used in the latter. The implementation takes as input a BIP model and a horizon file specifying at least the lower bound horizon, that is, $h_{\min}$. Since we do not have a concrete execution platform, the choice of $h_{\min}$ was done relatively to the timing constraints of the verified model, that is, we chose values of $h_{\min}$ that are always smaller than upper bounds of any timing constraints appearing in the components of a given system. This choice was motivated by the fact that deploying a system with timing constraints being of the same order of magnitude than the communication delays of the target platform is unlikely to

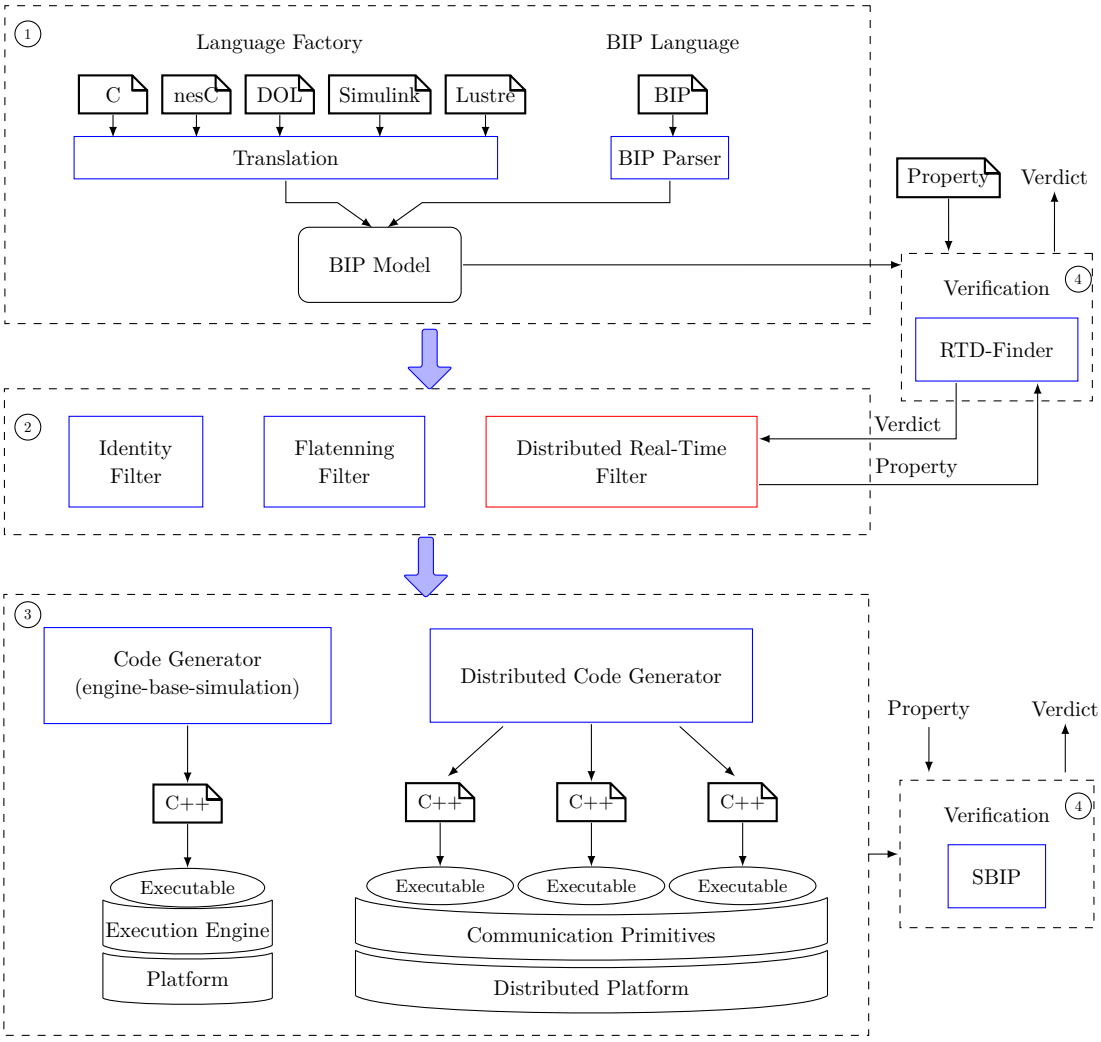


Figure 6 The BIP Toolbox.

happen. First, the front-end of the BIP compiler creates an abstract representation of the latter. Thereafter, the distributed real-time filter performs a model analysis in order to construct the predicates needed in ϕ, while keeping interactions upper horizons as free variables. In order to ease the verification process and remove the back and forth process between the predicates generation and their verification, we fully integrated the generation of the compositional invariants used by the RTD-Finder tool in the middle-end. Finally, a Yices [17] file including system invariants and the predicates approximating action-time-locks, is generated. The Yices solver checks then the satisfiability of $\widetilde{Reach}(LTS_g) \wedge \phi$. If the result is unsatisfiable, then planning does not miss the deadlines expressed by components time progress conditions. Otherwise, if the result is satisfiable, Yices generates a counter-example. Since for each interaction $h_{\min}$ is a fixed value, and due to the monotony of ϕ on $h_{\max}$, the generated counter-example is used to find the maximal value of $h_{\max}$ guaranteeing action-time-lock-freedom, and thus deadlock-freedom of the planning semantics. This part is however done manually, and based on binary search algorithm.

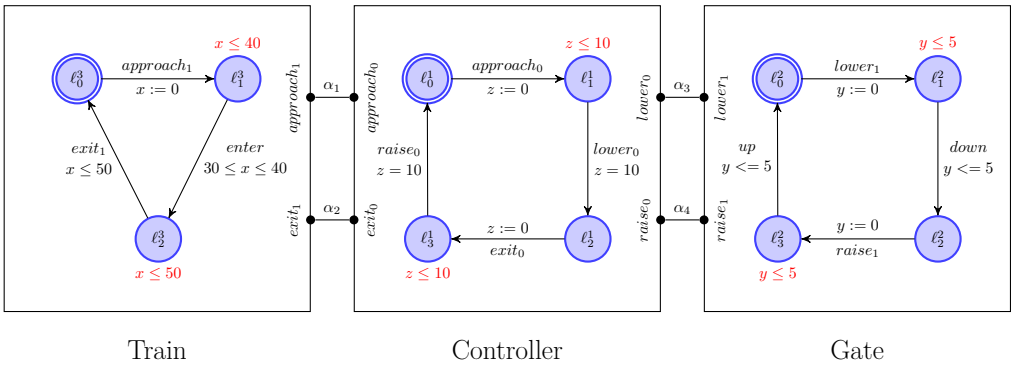


Figure 7 Train Gate Controller.

6.2 Benchmarks

For the experiments, we chose three additional benchmarks:

Train Gate Controller

The train gate controller [4] is a system composed of: a controller, a gate and a train. Figure 7 gives an overview of the system and its interactions: The train informs the controller about his position (w.r.t. to the crossing) through the interactions α_1 (approach) and α_2 (exit). On the other hand, the controller lowers (α_3) and raises (α_4) the gate whenever the train enters, respectively exits. Notice that actions $\{enter\}$ of the train, and $\{up, down\}$ of the gates are considered as singleton interactions.

Firewire

The IEEE 1394 root contention protocol (firewire) [14] is a standard protocol for interconnecting multimedia devices. It describes a serial bus used to transport digitized video and audio signals in a network of multimedia equipments. Among the different protocols used in this system, we put our interest in the leader election protocol called *tree identify protocol*. In this model, we consider two nodes (devices) and their respective channels. In order to elect a leader, each node sends a request via its respective channel asking its neighbor to be *a parent*. Once a neighbor receives a parent request, it either sends an acknowledgment or detects a *contention* in the case where it also sent a parent request. This contention is solved by assigning waiting times before the next send requests. Figure 8 depicts the model for the node component.

Gear Controller

The gear controller system describes the control system responsible for the gear change inside a vehicle. The used model encompasses formal models of the gear controller and its environment. The whole system includes five components: an interface, a controller, a clutch, an engine a gear-box and two global variables. In order to change the gear, the interface sends a signal to the controller. Consequently, the controller interacts with the engine, the clutch and the gear-box to achieve the gear change. The engine is responsible of either regulating the torque or synchronizing the speed. On the other hand, the gear-box sets the gear between some fixed bounds, whereas, the clutch is used whenever the engine is not able to function properly (under difficult driving conditions, for instance). The case study was initially designed by UPPAAL [25] and has been translated to BIP.

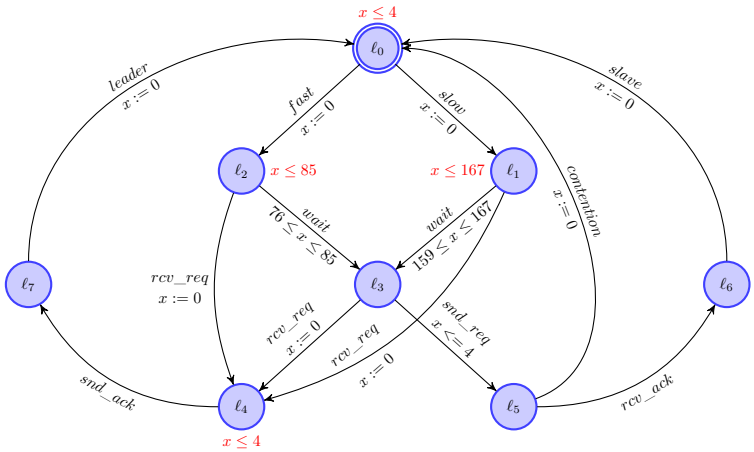


Figure 8 Timed Automaton for a Node.

Table 1 Detailed Results of the Task Manager Experiments.

$h_{\min}$	$h_{\max}(\alpha_1)$, $h_{\max}(\alpha_2)$	$h_{\max}(\alpha_3)$, $h_{\max}(\alpha_4)$	$h_{\max}(\alpha_5)$, $h_{\max}(\alpha_6)$	$h_{\max}(\alpha_7)$, $h_{\max}(\alpha_8)$
4	$+\infty$	$+\infty$	9	$+\infty$
3	$+\infty$	$+\infty$	8	$+\infty$
2	$+\infty$	$+\infty$	7	$+\infty$
1	$+\infty$	$+\infty$	6	$+\infty$

6.3 Results

Table 1 depicts the values $h_{\max}$ for each interaction of the running example, obtained while fixing $h_{\min}$. Notice that the symmetry of the system implies the same $h_{\max}$ for interactions $\alpha_i, \alpha_{i+1}, i \in \{1, 3, 5, 7\}$. By remarking that location ℓ_3^2 (resp. ℓ_3^3) has a time progress condition $x \leq 4$ (resp. $y \leq 4$), and by observing that the clock x is reset on the transition leading to this location, we can conclude that planning the system with $h_{\min} > 4$ will lead to an action-time-lock. Particularly, in Example 11, for $h_{\min} = 2$ interaction α_6 was planned with a horizon $\delta = 8$, and consequently, leads to a action-time-lock state. Our method detects such cases and thus, finds that the maximum horizon for this interaction is 7. Likewise, the $h_{\max}$ for interactions α_2, α_4 and α_8 (resp. α_1, α_3 and α_7) is found to be unbounded ($+\infty$).

Table 2 summarizes the experiments obtained on the benchmarks stated above, where n is the number of components, nb_{tpc} the number of time progress conditions that will be verified against action-time-lock freedom and $\max h_{\min}$ the maximum value of $h_{\min}$ for which the system is action-time-lock-free in the planning semantics. Additionally, the column $h_{\max}$ indicates whether a restriction on the upper horizons is required to avoid deadlocks. Finally, t_{exec} gives an overview of the execution time including both the invariants generation and the verification time.

As shown in table 1, the task manager model has a maximal $h_{\min}$ value of 4 TU and requires a restriction on the upper horizons for interactions α_5 and α_6. In the same way, we found that the train gate controller, the firewire and the train gate controller models have respectively maximal $h_{\min}$ value of 4 TU, 5 TU and 130 TU. However, they do not require any restriction on the upper horizons values of their interactions.

Table 2 Experiments Results.

Model	n	nb_{tpc}	$\max h_{\min}$	$h_{\max}$	$t_{exec}(s)$
Task Manager	4	4	4	B	0.11
Train Gate Controller	3	6	4	$+\infty$	0.16
Firewire	4	10	5	$+\infty$	3.03
Gear Controller	5	19	130	$+\infty$	4.65

7 Related Work

Timed automata are high-level representations which are useful for modeling, specifying and analyzing system behavior [4]. They rely on mathematical abstractions such as real-valued clocks, instantaneous executions and communications, which are no longer valid at implementation level. Following model-based design approaches, a valid question is how to derive implementations from timed automata? This problem has already been addressed for centralized execution platforms. More specifically, Abdellatif et al. [1] shows how to take into account execution times and provides sufficient conditions for an implementation to be robust with respect to execution times. [3] and [32] studied the preservation of properties when introducing various sources of delays and digital (discrete) clocks in the implementations, to represent realistic executions on the hardware platform. [21, 22] takes a different approach than [3, 32] by trying to actively counteract the effect of delays in the generated code so as to meet properties.

In the context of distributed platforms, existing implementation frameworks [18, 9, 19, 11] for real-time applications are restricted to time-deterministic systems, which is a strictly less expressive than timed automata as explained in [1]. They also consider much simpler coordination mechanisms than multiparty interactions proposed in this paper. The generation of distributed implementations from components subject to multiparty (nary) interactions has been extensively studied in the untimed context [10, 8], and more recently for timed systems under the assumption of non-decreasing deadlines in [28]. The principle is to transform multiparty interactions into coordination mechanisms using simpler primitives such as asynchronous point-to-point messages, so that they can be mapped directly on communication mechanisms offered by distributed platforms. We contribute to this research field by considering in addition delays between the decision to execute an interaction and its actual execution. They are due to the transmission delays between the component responsible for such a decision and the components involved in the interaction, and may have a huge impact in the satisfaction of timing constraints in real-time systems. Indeed, such delays may introduce behavioral flaws (e.g. deadlocks) when dealing with arbitrary timing constraints (i.e. no restriction to the non-decreasing deadlines case), as shown in [16]. Our contribution consists in *(i)* the introduction of a semantics based on partial states of the system components and that includes a complete formalization of the effect of the delays in this context, and *(ii)* practical means for enforcing system correctness in their presence. This paper is an extension of the work presented in [16]. The semantics proposed in [16] allows to choose arbitrary (i.e. non-negative) delays between decisions and executions, which is not realistic. We improve [16] by restricting such delays with respect to lower bounds representing worst-case estimates of communication delays. We also updated accordingly the underlying semantics by restricting the progress of time as well as the sufficient conditions for system correctness presented in [16]. As explained in [16], they can be used in some cases to derive simple execution strategies achieving correctness. When our method is not applicable (i.e. the sufficient conditions cannot be met), an alternative method could be to use existing frameworks for control synthesis in timed automata. However, as we explained in this paper the problem addressed here cannot be fully expressed in these frameworks [6, 2], and had to be simplified by a discretization step. Moreover, when applicable our method remains faster than this alternative.

8 Conclusion and Future Work

We presented a local planning semantics for scheduling real-time systems in a distributed context. The proposed approach intends to mitigate the effect of communication delays through planning interactions ahead. A sufficient deadlock-freedom condition has been proved, a compositional verification method for checking action-time-lock-freedom was provided, and a simple execution strategy, based on restricting upper bounds horizons planning of interactions, has been presented. Additionally, a formalization of the planning problem as a real-time controller synthesis approach has been provided. This work shows how to express the planning semantics as timed game automata and highlights the encountered issues met during the formalization.

This approach opens a number of directions for future work. In case of action-time-locks of the planning semantics, a first idea consists to study their origins and derive a refinement method for models in order to take into account the communication delays. Another interesting direction is the characterization of the reachable states of the planning semantics. Instead of using an over-approximation of systems reachable states under the standard semantics, a more accurate approach could be to define a method for deriving invariants w.r.t the local planning semantics. Finally, an interesting idea is to investigate how scheduler(s) can benefit from the information provided by the presented method in order to optimize their scheduling policy.

References

1 Tesnim Abdellatif, Jacques Combaz, and Joseph Sifakis. Model-based implementation of real-time applications. In *Proceedings of the 10th International conference on Embedded software, EMSOFT 2010, Scottsdale, Arizona, USA, October 24-29, 2010*, pages 229–238, 2010. doi:10.1145/1879021.1879052.

2 Karine Altisen, Gregor Gößler, and Joseph Sifakis. Scheduler Modeling Based on the Controller Synthesis Paradigm. *Real-Time Systems*, 23(1-2):55–84, 2002. doi:10.1023/A:1015346419267.

3 Karine Altisen and Stavros Tripakis. Implementation of Timed Automata: An Issue of Semantics or Modeling? In *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, pages 273–288, 2005. doi:10.1007/11603009_21.

4 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

5 Ananda Basu, Laurent Mounier, Marc Poulhiès, Jacques Pulou, and Joseph Sifakis. Using BIP for Modeling and Verification of Networked Systems – A Case Study on TinyOS-based Networks. In *Sixth IEEE International Symposium on Network Computing and Applications (NCA 2007), 12 - 14 July 2007, Cambridge, MA, USA*, pages 257–260, 2007. doi:10.1109/NCA.2007.52.

6 Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: Time for Playing Games! In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, pages 121–125, 2007. doi:10.1007/978-3-540-73368-3_14.

7 Johan Bengtsson and Wang Yi. On Clock Difference Constraints and Termination in Reachability Analysis of Timed Automata. In *Formal Methods and Software Engineering, 5th International Conference on Formal Engineering Methods, ICFEM 2003, Singapore, November 5-7, 2003, Proceedings*, pages 491–503, 2003. doi:10.1007/978-3-540-39893-6_28.

8 Saddek Bensalem, Marius Bozga, Jean Quilbeuf, and Joseph Sifakis. Optimized distributed implementation of multiparty interactions with observation. In *Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions, AGERE! 2012, October 21-22, 2012, Tucson, Arizona, USA*, pages 71–82, 2012. doi:10.1145/2414639.2414649.

9 Gérard Berry and Georges Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *Sci. Comput. Program.*, 19(2):87–152, 1992. doi:10.1016/0167-6423(92)90005-V.

10 Borzoo Bonakdarpour, Marius Bozga, and Jean Quilbeuf. Model-based implementation of distributed systems with priorities. *Design Autom. for Emb. Sys.*, 17(2):251–276, 2013. doi:10.1007/s10617-012-9091-0.

11 Sylvain Camier, Damien Chabrol, Vincent David, and Christophe Aussaguès. OASIS formal approach for distributed safety-critical real-time system design. In *ISoLA 2007, Workshop On Leveraging Applications of Formal Methods, Verification and Validation, Poitiers-Futuroscope, France, December 12-14, 2007*, pages 167–178, 2007. URL: http://editions-rnti.fr/?inprocid=1000543.

12 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Efficient On-the-Fly Algorithms for the Analysis of Timed Games. In *CONCUR 2005 - Concurrency Theory, 16th International Conference,*

CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings, pages 66–80, 2005. doi:10.1007/11539452_9.

13 Robert N. Charette. This Car Runs on Code. *IEEE Spectrum*, 2009.

14 Conrado Daws, Marta Z. Kwiatkowska, and Gethin Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *STTT*, 5(2-3):221–236, 2004. doi:10.1007/s10009-003-0118-5.

15 Mahieddine Dellabani, Jacques Combaz, Saddek Bensalem, and Marius Bozga. Knowledge Based Optimization for Distributed Real-Time Systems. In *24th Asia-Pacific Software Engineering Conference, APSEC 2017, Nanjing, China, December 4-8, 2017*, pages 751–756, 2017. doi:10.1109/APSEC.2017.106.

16 Mahieddine Dellabani, Jacques Combaz, Marius Bozga, and Saddek Bensalem. Local Planning of Multiparty Interactions with Bounded Horizons. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings*, pages 199–216, 2016. doi:10.1007/978-3-319-48989-6_13.

17 Bruno Dutertre and Leonardo de Moura. The Yices SMT solver. Technical report, SRI International, 2006.

18 Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and Verifying Real-Time Systems by Means of the Synchronous Data-Flow Language LUSTRE. *IEEE Trans. Software Eng.*, 18(9):785–793, 1992. doi:10.1109/32.159839.

19 Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. Giotto: a time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1):84–99, 2003.

20 Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic Model Checking for Real-Time Systems. *Inf. Comput.*, 111(2):193–244, 1994. doi:10.1006/inco.1994.1045.

21 BaekGyu Kim, Lu Feng, Linh T. X. Phan, Oleg Sokolsky, and Insup Lee. Platform-specific timing verification framework in model-based implementation. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, DATE 2015, Grenoble, France, March 9-13, 2015*, pages 235–240, 2015. URL: http://dl.acm.org/citation.cfm?id=2755804.

22 BaekGyu Kim, Lu Feng, Oleg Sokolsky, and Insup Lee. Platform-Specific Code Generation from Platform-Independent Timed Models. In *2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015*, pages 75–86, 2015. doi:10.1109/RTSS.2015.15.

23 Christos Kloukinas and Sergio Yovine. A model-based approach for multiple QoS in scheduling: from models to implementation. *Autom. Softw. Eng.*, 18(1):5–38, 2011. doi:10.1007/s10515-010-0074-8.

24 Hermann Kopetz. An Integrated Architecture for Dependable Embedded Systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 160–161, 2004. doi:10.1109/RELDIS.2004.1353016.

25 Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear Controller. In *Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, pages 281–297, 1998. doi:10.1007/BFb0054178.

26 Braham Lotfi Mediouni, Ayoub Nouri, Marius Bozga, Mahieddine Dellabani, Jacques Combaz, Axel Legay, and Saddek Bensalem. SBIP 2.0: Statistical Model Checking Stochastic Real-time Systems. Technical Report TR-2018-5, Verimag Research Report, 2018.

27 Souha Ben Rayana, Marius Bozga, Saddek Bensalem, and Jacques Combaz. RTD-Finder: A Tool for Compositional Verification of Real-Time Component-Based Systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 394–406, 2016. doi:10.1007/978-3-662-49674-9_23.

28 Ahlem Triki. *Distributed Implementations of Timed Component-based Systems. (Implémentations distribuées des systèmes temps-réel à base de composants).* PhD thesis, Grenoble Alpes University, France, 2015. URL: https://tel.archives-ouvertes.fr/tel-01169720.

29 Ahlem Triki, Borzoo Bonakdarpour, Jacques Combaz, and Saddek Bensalem. Automated Conflict-Free Concurrent Implementation of Timed Component-Based Models. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, pages 359–374, 2015. doi:10.1007/978-3-319-17524-9_25.

30 Stavros Tripakis. *L'analyse formelle des systèmes temporisés en pratique. (The Formal Analysis of Timed Systems in Practice).* PhD thesis, Joseph Fourier University, Grenoble, France, 1998. URL: https://tel.archives-ouvertes.fr/tel-00004907.

31 Stavros Tripakis. Verifying Progress in Timed Systems. In *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, pages 299–314, 1999. doi:10.1007/3-540-48778-6_18.

32 Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Almost ASAP Semantics: From Timed Models to Timed Implementations. In *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings*, pages 296–310, 2004. doi:10.1007/978-3-540-24743-2_20.

Improving WCET Evaluation using Linear Relation Analysis*

Pascal Raymond
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Pascal.Raymond@univ-grenoble-alpes.fr

Claire Maiza
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Claire.Maiza@univ-grenoble-alpes.fr

Catherine Parent-Vigouroux
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Catherine.Parent-Vigouroux@univ-grenoble-alpes.fr

Erwan Jahier
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Erwan.Jahier@univ-grenoble-alpes.fr

Nicolas Halbwachs
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Nicolas.Halbwachs@univ-grenoble-alpes.fr

Fabienne Carrier
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Fabienne.Carrier@univ-grenoble-alpes.fr

Mihail Asavoae
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Mihail.Asavoae@univ-grenoble-alpes.fr

Rémy Boutonnet
Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), VERIMAG, Grenoble, France
Rémy.Boutonnet@univ-grenoble-alpes.fr

Abstract

The precision of a worst case execution time (WCET) evaluation tool on a given program is highly dependent on how the tool is able to detect and discard semantically infeasible executions of the program. In this paper, we propose to use the classical abstract interpretation-based method of *linear relation analysis* to discover and exploit relations between execution paths. For this purpose, we add auxiliary variables (counters) to the program to trace its execution paths. The results are easily incorporated in the classical workflow of a WCET evaluator, when the evaluator is based on the popular *implicit path enumeration technique*. We use existing tools – a WCET evaluator and a linear relation analyzer – to build and experiment a prototype implementation of this idea.

* This work is supported by the French research fundation (ANR) as part of the W-SEPT project (ANR-12-INSE-0001)

2012 ACM Subject Classification Software and its engineering → Real-time systems software
Keywords and Phrases Worst Case Execution Time estimation, Infeasible Execution Paths, Abstract Interpretation
Digital Object Identifier 10.4230/LITES-v006-i001-a002
Received 2017-12-12 **Accepted** 2018-11-19 **Published** 2019-02-18

1 Introduction

The computation of a precise and safe approximation of the worst case execution time (WCET) of programs on a given architecture is an important step in the design of hard real-time systems [41]. It is part of the validation of the design, and a prerequisite for tasks scheduling. In this computation, over-approximation is mainly due to pessimistic abstraction of (1) complex hardware mechanisms (caches, pipeline) and (2) the program semantics (loop bounds, infeasible executions). Taking into account the target execution platform is, by far, the most difficult problem. It has been largely studied in the literature and remarkable tools exist, both in the academia [5, 27, 29] and in the industry [40].

In this paper, we specifically address the problem of taking into account the program semantics. The objective is to extract semantic properties that make some executions infeasible, and to exploit these properties in an existing WCET evaluator. It is generally admitted that such properties are easier to analyze on high-level code – e.g., C programs – than on binary, even if semantic analysis of executable code has been explored [3, 4, 36]. WCET evaluation is performed on object code in order to be able to take into account the execution architecture. This raises the problem of traceability between the source and the object code.

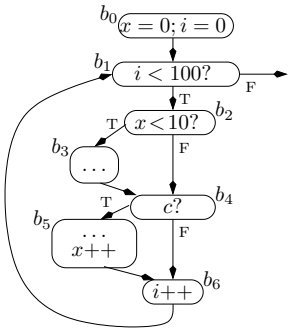
The most popular approach to evaluate the WCET is called *implicit path enumeration technique* (IPET) [28]. A micro-architectural analysis provides an evaluation of the duration of each basic block of the object-code control-flow graph. The WCET is then expressed as the solution of an integer linear programming problem (ILP) where the variables are the number of times each basic block is traversed during an execution. Relations between these variables come from the control-flow graph (flow equations) and from semantic "*flow facts*", including at least *loop bounds*. Indeed, each loop in the program should have a constant bound to guarantee that the execution time is finite; such bounds may be provided by the user, or discovered by program analysis.

Hence, the IPET-based evaluation takes into account semantic properties expressed as linear constraints on counters. A natural idea is then to combine it with a semantic analysis devoted to the discovery of invariant linear relations. Polyhedra-based abstract interpretation [2, 8, 17, 20], also called *linear relation analysis* (LRA), is such an analysis. It is able to associate with each control point of a sequential program a system of linear inequalities (whose set of solutions is a convex polyhedron) satisfied by the numerical variables at this control point in any execution of the program.

Our proposal consists in applying LRA to a copy of the source program enriched with counter variables, and translate the obtained relations between counters into constraints to be added to the ILP. Let us illustrate this proposal with a simple example.

```
x = 0; i = 0;
while (i < 100) {
  if (x < 10) {
    ...
  }
  if (c) {
    ...
    x++;
  }
  i++;
}
```

(a) initial program



(b) control flow graph

```
x = 0; i = 0; α= 0; β= 0; γ= 0;
while (i < 100) {α++;
  if (x < 10) {β++;
    ...
  }
  if (c) {γ++;
    ...
    x++;
  }
  i++;
}
```

(c) instrumented program

Figure 1 Instrumenting an example program with counters.

1.1 An example

Consider the program fragment of Figure 1.a with its control-flow graph (Fig. 1.b). Let us add counters α, β, γ to the main basic blocks as done in the instrumented program Figure 1.c. These counters are initialized to 0 and incremented in their corresponding block. An LRA analysis of this instrumented program automatically discovers that the following relations are satisfied at the end of the program:

$$\alpha = \texttt{i} = 100\ ,\ \gamma = \texttt{x}\ ,\ \beta + \gamma \leq 110\ ,\ \gamma \geq 0\ ,\ \beta \geq 0$$

The inequality $\alpha = 100$ gives the exact bound of the loop. More interestingly, $\beta + \gamma \leq 110$ means that there are at most 10 iterations of the loop where both blocks b_3 and b_5 are executed.

Assume the object code has the same control structure as the C program, i.e., the basic blocks of their control flow graphs are in an one-to-one correspondence. The standard WCET evaluation computes pessimistic execution times (say $t_i, i = 0..6$) of the basic blocks $(b_i, i = 0..6)$, and constructs the following ILP, where n_i (resp., $e_{i,j}$) denotes the number of occurrences of the basic block b_i (resp., the edge from b_i to b_j) in an execution of the program:

$$wcet = \max \sum_{i=0}^{6} n_i.t_i \quad , \quad \text{with the constraints} \quad \left\{ \begin{array}{lll} n_0 = 1 & , & e_{0,1} = n_0 \\ n_1 = e_{0,1} + e_{6,1} & , & e_{1,2} + 1 = n_1 \\ n_2 = e_{1,2} & , & e_{2,3} + e_{2,4} = n_2 \\ n_3 = e_{2,3} & , & e_{3,4} = n_3 \\ n_4 = e_{2,4} + e_{3,4} & , & e_{4,5} + e_{4,6} = n_4 \\ n_5 = e_{4,5} & , & e_{5,6} = n_5 \\ n_6 = e_{4,6} + e_{5,6} & , & e_{6,1} = n_6 \end{array} \right\}$$

If we are able to maintain the correspondence between basic blocks in the source and the object code, i.e., to associate our counters α, β, γ with the variables of the ILP (n_2, n_3, n_5 respectively), we can add to the ILP the corresponding constraints: $n_2 = 100, n_3 + n_5 \leq 110$, which is likely to reduce the maximum value of the objective function[1].

[1] In fact, for this simple example, the results can be computed symbolically: concerning the standard evaluation, if the number of iterations in the loop (= 100) is given as a flow fact, the result will be $t_0 + 101t_1 + 100(t_2 + t_3 + t_4 + t_5 + t_6)$. Taking the additional constraint into account, we get $t_0 + 101t_1 + 100(t_2 + t_4 + t_6) + 100\max(t_3, t_5) + 10\min(t_3, t_5)$ thus improving the previous result by $90\min(t_3, t_5)$.

1.2 Contents of the paper

In Section 2, we focus on some available tools, and experiment their semantics awareness on some simple examples. Two recent papers were dedicated to the state of the art related to semantic analyses for WCET estimation and infeasible path detection [1, 10]. Section 2.3 presents some more recent publications.

Our proposal consists in combining existing techniques, namely IPET-based WCET analysis and Linear Relation Analysis, recalled in Section 3, together with the specific tools that we used in our implementation. In Section 4, we explain how the counters are added and related to ILP counters thanks to debugging information provided by the compiler. Our implementation of the method is used to validate the approach on two existing benchmarks. We also investigated the robustness of the approach in presence of compiler optimizations. These experiments are summarized in Section 5. We conclude with the discussion of possible future work.

2 Existing tools

We have experimented with some existing tools, to evaluate their ability to discover and exploit semantic properties. Four tools have been considered, all of which go through similar steps:

1. extracting a control-flow graph from the object code,
2. performing a set of micro-architectural analyses to obtain execution times for each basic blocks,
3. using IPET to compute a safe WCET.

We compare these tools with respect to their capabilities to extract semantic properties to cut infeasible paths.

2.1 The tools

2.1.1 The Chronos Timing Analyzer

Chronos [27] is an academic tool developed at National University of Singapore. It takes as input a C program, performs limited data-flow analysis at C source code level to determine loop bounds, and requests the user to provide this information when it fails. The semantic analysis in Chronos uses a pattern-based method to detect infeasible paths [39]. The so-called two-phase technique addresses infeasibility from a *conflicting pairs* point of view. In the first phase, an analysis detects some conflicts that capture the fact that two branches can not be taken along the same path. In the second phase, each conflicting pair relation is encoded into an ILP constraint.

2.1.2 The Swedish Timing Analyzer

SWEET[2] [29] is a research toolbox developed at Mälardalen Real-Time Research Center (MRTC). The main objective of SWEET is flow analysis, which computes flow-facts, i.e., information about loop bounds and infeasible paths in the program. The main technique to discover flow-facts is abstract execution [16]. Abstract execution is a form of context-sensitive abstract interpretation, because it uses a symbolic execution to produce context information for each loop iteration and function call. Instead of using the fixpoint engine of abstract interpretation, abstract execution executes the program in the abstract domain, merging the execution paths at certain points in the program. SWEET does not support LRA. It currently implements only the abstract domain of intervals.

[2] `www.mrtc.mdh.se/projects/wcet`

2.1.3 AbsInt - The aiT Tool

Developed by AbsInt[3], aiT is the main industrial product for WCET analysis. It consists of a set of binary executables analyzers, which take the intrinsic cache and pipeline behavior into account. Concerning semantic analysis, aiT uses a value analysis based on intervals [13] to compute safe ranges of values for the program variables. aiT uses this information to determine loop bounds and detect infeasible paths. The approach towards computing loop bounds is not general, but it handles loop patterns. In order to gain precision, aiT pre-processes each loop by transforming its body into a function, in order to expose the iteration contexts. The key element in this transformation is to identify the loop index and to set it as a function parameter. Then, an interval analysis computes the ranges for all the loop variables. The loop transformation is based on loop patterns, which depend on the particularities of the architecture (e.g., parameter order) or on the loop structure (e.g., for-loops, triangular-loops, branch conditions). aiT is able to detect infeasible paths using the results of the value analysis, like conditions made infeasible because of the computed intervals.

2.1.4 oRange, the flow fact analyzer of OTAWA

OTAWA [5] is an academic toolbox, developed at IRIT (University of Toulouse), designed as a generic framework to develop static analyses for WCET computation. Although OTAWA implements several approaches to WCET computation, the one based on IPET is the most mature. OTAWA relies on an auxiliary tool, called oRange [9], to compute loop bounds. oRange analyses C code. As a first phase, oRange detects loop indices and constructs a normal form: a symbolic expression of the bound independently of the call context. In a second phase, by an abstract execution, a syntactic tree is built in function of a full or partial call context. It combines loop bounds and conditional expressions as numeric or symbolic expressions. Finally, the tree is computed in the full context in order to produce a file in the specific flow-facts format FFX [42].

2.2 Some experiments

In order to evaluate the capabilities of these tools to detect infeasible paths, we have applied each of them to programs containing various situations of semantic infeasibility. These situations are given in Figure 2:

- Example 1 is a case where simple pattern-based method may fail, since constant propagation is needed.
- Example 2 may be a problem for pattern-based methods for finding iteration numbers, since the apparent index x is modified.
- Example 3 is our introductory example of §1.1.
- Example 4 is a fragment of code generated by the SCADE[4] compiler, from a design manipulating arrays. On one hand, the loops are exited from inside, which complicates the evaluation of iteration numbers. On the other hand, the third loop is unreachable because of some non-trivial arithmetic conditions.

Table 1 summarizes the results of the tools on these examples. On Example 1, Chronos is unable to detect dead code[5]. Example 2 is correctly analyzed only by SWEET, because it unrolls

[3] AbsInt GmbH `www.absint.com/ait/`

[4] `www.esterel-technologies.com/products/scade-suite`

[5] We used the available version of Chronos. Some additional work has been done that complement the infeasible path analysis [6, 37], which is not part of the available version.

```
y = 2 ; x = y ;
if (x>3){// unreachable
  ...
}
```

Example 1

```
x = 0 ;
while (x<10){
  ...
  if (C1) {
    x = ...
    ...
  }
  x++ ;
  // iteration nbr. may be ≠ 10
}
```

Example 2

see example §1.1

Example 3

```
max_i = 3 ; s1 = 0;
for (i=0 ; i<10 ; i++) {
  s1 = s1 + t[i] ;
  // each t[i] assumed to be >5
  x = i+1 ;
  if (!(i<max_i)) break ;
  // intended for 4 iterations
}
int s2 = s1 ;
if (x < 6)
  for (i=0 ; i<10 ; i++){
    s2 = s2+2 ;
    if (!(i<5)) break ;
    // intended for 6 iterations
  }
int f = 0 ;
if (2*s1 < s2)
  // unreachable, since s1>20 and s2=s1+10
  for (i=0 ; i<10 ; i++){
    f = f+2 ;
    if (!(i<4)) break ;
  }
```

Example 4

Figure 2 Various cases of semantic infeasibilities.

Table 1 Results of tools on programs of Figure 2.

	Chronos	SWEET	oRange	aiT
Example 1	-	✓	✓	✓
Example 2	-	✓	-	-
Example 3	-	-	-	-
Example 4				
nbr. 1st loop	-	4	10	4
nbr. 2nd loop	-	5	10	5
dead code	-	-	-	✓

loops. None of the tools is able to find the property of Example 3. On Example 4, Chronos requires manual annotations for loop bounds; oRange estimates that both loops are iterated 10 times; SWEET and aiT find the exact loop bounds; only aiT detects dead code.

In this paper, we propose a method and a tool-chain that is able to discover the infeasible paths of these 4 examples, namely, infeasible paths that depend on a semantic analysis and that may concern distant program points.

2.3 Other approaches

An extended state of the art related to semantic analyses for WCET estimation can be found in [1] and a general survey of infeasible path detection in [10]. We complement them with some more recent publications.

Several recent works make use of SMT solvers [23, 37]. The idea is to ask the solver if the worst-case path obtained by the ILP solver is feasible. Whenever the path is infeasible, a corresponding constraint is added to the ILP. As in our approach, adding constraints does not always mean that the WCET is refined (2 paths may have the same WCET). In [18], the whole

path analysis is done through SMT solving instead of ILP: infeasible path analysis and worst-case path analysis are merged in one step. Path execution time is expressed as an SMT problem, and the question asked is no longer "is this path feasible?", but "is there a feasible path longer than K?", where K is a given constant (which is adjusted, e.g., by binary search). In [32, 35], a similar approach is taken, by asking this question to a bounded model checker.

3 Used techniques and tools

This section presents the existing techniques and tools used in our prototype: OTAWA implements the classical IPET-based WCET evaluation, and PAGAI performs Linear Relation Analyis.

3.1 WCET evaluation with OTAWA

The WCET estimation work-flow (Figure 3) involves a compiler, a Linear Program solver, and two tools from the OTAWA toolbox: oRange and owcet.

- Compilation: the source C code is compiled by a third party tool; for this experiment, we use a cross compiler from the GNU Compiler Collection (arm-elf-gcc 4.4.2), but other compilers can be used, provided that it produces ELF code (Executable and Linkable Format), with debugging information in DWARF format.
- oRange is a *data flow analyzing tool*, dedicated to the discovery of *loop bounds*. Bounds are stored in the OTAWA *flow facts* format (FFX).
- owcet is the OTAWA command dedicated to the WCET evaluation. The main steps of this tool, not detailed in Figure 3, are:
  - the construction of the control-flow graph (CFG) of the object code; during the construction, and thanks to debugging information, basic blocks (BB) are associated (if possible) to lines in the source program; thanks to this correspondence, the loop bounds computed by oRange are translated into control flow constraints in the CFG. The annotated CFG can be dumped in a file, allowing other tools to exploit it.
  - the micro-architectural analysis, which associates a local WCET estimation with each BB of the CFG.
  - the construction of the Integer Linear Programming (ILP) system; as in the introduction example (§1.1) the resulting system gathers (1) structural constraints (CFG structure), (2) loop bounds constraints (from oRange flow facts) (3) the objective function to be maximized (sum of BB counters weighted by their local WCET).
- ILP solver: the ILP system is then solved by a third-party tool; OTAWA integrates and uses LP_SOLVE[6] (any other equivalent tool can be used).

3.2 Linear Relation Analysis with PAGAI

3.2.1 Principles of LRA

Linear Relation Analysis [8] is a classical program analysis, based on abstract interpretation [7]. It is able to discover, at each control point of a sequential program, a conjunction of linear relations (equalities and inequalities) invariantly satisfied by the numerical variables at this point. Classical algorithms are used to propagate linear systems over the statements of the program. Several causes may result in information loss:

[6] `web.mit.edu/lpsolve/doc`

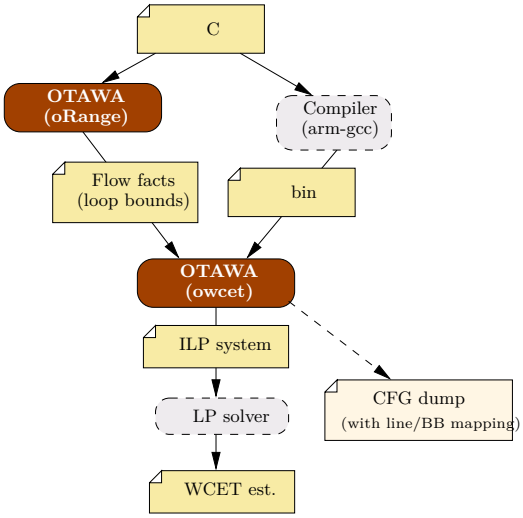


Figure 3 Otawa WCET estimation work-flow.

- the analysis safely ignores non-linear expressions in assignments and tests;
- the analysis performs a *convex hull* at control path junctions, instead of propagating the disjunction of incoming information. It means that the propagated value is the most precise conjunction of linear relations implied by both incoming systems;
- to avoid infinite propagation along loops, the classical *widening-narrowing* method is applied to guess a safe approximation of the limit. Note that, unlike in symbolic execution [23] or SMT methods [18], loops are not unrolled.

3.2.2 Applying LRA to our example

We do not detail further the techniques applied in LRA, and refer the reader to the bibliography. We just show the main steps of the analysis of our example of Figure 1. Let us consider the control point at the entry of the while loop. The first step of the analysis straightforwardly computes the first iterate:

$$x = i = \alpha = \beta = \gamma = 0$$

Its propagation through the loop body provides, with a convex hull at the end of the conditional:

$$i = \alpha = \beta = 1\ ,\ x = \gamma\ ,\ 0 \leq \gamma \leq 1$$

Now a convex hull with the first iterate gives the second iterate at the entry of the loop:

$$i = \alpha = \beta\ ,\ 0 \leq \alpha \leq 1\ ,\ x = \gamma\ ,\ 0 \leq \gamma \leq \alpha$$

Instead of continuing the iterations, a first widening/narrowing step is performed (using "lookahead widening" [14]), which provides:

$$i = \alpha = \beta\ ,\ x = \gamma\ ,\ 0 \leq \gamma \leq \alpha \leq 100\ ,\ \gamma \leq 10$$

Now, the "else" branch of the test `x<10` becomes feasible, and a second widening/narrowing step is performed, providing:

$$i = \alpha\ ,\ x = \gamma\ ,\ \beta + \gamma \leq \alpha + 10\ ,\ \beta \leq \alpha\ ,\ \gamma \leq \alpha \leq 100$$

which is found invariant after one more propagation. Propagated to the end of the program, it becomes:

$$i = \alpha = 100\ ,\ x = \gamma \leq 100\ ,\ \beta + \gamma \leq 110$$

3.2.3 LRA and loop bounds

It may happen, like in the previous example, that LRA discovers a bound to a loop counter, thus providing an essential information for WCET evaluation. However, finding loop bounds is *not* our main goal in this work, as the method is intrinsically unable to discover *non linear* relations, which drastically limits its capability to find loop bounds. As a matter of fact, in presence of nested loops, the number of executions of the body of the innermost loop is not linear in the constants of the program. For instance, in the program fragment "`for(i=0;i<n;i++){for(j=i;j<n;j++){...}}`" the number of executions of the body of the innermost loop is $n(n+1)/2$, which cannot be found by LRA.

The LRA method must then be used together with some other method able to bound nested loops. We can use existing tools such as oRange that comes with OTAWA, or more basically user-given bounds, given as pragmas in the code.

There exist also approaches based on polyhedra manipulation to find loop bounds, such as the one proposed in [38, 30]: it consists in building a polyhedral upper approximation P of the iteration domain, i.e., the set of possible valuations of loop counters (in the previous example, $P = \{(i,j) \mid 0 \leq i \leq j \leq n-1\}$). Under realistic assumptions concerning the determinism of the program, the number of executions of the innermost loop is bounded by the number of integer points in P, and algorithms are available to compute this number. Notice that LRA can be combined with this approach, since it can discover linear invariants reducing the iteration domain, thus improving the precision of the result. Notice also that LRA can deal with parameters (symbolic bounds, like n in our example), an issue specifically addressed by [38].

3.2.4 The PAGAI prototype analyzer

Several tools performing LRA are available ([2, 12, 19, 20] to cite a few). Here, we use the PAGAI prototype analyzer, which implements the basic LRA together with recent improvements like "lookahead widening" [14] and SMT-based "path focusing" [19]. PAGAI analyses LLVM code [24] produced from a C program (thanks to Clang[7]), and is able to return discovered properties at the C level. PAGAI may be used with other abstract domains than general linear systems – like octagons [33] – thanks to the common interface APRON [21].

4 Adding and tracing counters

4.1 The proposed workflow

Figure 4 shows the proposed workflow for the experiment. It involves two existing components: timing analysis with OTAWA (left) and program analysis with PAGAI (right). Two new tools have been developed to complete the workflow: a front-end (top, *Instrumentation*), which produces the input for the analyzers (OTAWA and PAGAI), and a back-end (*ILP translation & merge*), which gathers the results into a more constrained ILP system, and obtains a possibly enhanced WCET estimation.

[7] `http://clang.llvm.org/`

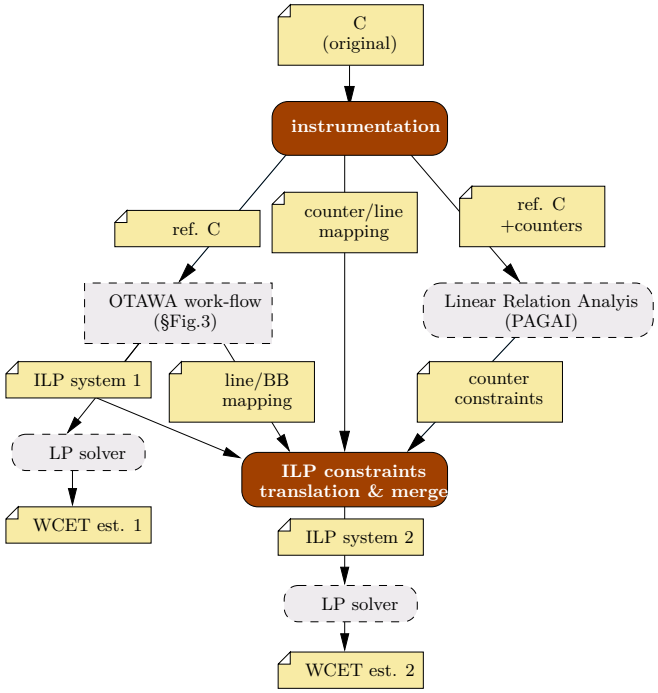


Figure 4 Instrumentation and analysis workflow.

These tools are detailed in this section. We illustrate the successive steps of the method by detailing the processing of an example program, called `lcdnum.c`, extracted from TacleBench programs suite [15]. The main program is given in Figure 5. It calls a function `num_to_lcd`, the execution time of which is taken into account by OTAWA.

4.2 Instrumented program version

The goal of the front-end ("instrumentation", Figure 4, top) is to produce, from the original C code, a reference C program. Some semantics preserving transformations of the source code are necessary or advisable, in order to use properly the analyzers, and trace the information between them.

- Some transformations are purely lexical, and do not change the program structure: because the standard ELF/DWARF traceability mechanism is line-based, line breaks are introduced to isolate each atomic statements on its own line.
- Some transformations that modify the control structure are necessary because of the limitation of the analyzers. For instance, a single-return statement per function is mandatory for exploiting the results of PAGAI: this unique control point is the place where counter invariants actually express properties on the whole execution of the function. Other transformations are required because of the limitation of both OTAWA and PAGAI: the control structure (CFG) must be statically known, which forbids dynamic computation of program pointers. In particular, "switch/case" statements must be rewritten into a static control structure based on "if" and "goto" statements.
- Another transformation is desirable in our case: the current version of PAGAI does not handle inter-procedural analysis. In order to exploit the plain capacity of this tool to find invariants, a light-weight solution is to inline function calls at the source level. This transformation is

indeed hardly admissible in real-life, but it must be seen here as a "trick" to reach our goal (study the ability of LRA to detect infeasible executions).

The front-end produces the reference C code in two flavors.

- The reference C code with counters (Figure 4, right) is instrumented with auxiliary counters, in the same manner as in the introductory example (§ 1.1). The present version introduces a counter for each sequential block in the program control flow. However, some strategy could be used to reduce the number of counters by targeting blocks that are more likely to have an influence [43].
- The reference C code without counters (Figure 4, left) is the same code, where all lines related the counters (declaration, initialization and incrementation) have been commented out. This method ensures a semantic equivalence between the programs analyzed by OTAWA and PAGAI: since they only differ on the side-effect-free local variables, these programs are naturally input/output equivalent. Moreover, at least at the source level, the two programs are also structurally equivalent: a block in the reference C code is executed if and only if the corresponding block (marked with a counter c) is executed in the reference C program with counters. This property becomes false in general at the binary level, since the C compiler may modify the control structure: this well-know problem of traceability is discussed later.
- An auxiliary file is generated, that contains the mapping between each counter and its corresponding source line in the reference C code.

▶ **Example 1.** Applied to our example program (Figure 5), our instrumentation front-end calls the C preprocessor, eliminates the multiple returns and switches (only within **`num_to_lcd`**, not shown), and produces the reference C programs. The first one (without counters) is shown on Figure 6; the second (not shown) is exactly the same with uncommented lines involving counters. An auxiliary file (not shown) simply lists the pairs "counter/line" (e.g., **`(cptr_main_1,144)`**, **`(cptr_main_2, 147)`**).

The first version is provided to OTAWA. Loop bounds computation by oRange is optional, which allows us to check if PAGAI is able to find them on its own. OTAWA calls the gcc compiler (here with -O0 optimization level), builds the CFG of the object code, performs the micro-architectural analysis, and builds the ILP problem.

PAGAI is applied to the second version of the program, and returns the following invariants:

```
-10+cptr_main_2 = 0
-10+cptr_main_4 = 0
5-cptr_main_3 >= 0
```

The first equation finds the exact loop bound (which may also be found by oRange). The second equation is structural (from the shape of the source CFG, cptr_main_2 and cptr_main_4 are equal). The third property is new, and expresses, in particular, that the function **`num_to_lcd`** is called at most 5 times.

4.3 Tracing back the counters

The back-end ("ILP constraints translation & merge", Figure 4, bottom) gathers the information coming form OTAWA and PAGAI:

- Thanks to the counter/C-line mapping provided by the front-end, and the C-line/binary-BB mapping provided by OTAWA (through the ELF/DWARF information), a counter/BB mapping is built. Note that this mapping is partial, and deliberately pessimistic: depending on the compilation process, it may happen that a counter is associated either to zero or to

```
unsigned char num_to_lcd( unsigned char a ) ;

volatile unsigned char IN = 120;
volatile unsigned char OUT;
int main( void ) {
  int i;
  unsigned char a;
  for( i=0; i< 10; i++ ) {
    a = IN;
    if(i<5) {
      a = a &0x0F;
      OUT = num_to_lcd(a);
    }
  }
  return 0;
}
```

Figure 5 The initial `lcdnum.c` program.

```
133 int main(void) {
134   int i ;
135   unsigned char a ;
136   unsigned char tmp ;
137   int __retres4 ;
138   //int cptr_main_1 = 0;
139   //int cptr_main_2 = 0;
140   //int cptr_main_3 = 0;
141   //int cptr_main_4 = 0;
142   //int cptr_main_5 = 0;
143   //cptr_main_1 ++; #line 144
144   i = 0;
145   while (i < 10) {
146     //cptr_main_2 ++; #line 147
147     a = (unsigned char )IN;
148     if (i < 5) {
149       //cptr_main_3 ++; #line 150
150       a = (unsigned char )((int )a & 15);
151       tmp = num_to_lcd(a);
152       OUT = (unsigned char volatile )tmp;
153     }
154     //cptr_main_4 ++; #line 155
155     i ++;
156   }
157   //cptr_main_5 ++; #158
158   __retres4 = 0;
159   return (__retres4);
160 }
```

Figure 6 The reference `lcdnum.c` program.

Table 2 Mapping between counters and blocks.

line number(s)	block(s)	reliable	counter
136,144	1	yes	cptr__main__1
145	1;2	no	
147;148	4	yes	cptr__main__2
150;151;152	5	yes	cptr__main__3
155	6	yes	cptr__main__4
158;159;160	3	yes	cptr__main__5

several binary basic blocks. In this case, the counter is simply ignored: only counters that are associated to one single BB are retained.

▶ **Example 1** (cont.). Table 2 shows the mapping between counters and blocks that is built by our back-end.

- The linear constraints on the retained counters are then translated literally into linear constraints on BB, and added to the basic ILP system provided by OTAWA.

  ▶ **Example 1** (cont.). The translation of the constraints discovered by PAGAI is the following:

```
x4_main = 10;
x6_main = 10;
x5_main <= 5;
```

- At last, both systems are solved and the corresponding estimations can be compared.

  ▶ **Example 1** (cont.). After a second call to LP_SOLVE, the final result is printed:

```
Estimation WITHOUT PAGAI: 1540
Estimation WITH PAGAI: 945
```

4.4 Traceability and optimization

In our framework, traceability is the ability to relate execution paths in the binary code (bin. CFG) to execution paths in the source code (source CFG).

Some optimizations performed by the compiler may strongly modify the control structure and thus alter traceability: loop unrolling, block replication, out-of-order execution. This is why most of the related works assume no compiler optimization to guarantee a perfect matching between the two CFGs.

However forbidding optimization is not satisfactory in real-time domains, where execution times have to be predictable, but also short. For a standard compiler like gcc, the observed speed-up between no optimization (-O0 option) and a standard level of optimization (-O1) is around two.

The most satisfactory solution would be a compiler that provides a precise traceability even in case of CFG optimization. Some work has been done to design and/or adapt the compilation process for this purpose, for instance [26, 31, 22].

Unfortunately, off-the-shelf standard compilers such as gcc hardly provide a precise and reliable information in case of CFG optimization. The idea is then to use the compiler options in order to forbid (as far as possible) CFG transformations, but still allow other optimizations, in particular those that concern data management.

The gcc compiler proposes numerous options to control optimizations, but there hardly exists a comprehensive and exhaustive description of their effects and inter-dependencies. For this experiment, we have empirically defined a *customized level* (called `C0` in the sequel). We started from the standard `-O1` level, and removed about 20 individual optimizations using the `-fno` directive (see appendix B). We cannot guarantee that this customized level will preserve the CFG for all programs, but the method is safe: as explained in Section 4.3, a counter (and then a source code line) that is not associated to exactly one basic block of the binary code is simply ignored. As a consequence, the only risk is to lose information that would have made the WCET estimation tighter. Note that this statement suppose that the gcc debugging information is reliable, which is indeed unprovable, but *empirically* reasonable.

▶ **Example 2.** When applying the `CO` method to our running example, we get 100% traceability. As a consequence, the interesting counter property (`5-cptr_main_3 >= 0`) can still be translated into a BB constraint (`x11_main <= 5;`) leading to the final result:

```
Estimation WITHOUT PAGAI: 641
Estimation WITH PAGAI: 421
```

On this example, we observe that code optimization leads to an initial WCET estimation 2.4x smaller (641 vs 1540). The traceability is preserved and the improvement due to the counter analysis is of the same order (34.3% vs 38.6%).

5 Experiments

5.1 Benchmarks

We tested our approach on programs from the TacleBench [11], a set of C programs widely used in the WCET community. [8] A first check has been made to retain only purely sequential programs that compile "out of the box": 53 applications of the 58 in the TacleBench[9]

For each program, we try to estimate the WCET of all functions appearing in the code, including the top-level one (main). For each function, inner function calls are recursively inlined at the C level (see Section 4.2). Recursive functions are rejected during this step, and not considered for WCET analysis.

Our goal is to study the influence of our counter-based method (Fig. 4) on a classical estimation (Fig. 3). A prerequisite is therefore that a *reference* estimation exists; hence the programs for which the basic WCET estimation fails are not selected. The OTAWA estimation may fails because of unsupported programming features (pointer arithmetics), or because the analysis does not terminates before a chosen timeout (2 hours).

After this initial selection, 589 functions (out of 639) from the 53 programs of the TacleBench suite are retained.

5.2 Experimental setup

The proposed framework as presented on Fig. 4 has numerous parameters (C code instrumentation, linear analysis tuning, compiler optimization etc.) leading to a combinatorial numbers of possibilities. For this systematic experiment, we focus only on two kinds of parameters: those that influence the precision of linear analysis, and those that influence the traceability. The other parameters are fixed once and for all as follows:

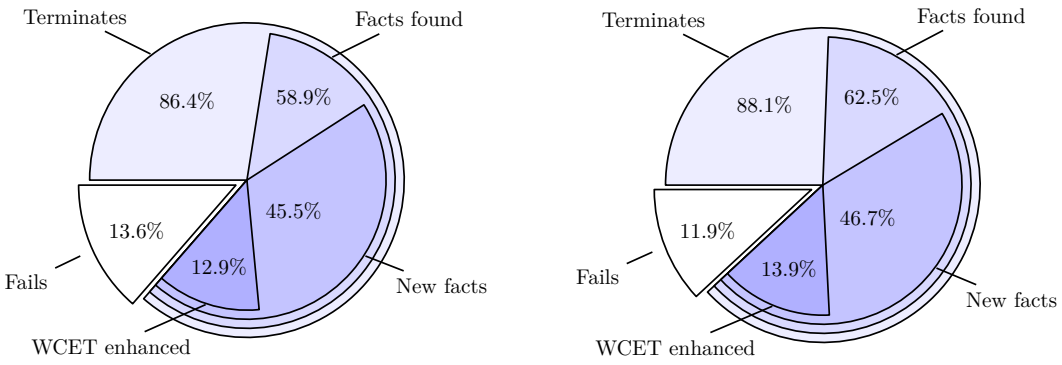
OTAWA hardware model: our goal is not to bench or "stress" OTAWA in terms of hardware. We only want it to give an initial IPET system in which we will insert flow facts discovered via LRA. In order to maximize the number of test benches for which OTAWA gives an initial ILP in reasonable time, we consider a very simple, cache-free, ARM-based architecture.

Misc. CFG transformations: some CFG transformations are necessary, due to limitations of OTAWA (switch statements not supported) and /or PAGAI (multiple return statements). This transformations are performed using the CIL library [34].

Inlining: because the current version of PAGAI has limited support for inter-procedural analysis, function calls are systematically inlined. This transformation is also implemented using the CIL library. This method improves the precision of the analysis, but makes the analysis much more costly in time and memory.

[8] The material necessary for reproducing the experiment is freely available at `https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/reproducible-research/LRA4w7`.

[9] The 5 missing applications are OS and/or architecture dependent.



(a) polyhedra domain **(b)** octagon domain

Figure 7 LRA analysis statistics on 589 functions, for the two relational abstract domains.

Loop bounds: as explained in 3.2.3, our method is intrinsically unable to bound nested loops, so a complementary method is necessary to find loop bounds. For this purpose, we can use oRange, but it appears that the CFG transformations performed using CIL strongly alters its performance[10]. In order to maximize the size of the benchmark we thus systematically exploit, when available, the user pragmas given in source code. Nevertheless, we made a complementary experiment, without using pragmas nor oRange, in order to identify the cases where LRA is sufficient to bound the execution time.

5.3 Lessons learnt

This section presents the lessons learnt form the experiment, by focusing on several points: the ability of the linear analysis to discover "flow facts", and hopefully to enhance the WCET estimation; the influence of the abstract domain on the analysis; the ability of linear analysis to discover loop bounds, and finally the influence of compiler optimizations on traceablity.

5.3.1 Linear analysis and flow facts discovery

When traceability allows it, the constraints discovered by linear analysis are directly translated into *flow facts* giving information on the (im)possible execution paths. These flow facts may be useless if they are redundant with the structural constraints, otherwise they are *new facts*, giving non trivial information on the execution paths. However, even new facts can be useless if they do not concern the worst case execution path. A utility has been developed to check whether the facts discovered by LRA analysis are new or not. Each fact is checked by adding its negation to the set of structural constraints: the fact is redundant if and only if the system becomes infeasible.

Figure 7 gives statistic on the behavior of the LRA method, for the two relational domains (octagon and polyhedra). Let us focus on the polyhedra case first (a). The PAGAI tool terminates for 509 cases out of 589 (86.4%); for the missing cases (13.6%), it runs out of resources in memory or time. Flow facts are found in 347 cases, and at least one fact is new for 268 ones; finally, new facts lead to a WCET improvement for 76 cases. Statistics are similar for the octagon domain, except that it terminates more often: this explains why the WCET is enhanced more often with octagons, even if this domain is less precise.

[10] The CIL tool normalizes the code by using only unbounded *while* and *break* statements, that are badly handled by oRange. However oRange performs well for the original programs, made of human-written *for* loops.

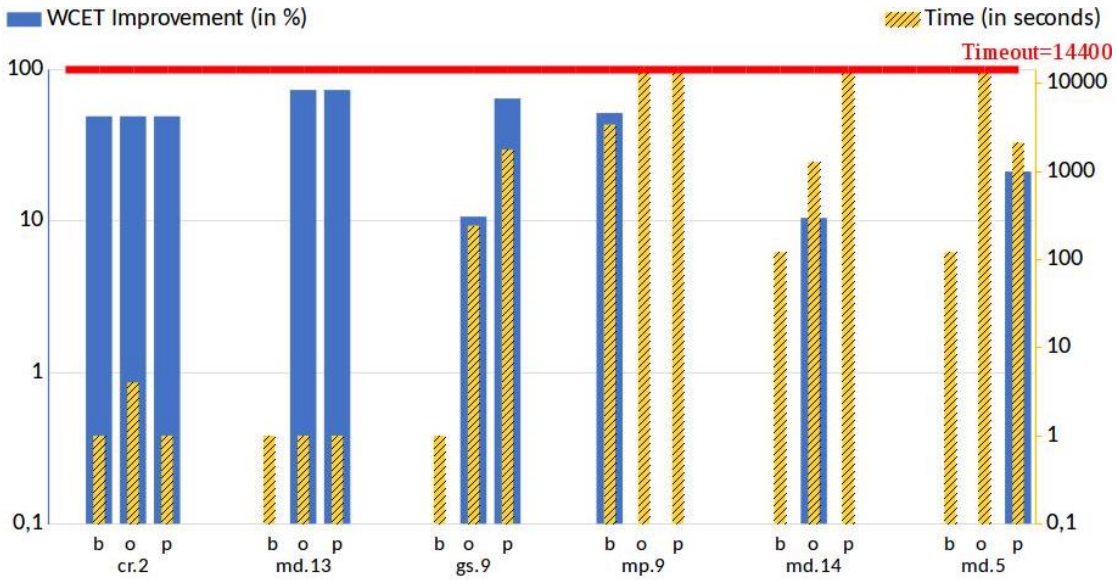


Figure 8 WCET improvement and analysis time depending on abstract domains (b=box, o=octagons, p=polyhedra).

Table 3 Some WCET improvement results (full table page 24).

Ref	Initial WCET	Box			Octagons			Polyhedra		
		Δ	Imp^t	Time	Δ	Imp^t	Time	Δ	Imp^t	Time
cr.2	227K	111K	**48.7**	<1s	111K	**48.7**	4s	111K	**48.7**	1s
md.13	2648	0	0.0	<1s	1920	**72.5**	<1s	1920	**72.5**	<1s
gs.9	6934	0	0.0	1s	738	10.6	4m	4428	**63.8**	29m
an.0	466M	0	0.0	4s	4M	0.8	7m	115M	**24.6**	1m
mp.9	52M	27M	**51.1**	56m	-	-	-	-	-	-
md.14	51K	0	0.0	2m	5K	**10.4**	21m	-	-	-
md.5	13M	0	0.0	2m	-	-	-	3M	**21.0**	35m

A possible conclusion is that LRA, when it works, is actually good at finding non redundant semantic facts (more than half of the time, when it terminates), but that those facts do not necessarily lead to a WCET improvement (about 15% of the termination cases).

5.3.2 Abstract domains

The main goal of the experiment is to observe the influence of the linear analysis on the WCET estimation. The linear analysis performed by PAGAI is parameterized by the choice of an abstract domain to represent the possible values of the counters. Two domains proposed by PAGAI are *relational*, and thus are likely to express relations between our counters and the original variables in the programs:

- The polyhedra domain is the most precise since it can handle any linear relation, and its algorithmic cost is exponential in the worst case.
- The octagon domain handles intervals and bounded pairwise sums or differences. It is less precise, but has a polynomial cost in the worst case: $O(n^3)$ in time, and $O(n^2)$ in space.

To be exhaustive, we also consider the domain *box*, which handles only intervals. Since this domain is non-relational, it is intrinsically unable to relate our additional counters to the program variables. The flow facts that can be discovered with the box domain are thus limited (basically, counters stuck down to zero, which correspond to dead code).

The WCET estimation is improved by at least one domain for 90 functions. The gain ranges from negligible (0.1%) to interesting (around 10%) or even huge (more than 50%). We limit here the comments to the cases where the enhancement is greater 0.8%. The detailed results for these 60 cases are given in appendix (table 6, page 24), and a selection of typical cases is given in table 3.

The experiment gives some interesting information:

- The interest of the *box domain* is very limited: it is an indirect way of performing constant propagation and dead code "pruning". Most of the time it gives no improvement (42 out of 60, e.g., md.13, gs.9). However, since it is the cheapest domain, it may give results when other domains fail (6 times, e.g., mp.9).
- When both *octagons* and *polyhedra* terminate, they often give the same result (34 out of 60 cases, e.g., cr.2, md.13). However there are some cases (12 out of 60, e.g., gs.9), where the expressiveness of polyhedra is actually useful (constraint involving 3 or more variables, and pairwise relations with non unit coefficients).
- In compliance with the theoretical complexity, *octagons* may terminates while *polyhedra* fails (7 cases, e.g., md.14). Nevertheless, there is also one case where *octagons* fail while *polyhedra* works (md.5). This is due to the fact that the cost of octagons is almost always cubic in the number of variables, while the exponential cost of polyhedra is rarely reached in practice.

5.3.3 Loop bounds

LRA is intrinsically limited to the discovery of single loop bounds (cf. 3.2.3). We made a complementary experiment to check if and when LRA actually finds such loops. For this experiment, we only consider the short-list of programs from Table 6 where PAGAI terminates when using a relational domain (octagon or polyhedra); as a matter of fact, using the *box* domain is irrelevant since it can't find any loop bound other than 0.

For these 54 programs, we have:

- computed the *loop level*, which is maximal depth of nested loops appearing in the program (0: no loop at all, 1: only single loops, 2 or more: nested loops);
- launched our tool without using oRange nor user-pragmas. The LRA analysis is performed twice: with the octagon and the polyhedra domain, and we keep only the best result.

Table 7 (page 25) lists the results; the column "pagai" simply indicates if the analysis give a bounded WCET, since the WCET value is, in this case, the same as the one in Table 6.

There are 10 test cases that are loop-free, and thus with no bounds to found. There are 25 programs with only single loops (level=1); these are the cases where PAGAI is supposed to find bounds, and it actually does it for most of the cases (19 out of 25). In fact, PAGAI finds the bounds for all loops that are semantically guarded by a counter condition, that is, *for* loops or equivalent. The cases where PAGAI does not find bounds are those where the loop is guarded by a *points-to* condition (e.g., `while (*p++)`).

We expected PAGAI not to bound any program with a loop level greater than 1, which is the case except for one program (ex.2). In fact this example is a "false counter-example": the loop depth is *syntactically* 2, but the inner-loop appears in a branch which is never executed. The loop depth is then *semantically* 1.

Table 4 Impact of compiler optimizations on WCET and LRA

	O0			CO						
Ref	Initial WCET	Best WCET	Best Imp^t	Opt. speedup	Initial WCET	Best WCET	Best Imp^t	Traceability		
md.13	2648	728	72.5	3.3x	791	215	72.8	100%	of	2
an.0	466M	351M	24.6	3.0x	157M	116M	25.9	100%	of	44
cr.2	227K	116K	48.7	2.3x	97K	50K	48.7	41%	of	24
md.5	13M	10M	21.0	3.4x	4M	3M	15.0	80%	of	40
ex.2	278K	224K	19.2	1.3x	218K	218K	0.0	46%	of	13

5.3.4 Optimization level and traceability

The main focus of this work is the influence of linear analysis on the precision of the WCET estimation. Nevertheless, since analysis is performed at the C level, the problem of the traceability between the C and the binary code must be considered. Forbidding any optimization is not an option in real-time domain. We argue that a well-chosen set of optimizations can lead to a reasonable compromise between traceability and program speed-up.

For all functions that give some enhancement on the non-optimized code, we run the experiments using the custom optimization (CO) level defined in 4.4. Since the counter analysis is completely independent to the compilation method, the linear relations found are the same, and the ability to enhance the WCET estimation is only due to traceability.

The detailed results of this experiment are given in appendix (table 8, page 26), and a selection of typical cases is given in table 4. The table gathers the results obtained with the non-optimized binary code O0, and the optimized one CO. For each optimization level, the table gives:

- the *Initial WCET*, in CPU cycles, computed by OTAWA,
- the *Best WCET*, enhanced thanks to the properties discovered with PAGAI, with some abstract domain,
- the corresponding **Improvement** percentage.

The table also shows the *Optimization speed-up*, which is the ratio between the initial O0 and the initial CO estimation, i.e., it measures the gain obtained just because of the compilation, before applying the counter method. Finally, for CO compilation, the table gives an information on the *Traceability*: the percentage of counters introduced for LRA at C level, that are actually associated to some basic block, at binary level. Traceability in the O0 mode is not shown in the table as it is always 100%.

The interesting information given by the experiment are:

- Even if the CO level is very limited (subset of O1 level, and a fortiori of O2), it generates a fairly optimized code: the speed-up is mostly between 2x and 4x.
- In most of the cases (53 out of 60) traceability is 100%, and one can observe an enhancement due to LRA similar to the one obtained with O0 code. Indeed, this improvement is obtained on the CO initial WCET, which is already much smaller than the one obtained for the non-optimized code (e.g., md.13, an.0).
- In some cases, traceability is partly lost, but remain sufficient to enhance the estimation (4 cases, e.g., cr.2, md.5).
- Finally, for 3 cases, partial traceability leads to no enhancement (e.g., ex.2).

6 Conclusion and future work

Linear Relation Analysis is a powerful technique to discover invariant linear relations between numerical variables of a program. On the other hand, the classical evaluation of WCET using Implicit Path Enumeration Technique is based on expressing the WCET as the solution of an Integer Linear Program, the variables of which are counters associated with the basic blocks of the program. So, the idea of adding these counters as auxiliary variables in the program, and using the results of LRA as semantic flow-facts to be added to the ILP, is rather natural. Our goal, in this paper, was to conduct a light-weight experiment – by combining existing tools – to evaluate the benefits of the approach. Secondarily, such an experiment raised the question of traceability, since semantic flow-facts are discovered on the source program, while the WCET is evaluated on the executable code. The conclusion of this experiment on public benchmarks is manyfold:

- LRA finds new semantic facts in many examples (46%), but many of these new facts do not influence the evaluated WCET. However, the WCET is improved on a significant subset (almost 14%) of the examples, and the improvement is often interesting.
- the traceability problems can be safely dealt with, using the debugging information provided by the compiler; this is the case even in the presence of strong compiling optimizations, as long as these optimizations do not modify too much the control structure of the program.

This work could be continued in several directions.

- It would be interesting to limit the number of counters, as the cost of LRA can be exponential in the number of variables. Of course, counters which are structurally related by flow equations can be saved, but their cost is low in polyhedra computations (they are linked to each other by equations). An appealing idea would be to introduce counters on the branches of a conditional, only when these branches appear to have strongly different execution times, a measure that is roughly available after the micro-architectural analysis [43].
- Existing LRA analyzers (like PAGAI) are generally not inter-procedural, which forced us to inline the procedures in our experiments. An inter-procedural version of LRA must be studied to solve this problem. The *relational* nature of LRA is surely an advantage, since a procedure can be associated a summary as an input-output relation. Summaries of called procedures can then be used in the caller, in a bottom-up fashion.
- Traceability is still a concern, which would benefit from a better cooperation of the compiler [25].

References

1 Mihail Asavoae, Claire Maiza, and Pascal Raymond. Program Semantics in Model-Based WCET Analysis: A State of the Art Perspective. In *13th International Workshop on Worst-Case Execution Time Analysis, WCET 2013, July 9, 2013, Paris, France*, volume 30 of *OASICS*, pages 32–41. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.

2 Roberto Bagnara, Elisa Ricci, Enea Zaffanella, and Patricia M. Hill. Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library. In M. V. Hermenegildo and G. Puebla, editors, *9th International Symposium on Static Analysis, SAS'02*, Madrid, Spain, September 2002. LNCS 2477. doi:10.1007/3-540-45789-5_17.

3 Gogul Balakrishnan and Thomas W. Reps. DIVINE: DIscovering variables IN executables. In *Verification, Model Checking, and Abstract Interpretation, VMCAI 2007*, pages 1–28, Nice, France, January 2007.

4 Gogul Balakrishnan, Thomas W. Reps, David Melski, and Tim Teitelbaum. WYSINWYX: what you see is not what you execute. In *Verified Software: Theories, Tools, Experiments, VSTTE 2005*, pages 202–213, Zurich, Switzerland, October 2005.

5 Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat. OTAWA: An open toolbox for adaptive WCET analysis. In *SEUS*, 2010.

6 Duc-Hiep Chu, Joxan Jaffar, and Rasool Maghareh. Precise Cache Timing Analysis via Symbolic Execution. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, 2016.

7 Patrick Cousot and Radia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

8 Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *5th ACM Symposium on Principles of Programming Languages, POPL'78*, Tucson (Arizona), January 1978.

9 Marianne de Michiel, Armelle Bonenfant, Hugues Cassé, and Pascal Sainrat. Static loop bound analysis of C programs based on flow analysis and abstract interpretation. In *IEEE Int'l Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2008.

10 Sun Ding, Hee Beng Kuan Tan, and Kaiping Liu. A Survey of Infeasible Path Detection. In *Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012), Wroclaw, Poland, 29-30 June, 2012.*, pages 43–52, 2012.

11 Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sorensen, Peter Wägemann, and Simon Wegener. TACLeBench: A Benchmark Collection to Support Worst-Case Execution Time Research. In *16th International Workshop on Worst-Case Execution Time Analysis, WCET 2016, July 5, 2016, Toulouse, France*, pages 2:1–2:10, 2016.

12 Paul Feautrier and Laure Gonnord. Accelerated Invariant Generation for C Programs with Aspic and C2fsm. In *Tools for Automatic Program AnalysiS (TAPAS)*, Perpignan, France, September 2010.

13 Christian Ferdinand, Florian Martin, Christoph Cullmann, Marc Schlickling, Ingmar Stein, Stephan Thesing, and Reinhold Heckmann. New Developments in WCET Analysis. In *Program Analysis and Compilation*, pages 12–52, 2006.

14 Denis Gopan and Thomas Reps. Lookahead widening. In *CAV'06*, Seattle, 2006.

15 Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET Benchmarks: Past, Present And Future. In *Proc. of WCET*, pages 136–146, 2010.

16 Jan Gustafsson, Andreas Ermedahl, Christer Sandberg, and Björn Lisper. Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution. In *RTSS*, 2006.

17 Nicolas Halbwachs, Yann-Eric Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

18 Julien Henry, Mihail Asavoae, David Monniaux, and Claire Maiza. How to compute worst-case execution time by optimization modulo theory and a clever encoding of program semantics. In *SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems 2014, LCTES '14*, pages 43–52, June 2014.

19 Julien Henry, David Monniaux, and Matthieu Moy. PAGAI: A Path Sensitive Static Analyser. *Electr. Notes Theor. Comput. Sci.*, 289:15–25, 2012.

20 François Irigoin, Pierre Jouvelot, and Rémy Triolet. Semantical Interprocedural parallelization: An overview of the PIPS Project. In *ACM Int. Conf. on Supercomputing, ICS'91, Köln*, 1991.

21 Bertrand Jeannet and Antoine Miné. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *Computer Aided Verification (CAV 2009), Grenoble, France*, pages 661–667, June 2009.

22 Raimund Kirner, Peter Puschner, and Adrian Prantl. Transforming flow information during code optimization for timing analysis. *Journal on Real-Time Systems*, 45(1-2), 2010.

23 Jens Knoop, Laura Kovács, and Jakob Zwirchmayr. WCET squeezing: on-demand feasibility refinement for proven precise WCET-bounds. In *Proceedings of the 21st International Conference on Real-Time Networks and Systems*, pages 161–170. ACM, 2013.

24 Chris Lattner and Vikram Adve. LLVM: a compilation framework fopr lifelong program analysis & transformation. In *CGO'04*, pages 75–86, Washington, DC, August 2004. IEEE Computer Society.

25 Hanbing Li, Isabelle Puaut, and Erven Rohou. Traceability of Flow Information: Reconciling Compiler Optimizations and WCET Estimation. In *22nd International Conference on Real-Time Networks and Systems, RTNS'14, Versailles, France, October 8-10, 2014*, 2014.

26 Hanbing Li, Isabelle Puaut, and Erven Rohou. Tracing Flow Information for Tighter WCET Estimation: Application to Vectorization. In *21st IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, page 10, Hong-Kong, China, August 2015. URL: `https://hal.inria.fr/hal-01177902`.

27 Xianfeng Li, Liang Yun, Tulika Mitra, and Abhik Roychoudhury. Chronos: A timing analyzer for embedded software. *Sci. Comput. Program.*, 69(1-3):56–67, 2007.

28 Yau-Tsun Steven Li and Sharad Malik. Performance analysis of embedded software using implicit path enumeration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(12), 1997.

29 Björn Lisper. SWEET – a tool for WCET flow analysis. In *6th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISOLA)*, October 2014.

30 Paul Lokuciejewski, Daniel Cordes, Heiko Falk, and Peter Marwedel. A Fast and Precise Static Loop Analysis Based on Abstract Interpretation, Program Slicing and Polytope Models. In *Proceedings of the CGO 2009, The Seventh International Symposium on Code Generation and Optimization*, pages 136–146, Seattle, Washington, USA, March 2009.

31 Paul Lokuciejewski and Peter Marwedel. *Worst-Case Execution Time Aware Compilation Techniques for Real-Time Systems.* Springer, 2011. `doi:10.1007/978-90-481-9929-7`.

32 Ravindra Metta, Martin Becker, Prasad Bokil, Samarjit Chakraborty, and R. Venkatesh. TIC: a scalable model checking based approach to WCET estimation. In *Proceedings of the 17th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, Tools, and Theory for Embedded Systems, LCTES 2016, Santa Barbara, CA, USA, June 13 - 14, 2016*, pages 72–81, 2016. `doi:10.1145/2907950.2907961`.

33 Antoine Miné. The Octagon Abstract Domain. In *Proceedings of the Eighth Working Conference on*

Reverse Engineering, WCRE'01, Stuttgart, Germany, October 2-5, 2001, page 310, 2001. doi:10.1109/WCRE.2001.957836.

34 George C. Necula, Scott McPeak, Shree P. Rahul, and Westley Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. In R. Nigel Horspool, editor, *Compiler Construction*, pages 213–228, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

35 Pascal Raymond, Claire Maiza, Catherine Parent-Vigouroux, Fabienne Carrier, and Mihail Asavoae. Timing analysis enhancement for synchronous program. *Real-Time Systems*, pages 1–29, 2015.

36 Jordy Ruiz and Hugues Cassé. Using SMT Solving for the Lookup of Infeasible Paths in Binary Programs (regular paper). In *Workshop on Worst-Case Execution Time Analysis, Lund, Sweden, 07/07/2015*, pages 95–104. OASICs, Dagstuhl Publishing, July 2015.

37 Thomas Sewell, Felix Kam, and Gernot Heiser. Complete, High-Assurance Determination of Loop Bounds and Infeasible Paths for WCET Analysis. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*, pages 185–195, 2016. doi:10.1109/RTAS.2016.7461326.

38 Björn Lisper Stefan Bygde, Andreas Ermedahl. An Efficient Algorithm for Parametric WCET Calculation. *Journal of Systems Architecture*, 57(6):614–624, May 2011.

39 Vivy Suhendra, Tulika Mitra, Abhik Roychoudhury, and Ting Chen. Efficient detection and exploitation of infeasible paths for software timing analysis. In *DAC*, pages 358–363, 2006.

40 Stephan Thesing, Jean Souyris, Reinhold Heckmann, Famantanantsoa Randimbivololona, Marc Langenbach, Reinhard Wilhelm, and Christian Ferdinand. An Abstract Interpretation-Based Timing Validation of Hard Real-Time Avionics Software. In *DSN*, pages 625–632, 2003.

41 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embedded Comput. Syst. (TECS)*, 7(3), 2008.

42 Jakob Zwirchmayr, Armelle Bonenfant, Marianne de Michiel, Hugues Cassé, Laura Kovács, and Jens Knoop. FFX: A portable WCET annotation language (regular paper). In *International Conference on Real-Time and Network Systems (RTNS), Pont-à-Mousson, 08/11/2012-09/11/2012*, pages 91–100, November 2012.

43 Jakob Zwirchmayr, Pascal Sotin, Armelle Bonenfant, Denis Claraz, and Philippe Cuenot. Identifying Relevant Parameters to Improve WCET Analysis (regular paper). In *Workshop on Worst-Case Execution Time Analysis, Madrid, 08/07/2014*, pages 91–100. OASICs, Dagstuhl Publishing, July 2014.

A Experiment Results

The material necessary for reproducing the experiment presented here is freely available at `https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/reproducible-research/LRA4w7`.

Experiment was performed on 589 individual C functions extracted from the TACLeBench [11]. An improvement of the WCET estimation is observed for 90 functions (15% of the cases). This section details the results for the 60 cases where the improvement is greater than 0.8%.

Table 5 contains label definitions to ease and shorten the reference to the bench functions: the label (column 1), the source folder in the TACLeBench (column 2), and the function name (column 3).

Table 6 contains the experiment results using the gcc `-O0` compilation level. The first column holds the function label, and the second one holds the initial WCET estimation computed by OTAWA. The remaining columns hold information related to the improvement obtained (or not) with Linear relation analysis, using 3 different abstract domains: boxes (intervals), octagons and polyhedra. For each domain, the table gives the improvement in number of cycles (Δ) and percentage (Imp^t), and the time necessary to perform the LRA with PAGAI[11]. Numbers in bold highlight the best improvements among various methods (box, octagons, polyhedra). Empty cells ('-') mean that the corresponding case triggered the 2 hours timeout set for the experiment.

Table 7 gives information on the ability of PAGAI to discover loop bounds ; to obtain this table, the experiments are re-played without the help of any external method (neither oRange nor the user-given pragmas). For each program, the table gives its loop level (maximal depth of nested loops) and indicates wheter PAGAI finds a bounded WCET or not.

Finally, table 8 aims at observing the impact of compiler optimization on WCET estimation in general, and our method in particular. We consider two optimization levels: the standard `-O0` (no optimization at all), and the ad hoc customized `-O1` level (designed to limit CFG transformation and maximize traceability). Since the LRA analysis is performed at the C level, the flow facts discovered are the same whatever is the optimization level. A lack of improvement in the case of optimized code is then necessarily due to an "imperfect" traceability.

The first group of columns recalls the results optained with `-O0`; it only gives the best result, obtained for some abstract domain (refer to Table 6 for details). The second group gives information on the optimized code:

- the initial WCET estimation given by OTAWA, together with the corresponding *speed-up* factor which indicates how "faster" is the optimized code compared to the non-optimized one;
- the best WCET estimation (together with the improvement percentage) optained using PAGAI;
- the *traceability* ratio indicates how many counters introduced by our method are actually associated to some basic block in the binary code. With a traceability of 100%, we expect to observe an improvement percentage of the same order than the one obtained on the non-optimized code. Note that the traceability with the non-optimized code is not given since it is always 100%.

[11] Results obtained on an Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz

Table 5 TacleBench functions Reference Labels.

Ref	Directory	Function Names
ad.6	sequential/adpcm_dec	adpcm_dec_logsch
ad.7	sequential/adpcm_dec	adpcm_dec_logscl
ad.14	sequential/adpcm_dec	adpcm_dec_uppol2
ae.7	sequential/adpcm_enc	adpcm_enc_logsch
ae.8	sequential/adpcm_enc	adpcm_enc_logscl
ae.10	sequential/adpcm_enc	adpcm_enc_quantl
ae.16	sequential/adpcm_enc	adpcm_enc_uppol2
am.12	sequential/ammunition	ammunition_bit_string_set
am.17	sequential/ammunition	ammunition_divide_unsigned_integer
am.18	sequential/ammunition	ammunition_divide_unsigned_integer_without_overflow
am.47	sequential/ammunition	ammunition_multiply_integer
am.49	sequential/ammunition	ammunition_multiply_unsigned_integer
am.50	sequential/ammunition	ammunition_multiply_unsigned_integer_without_overflow
am.68	sequential/ammunition	ammunition_unsigned_integer_remainder
an.0	sequential/anagram	anagram_AddWords
an.2	sequential/anagram	anagram_BuildWord
an.8	sequential/anagram	anagram_init
an.14	sequential/anagram	anagram_ReadDict
an.15	sequential/anagram	anagram_Reset
an.16	sequential/anagram	anagram_return
bs.0	kernel/bsort	bsort_BubbleSort
bs.3	kernel/bsort	bsort_main
bs.4	kernel/bsort	bsort_return
bs.5	kernel/bsort	main
cr.2	crc	main
du.2	test/duff	duff_init
du.5	test/duff	main
ex.2	expint	main
gd.4	sequential/gsm_dec	gsm_dec_Coefficients_0_12
gd.5	sequential/gsm_dec	gsm_dec_Coefficients_13_26
gd.6	sequential/gsm_dec	gsm_dec_Coefficients_27_39
gd.11	sequential/gsm_dec	gsm_dec_Decoding_of_the_coded_Log_Area_Ratios
gd.16	sequential/gsm_dec	gsm_dec_Postprocessing
ge.11	sequential/gsm_encode	Gsm_Preprocess
ge.13	sequential/gsm_encode	Gsm_Short_Term_Analysis_Filter
gs.2	sequential/g723_enc	g723_enc_fmult
gs.8	sequential/g723_enc	g723_enc_predictor_pole
gs.9	sequential/g723_enc	g723_enc_predictor_zero
gs.10	sequential/g723_enc	g723_enc_quan
gs.11	sequential/g723_enc	g723_enc_quantize
gs.16	sequential/g723_enc	g723_enc_update
hd.1	sequential/h264_dec	h264_dec_init
lc.0	lcdnum	main
li.3	app/lift	lift_controller
li.7	app/lift	lift_ctrl_set_vals
md.3	kernel/md5	md5_final
md.5	kernel/md5	md5_InitRandomStruct
md.13	kernel/md5	md5_R_RandomInit
md.14	kernel/md5	md5_R_RandomUpdate
md.15	kernel/md5	md5_transform
md.16	kernel/md5	md5_update
mp.9	sequential/mpeg2	mpeg2_frame_estimate
mp.11	sequential/mpeg2	mpeg2_fullsearch
sh.2	kernel/sha	sha_final
sm.0	sequential/statemate	main
sm.1	sequential/statemate	statemate_FH_DU
sm.2	sequential/statemate	statemate_generic_BLOCK_ERKENNUNG_CTRL
sm.3	sequential/statemate	statemate_generic_EINKLEMMSCHUTZ_CTRL
sm.4	sequential/statemate	statemate_generic_FH_TUERMODUL_CTRL
sm.8	sequential/statemate	statemate_main

Table 6 How LRA can improve the estimated WCET of TacleBench.

	Initial	Box			Octagons			Polyhedra		
Ref	WCET	Δ	Imp^t	Time	Δ	Imp^t	Time	Δ	Imp^t	Time
md.13	2648	0	0.0	<1s	1920	**72.5**	<1s	1920	**72.5**	<1s
an.15	173K	0	0.0	<1s	121K	**69.9**	1s	121K	**69.9**	<1s
gs.2	1105	0	0.0	<1s	738	**66.7**	9s	738	**66.7**	4s
hd.1	2092K	0	0.0	<1s	1371K	**65.5**	<1s	1371K	**65.5**	<1s
gs.8	2268	0	0.0	<1s	1476	**65.0**	1m	1476	**65.0**	1m
gs.9	6934	0	0.0	1s	738	10.6	4m	4428	**63.8**	29m
du.2	19K	0	0.0	<1s	12K	**60.1**	<1s	12K	**60.1**	<1s
du.5	22K	0	0.0	<1s	12K	**53.8**	<1s	12K	**53.8**	<1s
mp.9	52M	27M	**51.1**	56m	-	-	-	-	-	-
mp.11	10M	5M	**51.1**	2m	-	-	-	-	-	-
cr.2	227K	111K	**48.7**	<1s	111K	**48.7**	4s	111K	**48.7**	1s
lc.0	1540	0	0.0	<1s	595	**38.6**	<1s	595	**38.6**	<1s
md.15	8600	0	0.0	<1s	2304	**26.7**	<1s	2304	**26.7**	<1s
an.2	235K	0	0.0	<1s	58K	**24.8**	16s	58K	**24.8**	2s
an.0	466M	0	0.0	4s	4M	0.8	7m	115M	**24.6**	1m
md.5	13M	0	0.0	2m	-	-	-	3M	**21.0**	35m
ex.2	278K	1K	0.1	4s	28K	9.9	23s	53K	**19.2**	6s
sm.3	87	16	**18.3**	<1s	16	**18.3**	<1s	16	**18.3**	<1s
md.3	34K	0	0.0	18s	6K	17.2	6m	6K	**18.2**	31m
md.16	13K	0	0.0	7s	2K	**17.7**	19s	2K	**17.7**	10s
sm.0	268K	46K	**17.1**	16s	-	-	-	-	-	-
gs.10	942	0	0.0	<1s	126	**13.3**	1s	126	**13.3**	<1s
am.47	3649	0	0.0	13m	76	2.0	14m	485	**13.2**	14m
gs.11	2020	0	0.0	1s	252	**12.4**	17s	252	**12.4**	4s
md.14	51K	0	0.0	2m	5K	**10.4**	21m	-	-	-
sm.4	595	62	**10.4**	3s	62	**10.4**	2m	62	**10.4**	72m
li.7	1088	0	0.0	<1s	102	**9.3**	2s	102	**9.3**	<1s
gs.16	3760	0	0.0	7s	254	**6.7**	63m	-	-	-
ad.6	66	0	0.0	<1s	4	**6.0**	<1s	4	**6.0**	<1s
ae.7	66	0	0.0	<1s	4	**6.0**	<1s	4	**6.0**	<1s
ad.7	67	0	0.0	<1s	4	**5.9**	<1s	4	**5.9**	<1s
ae.8	67	0	0.0	<1s	4	**5.9**	<1s	4	**5.9**	<1s
sm.1	266K	14K	**5.1**	21s	-	-	-	-	-	-
sm.8	266K	14K	**5.1**	26s	-	-	-	-	-	-
an.16	530	0	0.0	<1s	25	**4.7**	<1s	25	**4.7**	<1s
am.50	2251	0	0.0	15m	76	3.3	15m	95	**4.2**	15m
am.49	2281	0	0.0	9m	76	3.3	14m	95	**4.1**	12m
sm.2	248	10	**4.0**	<1s	10	**4.0**	1s	10	**4.0**	1s
bs.4	5580	0	0.0	<1s	196	**3.5**	<1s	196	**3.5**	<1s
am.12	468	0	0.0	12m	16	**3.4**	12m	16	**3.4**	12m
ad.14	120	0	0.0	<1s	4	**3.3**	<1s	4	**3.3**	<1s
ae.16	120	0	0.0	<1s	4	**3.3**	<1s	4	**3.3**	<1s
li.3	3405	0	0.0	11s	102	**2.9**	47m	-	-	-
ae.10	1473	0	0.0	<1s	33	**2.2**	<1s	33	**2.2**	<1s
ge.11	47K	0.16K	0.3	1s	1K	**2.0**	1m	1K	**2.0**	16s
am.68	12K	0	0.0	10m	0.15K	**1.3**	73m	-	-	-
gd.16	23K	0.32K	**1.3**	<1s	0.32K	**1.3**	2s	0.32K	**1.3**	<1s
gd.4	1333	16	**1.2**	1s	16	**1.2**	6s	16	**1.2**	1s
gd.6	1333	16	**1.2**	<1s	16	**1.2**	3s	16	**1.2**	<1s
sh.2	25K	0.31K	**1.2**	6s	0.31K	**1.2**	3m	0.31K	**1.2**	1m
an.8	3079K	0	0.0	<1s	0	0.0	7s	34K	**1.1**	1s
an.14	3079K	0	0.0	<1s	0	0.0	5s	34K	**1.1**	1s
gd.11	1420	16	**1.1**	1s	16	**1.1**	35m	-	-	-
ge.13	727K	8K	**1.0**	1m	-	-	-	-	-	-
bs.0	1045K	0	0.0	<1s	0	0.0	2s	10K	**0.9**	<1s
bs.3	1045K	0	0.0	<1s	0	0.0	1s	10K	**0.9**	<1s
bs.5	1053K	0	0.0	<1s	0.20K	0.0	7s	10K	**0.9**	1s
gd.5	837	8	**0.9**	<1s	8	**0.9**	1s	8	**0.9**	<1s
am.17	8930	0	0.0	9m	76	**0.8**	20m	-	-	-
am.18	8901	0	0.0	11m	76	**0.8**	20m	-	-	-

Table 7 Loop bounds discovery, using PAGAI without the help of oRange nor the user-given bounds. This experiment is performed for the 54 cases from Table 6 where PAGAI terminates with either octagons or polyhedra ; the box domain is unable to find loop bounds and is not considered here. Within this test set, 10 programs contain no loop and are thus trivially bounded (adVI, adVII, adXIV, aeVII, aeVIII, aeXVI, gdXI, smII, smIII, smIV). For the remaining programs, first column gives the depth of nested loops and column two indicates if PAGAI gives a bounded (i.e., finite) WCET estimation. The WCET value is not given: it corresponds to the best PAGAI estimation in Table 6. The "paradoxal" result for ex.2 (loop depth 2 and bounded) is due to the fact that PAGAI "bounds" the inner-loop to 0 (i.e., the loop appears in a infeasible branch).

	loop depth	PAGAI
ae.10	1	**bounds**
bs.4	1	**bounds**
gd.4	1	**bounds**
gd.5	1	**bounds**
gd.6	1	**bounds**
gd.16	1	**bounds**
ge.11	1	**bounds**
gs.2	1	**bounds**
gs.8	1	**bounds**
gs.16	1	**bounds**
lc.0	1	**bounds**
li.7	1	**bounds**
md.15	1	**bounds**
du.2	1	**bounds**
du.5	1	**bounds**
hd.1	1	**bounds**
md.13	1	**bounds**
an.15	1	**bounds**
an.16	1	**bounds**
am.12	1	$\top$
an.2	1	$\top$
gs.10	1	$\top$
gs.11	1	$\top$
li.3	1	$\top$
sh.2	1	$\top$

	loop depth	PAGAI
ex.2	2	**bounds**
am.47	2	$\top$
am.49	2	$\top$
am.50	2	$\top$
an.8	2	$\top$
an.14	2	$\top$
bs.0	2	$\top$
bs.3	2	$\top$
bs.3	2	$\top$
cr.2	2	$\top$
gs.9	2	$\top$
md.3	2	$\top$
md.14	2	$\top$
md.16	2	$\top$
am.17	3	$\top$
am.18	3	$\top$
am.68	3	$\top$
an.0	3	$\top$
md.5	4	$\top$

Table 8 Observing the impact of compilation levels on LRA.

	O0			CO						
Ref	Initial WCET	Best WCET	Best Imp^t	Opt. speedup	Initial WCET	Best WCET	Best Imp^t	Traceability		
md.13	2648	728	72.5	3.3x	791	215	72.8	100%	of	2
an.15	173K	52K	69.9	3.0x	58K	17K	69.9	100%	of	6
gs.2	1105	367	66.7	2.3x	479	171	64.3	100%	of	14
hd.1	2092K	721K	65.5	2.1x	1019K	350K	65.6	100%	of	4
gs.8	2268	792	65.0	2.4x	963	347	63.9	100%	of	28
gs.9	6934	2506	63.8	2.4x	2910	1062	63.5	100%	of	28
du.2	19K	8K	60.1	2.3x	8K	3K	64.0	100%	of	2
du.5	22K	10K	53.8	2.4x	9K	4K	59.3	100%	of	3
mp.9	52M	25M	51.1	4.6x	11M	11M	0.0	79%	of	890
mp.11	10M	5M	51.1	4.6x	2M	2M	0.0	79%	of	178
cr.2	227K	116K	48.7	2.3x	97K	50K	48.7	41%	of	24
lc.0	1540	945	38.6	2.4x	641	421	34.3	100%	of	4
md.15	8600	6296	26.7	2.8x	3064	2200	28.1	100%	of	2
an.2	235K	176K	24.8	2.9x	80K	59K	25.9	100%	of	21
an.0	466M	351M	24.6	3.0x	157M	116M	25.9	100%	of	44
md.5	13M	10M	21.0	3.4x	4M	3M	15.0	80%	of	40
ex.2	278K	224K	19.2	1.3x	218K	218K	0.0	46%	of	13
sm.3	87	71	18.3	1.1x	78	59	24.3	100%	of	5
md.3	34K	28K	18.2	2.7x	13K	10K	17.1	100%	of	23
md.16	13K	11K	17.7	2.6x	5K	4K	17.2	100%	of	10
sm.0	268K	222K	17.1	1.1x	237K	193K	18.8	100%	of	108
gs.10	942	816	13.3	2.5x	381	325	14.6	100%	of	4
am.47	3649	3164	13.2	2.6x	1417	1282	9.5	100%	of	26
gs.11	2020	1768	12.4	2.4x	830	718	13.4	100%	of	11
md.14	51K	46K	10.4	2.7x	19K	17K	10.8	97%	of	37
sm.4	595	533	10.4	1.1x	547	493	9.8	100%	of	42
li.7	1088	986	9.3	2.1x	516	468	9.3	100%	of	9
gs.16	3760	3506	6.7	2.3x	1653	1539	6.8	100%	of	69
ad.6	66	62	6.0	3.0x	22	20	9.0	100%	of	3
ae.7	66	62	6.0	3.0x	22	20	9.0	100%	of	3
ad.7	67	63	5.9	2.9x	23	21	8.6	100%	of	3
ae.8	67	63	5.9	2.9x	23	21	8.6	100%	of	3
sm.1	266K	252K	5.1	1.1x	237K	224K	5.1	100%	of	97
sm.8	266K	252K	5.1	1.1x	237K	224K	5.1	100%	of	96
an.16	530	505	4.7	1.7x	316	299	5.3	100%	of	3
am.50	2251	2156	4.2	2.6x	860	823	4.3	100%	of	8
am.49	2281	2186	4.1	2.6x	865	839	3.0	100%	of	9
sm.2	248	238	4.0	1.1x	230	220	4.3	100%	of	11
bs.4	5580	5384	3.5	1.9x	2883	2687	6.7	100%	of	5
am.12	468	452	3.4	2.6x	181	177	2.2	100%	of	13
ad.14	120	116	3.3	2.6x	46	44	4.3	100%	of	8
ae.16	120	116	3.3	2.6x	46	44	4.3	100%	of	8
li.3	3405	3303	2.9	1.6x	2093	2045	2.2	100%	of	57
ae.10	1473	1440	2.2	2.1x	706	690	2.2	100%	of	6
ge.11	47K	46K	2.0	2.5x	19K	18K	5.0	100%	of	24
am.68	12K	12K	1.3	1.2x	9K	9K	0.5	100%	of	52
gd.16	23K	23K	1.3	2.3x	10K	10K	3.1	100%	of	12
gd.4	1333	1317	1.2	2.4x	553	537	2.8	100%	of	12
gd.6	1333	1317	1.2	2.4x	553	537	2.8	100%	of	12
sh.2	25K	24K	1.2	2.2x	11K	11K	0.9	76%	of	39
an.8	3079K	3045K	1.1	2.3x	1333K	1310K	1.7	100%	of	14
an.14	3079K	3045K	1.1	2.3x	1333K	1310K	1.7	100%	of	15
gd.11	1420	1404	1.1	2.4x	601	585	2.6	100%	of	121
ge.13	727K	719K	1.0	1.9x	377K	369K	2.0	100%	of	289
bs.0	1045K	1035K	0.9	2.7x	389K	385K	0.9	100%	of	6
bs.3	1045K	1035K	0.9	2.7x	389K	385K	0.9	100%	of	5
bs.5	1053K	1043K	0.9	2.7x	393K	389K	1.0	100%	of	10
gd.5	837	829	0.9	2.5x	337	329	2.3	100%	of	7
am.17	8930	8854	0.8	1.1x	8431	8405	0.3	100%	of	36
am.18	8901	8825	0.8	1.1x	8426	8400	0.3	100%	of	36

B Compiler optimization level

Options controlling optimizations are numerous and may vary a lot depending on the targeted processor and the compiler version. Options listed here are for the compiler used for our experiment (`arm-elf-gcc (GCC) 4.4.2`), with no guarantee that they apply directly to other compilers. Ensuring the coherence of a set of optimizations is technically hard, this is why we start with a predefined level of optimization (`-O1`) and remove optimizations that may modify the control structure (using the corresponding `-fno` flag). To select the options, we just rely on the user manual: we remove any optimization that mention a possible influence on the control structure or that may affect the precision of the debugging information (i.e. the association instruction/source line).

Loop transformations may drastically change the structure of the program, and are thus forbidden.

```
-fno-loop-block \
-fno-loop-interchange \
-fno-loop-strip-mine \
-fno-move-loop-invariants \
-fno-reschedule-modulo-scheduled-loops \
-fno-unroll-loops \
-fno-unroll-all-loops \
-fno-unsafe-loop-optimizations \
```

Miscelaneous CFG transformations concern dead code elimination, inlining, branch removal, block reordering etc.

```
-fno-dce \
-fno-dse \
-fno-guess-branch-probability \
-fno-inline-small-functions \
-fno-crossjumping \
-fno-if-conversion \
-fno-if-conversion2 \
-fno-jump-tables \
-fno-reorder-blocks \
-fno-reorder-blocks-and-partition \
-fno-unswitch-loops \
```

SSA tree optimizations and misc. global optimizations have an indirect influence in the control structure, by removing, regrouping or re-ordering instructions. They also affect the precision of the debugging information (dwarf) which is the only information we have to relate the binary and the source code.

```
-fno-tree-builtin-call-dce \
-fno-tree-ccp \
-fno-tree-ch \
-fno-tree-copyrename \
-fno-tree-dce \
-fno-tree-dominator-opts \
-fno-tree-dse \
-fno-tree-fre \
```

```
-fno-tree-loop-distribution \
-fno-tree-loop-im \
-fno-tree-loop-ivcanon \
-fno-tree-loop-linear \
-fno-tree-loop-optimize \
-fno-tree-sra \
-fno-tree-ter \
-fno-auto-inc-dec \
-fno-cprop-registers \
-fno-defer-pop \
-fno-ipa-pure-const \
-fno-ipa-reference \
-fno-merge-constants\
-fno-split-wide-types \
-fno-unit-at-a-time \
```

A Survey of Probabilistic Timing Analysis Techniques for Real-Time Systems

Robert I. Davis
University of York, UK and Inria, France
rob.davis@york.ac.uk

Liliana Cucu-Grosjean
Inria, France
liliana.cucu@inria.fr

Abstract

This survey covers probabilistic timing analysis techniques for real-time systems. It reviews and critiques the key results in the field from its origins in 2000 to the latest research published up to the end of August 2018. The survey provides a taxonomy of the different methods used, and a classification of existing research. A detailed review is provided covering the main subject areas: static probabilistic timing analysis, measurement-based probabilistic timing analysis, and hybrid methods. In addition, research on supporting mechanisms and techniques, case studies, and evaluations is also reviewed. The survey concludes by identifying open issues, key challenges and possible directions for future research.

2012 ACM Subject Classification Software and its engineering → Software organization and properties, Software and its engineering → Software functional properties, Software and its engineering → Real-time schedulability, Computer systems organization → Real-time systems
Keywords and Phrases Probabilistic, real-time, timing analysis
Digital Object Identifier 10.4230/LITES-v006-i001-a003
Received 2018-01-04 **Accepted** 2019-02-26 **Published** 2019-05-14

1 Introduction

Systems are characterised as *real-time* if, as well as meeting functional requirements, they are required to meet timing requirements. Real-time systems may be further classified as *hard* real-time, where failure to meet their timing requirements constitutes a failure of the system; or *soft* real-time, where such failure leads only to a degraded quality of service. Today, both hard and soft real-time systems are found in many diverse application areas including; automotive, aerospace, medical systems, robotics, and consumer electronics.

Real-time systems are typically implemented via a set of programs, also referred to as tasks, which are executed on a recurring basis. The programs used in a real-time system have a *functional* behaviour, and also a *timing* behaviour. The functional behaviour of a given run of a program depends on the *input state*, which comprises a set of values for the input variables, and a set of values for the software state variables (which are related to the values of the input variables used on previous runs). The input state affects the path taken through the code, and the values of the outputs produced. Typically programs have a functional behaviour which is *deterministic*, in other words, given the exact same inputs they will produce the exact same outputs. Functional behaviour may also be *non-deterministic*, for example in a randomised search the same input state may lead to different outputs depending on the behaviour of a random number generator. In this survey we are mainly concerned with the timing behaviour of programs that have deterministic functionality.

In keeping with the majority of the work on program timing behaviour, in the following we consider programs that are run without interruption or preemption and without any interference from other programs that could be running on the same or different processor cores (i.e. no multi-threading and no cross-core interference). We return to this point in the conclusions.

The timing behaviour of a program with deterministic functionality depends on both the input state, and the initial values of hardware state variables, referred to as the *hardware state*. Examples of hardware state variables include the contents of internal buffers, pipelines, caches, scratchpads, and certain register values. While the hardware state may affect the timing behaviour of the program, it has no effect on the functional behaviour. A hardware platform is referred to as *time-predictable* if it always takes the same amount of time to execute a deterministic program when starting from the same input state and the same hardware state. By contrast, a *time-randomised* hardware platform may take a variable amount of time to execute such a program when starting from the same input state and hardware state, due to the behaviour of underlying random elements in the hardware[1].

1.1 Conventional Timing Analysis Techniques

Understanding the timing behaviour of each program is fundamental to verifying the timing requirements of a real-time system. Key to this is *timing analysis*, which seeks to characterise the amount of time that each program can take to execute on the given hardware platform. Typically, this is done by upper bounding or estimating the *Worst-Case Execution Time*.

▶ **Definition 1.** The *Worst-Case Execution Time (WCET)* of a program is an upper bound on the execution time of that program for any valid input state and initial hardware state (i.e. the WCET is an upper bound on the execution time for any single run of the program, and there is at least one run of the program that can realise the WCET).

The methods traditionally used for timing analysis can be classified into three main categories:

- *Static Analysis*: These methods do not execute the program on the actual hardware or on a simulator. Rather they analyse the code for the program and some annotations (providing information about input values), along with an abstract model of the hardware. Typically static analysis proceeds in three steps. First, control flow analysis is used to derive constraints on feasible paths, including loop bounds. Second, micro-architectural analysis is used to provide an over-approximation of the program execution on the feasible paths, accounting for the behaviour of hardware features such as pipelines and caches. Third, path analysis uses integer linear programming (ILP) to combine the results of control flow analysis and micro-architectural analysis, and so derive an upper bound on the WCET of the program.
  To derive an upper bound on the WCET static analysis has to determine properties relating to the dynamic behaviour of the program without actually executing it. In practice, it may not be possible to precisely determine all of these properties due to issues of tractability and decidability. For example determining the precise cache contents at a given program point may not be possible when there is a dependency on the input values. Properties which cannot be precisely determined must be conservatively approximated to ensure that the computed WCET remains a valid upper bound; however, such approximations may lead to significant pessimism. For advanced hardware platforms, there are two main challenges for static analysis methods. Firstly, obtaining and validating all of the information necessary to build an accurate

[1] Note here the basic random elements in the hardware (e.g. a random number generator) are not considered to be part of the hardware state.

model of the hardware components that impact program execution times. Secondly, modelling those components and their interactions without substantial loss of precision in the derived WCET upper bound.

- *Dynamic or Measurement-Based Analysis*: These methods derive an estimate of the WCET by running the program on the actual hardware or on a cycle-accurate timing simulator. A measurement protocol provides test vectors (sets of input values) and initial hardware configurations that are used to exercise a subset of the possible paths through the code, as well as the possible hardware states that may affect the timing behaviour[2]. The execution times for multiple runs of the program are collected and the maximum observed execution time recorded. This value may be used as a (lower bound) estimate of the WCET, or alternatively, an engineering margin (e.g. 20%) may be added to give an estimate of the WCET. This margin comes from industrial practice and engineering judgement [129]; however, there is no guarantee that it results in an upper bound on the actual WCET. For complex programs and advanced hardware platforms, the two main challenges for measurement-based analysis methods both involve designing an appropriate measurement protocol. Firstly, if it were known which values for the input variables and software state variables would lead to the WCET, then the measurement protocol could ensure that those values were present in the test vectors used; however, typically these values are not known and cannot easily be derived. Secondly, it may not be known, or easy to derive, which initial hardware states will lead to the WCET; it may also be difficult to force the hardware into a particular initial state. Nevertheless, measurement-based analysis is commonly used in industry and may give engineers a useful perspective on the timing behaviour of a program.
- *Hybrid Analysis*: These methods combine elements of both static analysis and measurement-based analysis. For example, a hybrid approach may record the maximum observed execution time for short sub-paths through the code, and then combine these values using information obtained via static analysis of the program's structure (e.g. the control flow graph) to estimate the WCET. The aim of hybrid analysis is to overcome the disadvantages of both static and dynamic methods. By measuring execution times for short sub-paths through the code, hybrid methods avoid the need for a model of the hardware, which may be difficult to obtain and to validate. By using static analysis techniques to determine the control flow graph, the problem of having to find input values that exercise the worst-case path is ameliorated. Instead, the measurement protocol can focus on ensuring that measurements are obtained for all sub-paths, i.e. structural coverage, which is far simpler to achieve than full path coverage. Nevertheless, hybrid methods still inherit many of the challenges of measurement-based methods. On advanced hardware (e.g. with pipelines and caches) the execution time of a sub-path may be dependent on the execution history, and hence on the previous sub-paths that were executed. This exacerbates the problem of composing the overall WCET estimate from observations for individual sub-paths, and may degrade the precision of that estimate. In general, today's hybrid methods cannot guarantee to upper bound the WCET; however, the estimates they produce may be more accurate than those based on measurements alone.

During the past two decades, the hardware platforms used, or proposed for use, in real-time systems have become increasingly more complex. Architectures include advanced hardware acceleration features such as pipelines, branch prediction, out-of-order execution, caches, write-buffers, scratchpads, and multiple levels of memory hierarchy. These advances, along with increasing software complexity, greatly exacerbate the timing analysis problem. Most acceleration

[2] Exercising all possible paths and initial hardware states is often impractical.

features are designed to optimise average-case rather than worst-case behaviour and can result in significant variability in execution times. This is making it increasingly difficult, if not impossible, to obtain tight WCET estimates[3] from conventional static timing analysis methods that seek to provide an upper bound on the WCET. Further, increases in software and hardware complexity make it difficult to design measurement protocols capable of ensuring that the worst-case path(s) through the code are exercised, and that the worst-case hardware states are encountered when using measurement-based and hybrid analyses. (Appendix A provides further discussion of measurement protocols).

1.2 Probabilistic Timing Analysis Techniques

Probabilistic timing analysis[4] differs from traditional approaches in that it lifts the characterisation of the timing behaviour of a program from the consideration of a single run, to the consideration of a repeating sequence of many runs, referred to as a *scenario*, and hence lifts the results from a scalar value (the WCET) to a probability distribution (the pWCET distribution, defined in Section 2.1). Traditional timing analysis methods aim to tightly upper bound the execution time that could occur for a single run of a program out of all possible runs. Similarly, probabilistic timing analysis methods aim to tightly upper bound the distribution of execution times that could potentially occur for some scenario of operation, out of all possible scenarios of operation.

Research into probabilistic timing analysis can be classified into five main categories. This classification forms the basis for the main sections of this survey. Note, for ease of reference we have numbered these categories below, starting at 3, to match the section of survey.

3. *Static Probabilistic Timing Analysis (SPTA)*: Similar to traditional static analyses, SPTA methods do not execute the program on the actual hardware or on a simulator. Instead they analyse the code for the program and information about input values, along with an abstract model of the hardware behaviour. The difference is that SPTA methods account for some form of random behaviour in either the hardware, the software, or the environment (i.e. the inputs) by using probability distributions, and therefore construct an upper bound on the pWCET distribution rather than a upper bound on the WCET. In common with traditional static analyses, SPTA methods have to determine properties relating to the dynamic behaviour of the program without actually executing it. Here, the conservative approximation of properties which cannot be precisely determined (e.g. cache states in a random replacement cache) may lead to significant pessimism in the estimated pWCET distribution. The two main challenges for SPTA methods are obtaining and validating the information necessary to build an accurate model of the hardware components; and modelling those components and their interactions without substantial loss of precision in the upper bound pWCET distribution derived.
4. *Measurement-Based Probabilistic Timing Analysis (MBPTA)*: Today, most of the current MBPTA methods use Extreme Value Theory (EVT) to make a *statistical estimate* of the pWCET distribution of a program. This estimate is based on a sample of execution time observations obtained by executing the program on the hardware or a cycle accurate simulator according to a *measurement protocol*. The measurement protocol samples some scenario(s) of operation, i.e. it executes the program multiple times according to a set of feasible input states and initial hardware states. As with traditional measurement-based analysis methods, the

[3] By a tight WCET estimate we mean one that is relatively close to the actual WCET, for example perhaps no more than 10-20% larger.

[4] In this survey, we adopt the widely used term "probabilistic timing analysis" noting that it can easily be misinterpreted. To clarify, while the results produced are expressed in terms of probability distributions, the analysis methods themselves are deterministic in the sense of always producing the same results from the same inputs, unlike for example randomised search techniques.

main challenge for MBPTA methods involves designing an appropriate measurement protocol. In particular, in order for the estimated pWCET distribution derived by EVT to be valid for a future scenario of operation, then the sample of input states and hardware states used for analysis must be *representative* of those that will occur during that future scenario of operation. An important issue here is that there may not be a single sample of input states and hardware states that is representative of all possible future scenarios of operation.

5. *Hybrid Probabilistic Timing Analysis (HyPTA)*: These methods combine in some way both statistical and analytical approaches. For example by taking measurements at the level of basic blocks or sub-paths, and then composing the results (i.e. the estimated pWCET distributions for the sub-paths) using structural information obtained from static analysis of the code.
6. *Enabling mechanisms*: These mechanisms aim to facilitate the use of one or other of the above analysis methods.
7. *Evaluation*: Case studies, benchmarks, and metrics, which aim to evaluate the efficiency, effectiveness, and applicability of probabilistic timing analysis methods.

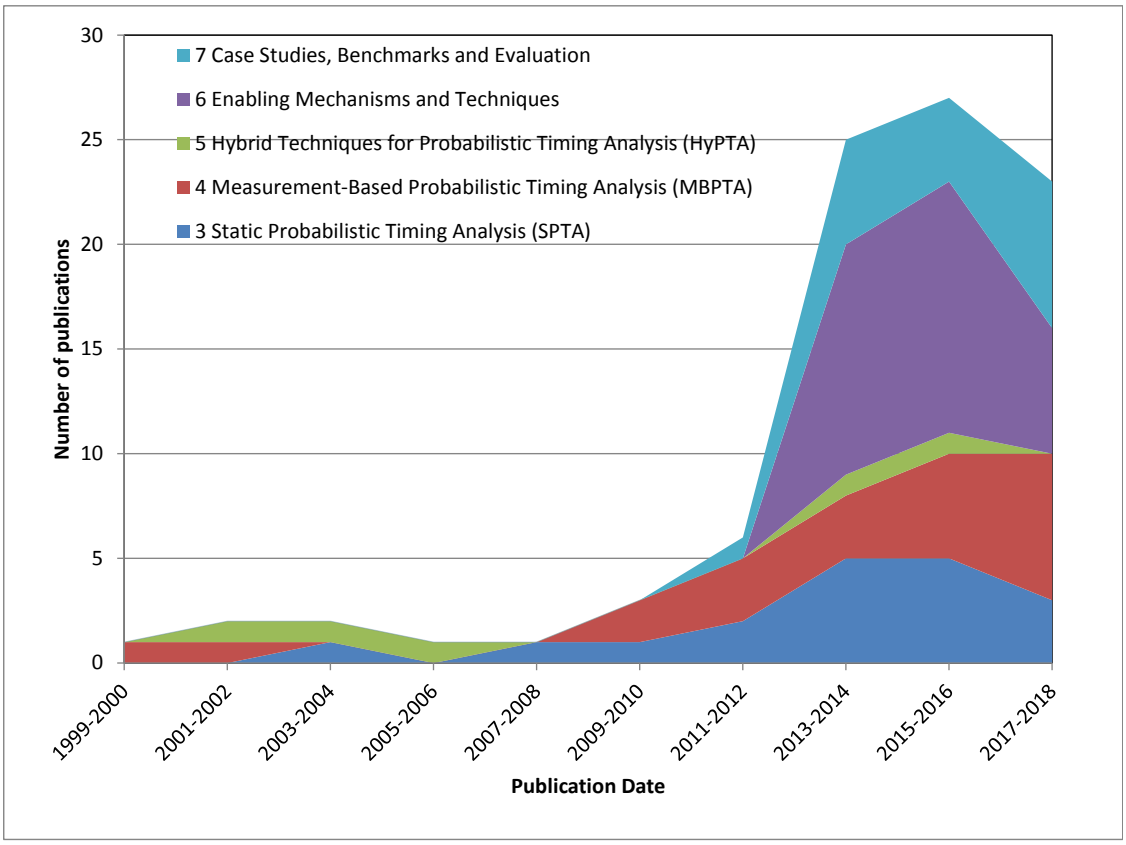


Figure 1 Intensity of research in the different categories corresponding to Sections 3 to 7 of this survey.

The research in these categories is summarised by authors and citations in Table 1. Note the sub-categories correspond to the subsections of this survey.

It is interesting to note how research in the different categories has progressed over time. Figure 1 illustrates the number of papers reviewed in each of the main categories covered by this survey that have been published during 2-year time intervals from 1999 to 2018. (This figure is best viewed online in colour). A number of observations can be drawn from Figure 1. Firstly, the volume of research into probabilistic timing analysis was relatively flat until around 2008

Table 1 Summary of publications from different authors in the categories described in the main sections and subsections of this survey.

3 Static Probabilistic Timing Analysis (SPTA)
3.1 SPTA based on Probabilities from Inputs David and Puaut [35], Liang and Mitra [86]
3.2 SPTA based on Probabilities from Faults Hofig [62], Hardy and Puaut [58, 59], Chen et al. [31, 29]
3.3 SPTA based on Probabilities from Random Replacement Caches Quinones et al. [106], Burns and Griffin [23], Cazorla et al. [26], Davis et al. [39], Altmeyer and Davis [7], Altmeyer et al. [6], Griffin et al. [52], Lesage et al. [83, 82], Chen and Beltrame [28], Chen et al. [30]
4 Measurement-Based Probabilistic Timing Analysis (MBPTA)
4.1 EVT and i.i.d. observations Burns and Edgar [22], Edgar and Burns [45], Hansen et al. [57], Griffin and Burns [51], Lu et al. [89, 90], Cucu-Grosjean et al. [34]
4.2 EVT and observations with dependences Melani et al. [93], Santinelli et al. [112], Berezovskyi et al. [16, 15], Guet et al. [53, 54], Fedotova et al. [47], Lima and Bate [87]
4.3 EVT and representativity Lima et al. [88], Maxim et al. [92], Santinelli et al. [110], Santinelli and Guo [111], Guet et al. [55], Abella et al. [3], Milutinovic et al. [101]
5 Hybrid Techniques for Probabilistic Timing Analysis (HyPTA)
5.1 HyPTA and the Path Coverage Problem Bernat et al. [19, 18, 17], Kosmidis et al. [70], Ziccardi et al. [136]
6 Enabling Mechanisms and Techniques
6.1 Caches and Hardware Random Placement Kosmidis et al. [68, 69, 67], Slijepcevic et al. [120], Anwar et al. [9], Hernandez et al. [61], Trillia et al. [127], Benedicte et al. [11]
6.2 Caches and Software Random Placement Kosmidis et al. [72, 73, 71, 78]
6.3 Cache Risk Patterns with Random Placement Abella et al. [4], Benedicte et al. [13, 14], Milutinovic et al. [98, 99, 97, 100]
6.4 Buffers, Buses and other Resources Slijepcevic et al. [119], Cazorla et al. [27], Kosmidis et al. [74], Jalle et al. [65], Panic et al. [102], Agirre et al. [5], Hernandez et al. [60], Slijepcevic et al. [117, 118], Benedicte et al. [12]
7 Case Studies, Benchmarks and Evaluation
7.1 Critiques Reineke [108], Mezzetti et al. [96], Stephenson et al. [123], Gil et al. [49]
7.2 Case Studies and Evaluation Santos et al. [113], Kosmidis et al. [75, 77], Abella et al. [2], Wartel et al. [129, 128], Lesage et al. [85], Mezzetti et al. [94], Fernandez et al. [48], Cros et al. [33] Diaz et al. [43], Silva et al. [116], Reghenzani et al. [107]

and then increased rapidly in the decade from 2010. Inline with this the number of publications on the main theme of measurement-based probabilistic timing analysis (Section 4) has steadily increased since 2009/2010. Work on static probabilistic timing analysis (Section 3) peaked during the period 2013–2016, coinciding with research effort on the EU Proxima project[5]. Also, as a result of that project, there was a significant peak in research on enabling techniques aimed at facilitating the use of MBPTA (Section 6). More recently, in 2017/2018, the focus has been on extending MBPTA to more complex systems and exploring the effectiveness of the approach via case studies and benchmarks (Section 7).

Before moving to the sections of this survey which review the literature, we first discuss (in Section 2) fundamental concepts and methods relating to probabilistic timing analysis.

Note that conventional timing analysis techniques aimed at providing an upper bound on the WCET value as the solution to the timing analysis problem are outside of the scope of this survey, they are reviewed in detail by Wilhelm et al. [133].

2 Fundamental Concepts and Methods

The term *probabilistic real-time systems* is used to refer to real-time systems where one or more of the parameters, such as program execution times, are modelled by random variables. Although a parameter is described (i.e. *modelled*) by a random variable, this does not necessarily mean that the actual parameter itself exhibits random behaviour or that there is necessarily any underlying random element to the system that determines its behaviour. The actual behaviour of the parameter may depend on complex and unknown or uncertain behaviours of the overall system. As an example, the outcome of a coin toss can be modelled as a random variable with heads and tails each having a probability of 0.5 of occurring, assuming that the coin is fair. However, the actually process of tossing a coin does not actually have a random element to it. The outcome could in theory be predicted to a high degree of accuracy if there were sufficiently precise information available about the initial state and the complex behaviour and evolution of the overall physical processes involved. There are however many useful results that can be obtained by modelling the outcome of a coin toss as a random variable. The same is true of the analysis of probabilistic real-time systems.

In this section, we first discuss a fundamental and often misunderstood concept in probabilistic timing analysis, *probabilistic Worst-Case Execution Time (pWCET)* distributions. The remainder of the section then provides an outline of the two main approaches to obtaining pWCET distributions, *Static Probabilistic Timing Analysis (SPTA)* and *Measurement-Based Probabilistic Timing Analysis (MBPTA)*.

2.1 probabilistic Worst-Case Execution Time (pWCET)

The term *probabilistic Worst-Case Execution Time (pWCET)* distribution has been used widely in the literature, with a number of different definitions given. Below, we provide an overarching definition of the term. (We use calligraphic characters, such as $\mathcal{X}$, to denote random variables).

▶ **Definition 2.** The *probabilistic Worst-Case Execution Time (pWCET)* distribution for a program is the least upper bound, in the sense of the greater than or equal to operator $\succeq$ (defined below), on the execution time distribution of the program for every valid scenario of operation, where a *scenario* of operation is defined as an infinitely repeating sequence of input states and initial hardware states that characterise a feasible way in which recurrent execution of the program may occur.

[5] http://www.proxima-project.eu/

▶ **Definition 3.** (From Diaz et al. [44]) The probability distribution of a random variable $\mathcal{X}$ is *greater than or equal to* (i.e. upper bounds) that of another random variable $\mathcal{Y}$ (denoted by $\mathcal{X} \succeq \mathcal{Y}$) if the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is never above that of $\mathcal{Y}$, or alternatively, the 1-CDF of $\mathcal{X}$ is never below that of $\mathcal{Y}$.

Graphically, Definition 2 means that the 1 - CDF of the pWCET distribution is never below that of the execution time distribution for any scenario of operation. Hence the 1 - CDF or *exceedance funtion* of the pWCET distribution may be used to determine an upper bound on the probability p that the execution time of a randomly selected run of the program will exceed an execution time budget x, for any chosen value of x. This upper bound is valid for any feasible scenario of operation.

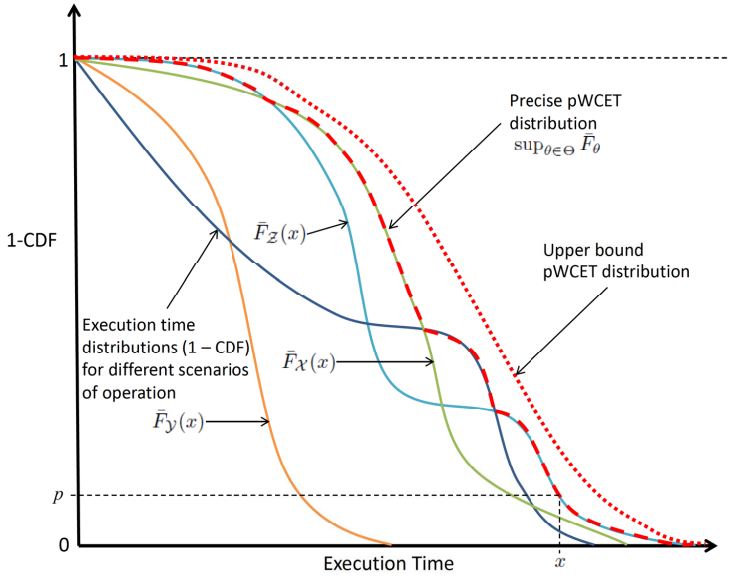


Figure 2 Exceedance function or 1-CDF for the pWCET distribution of a program, and also execution time distributions for specific scenarios of operation.

Figure 2 illustrates the execution time distributions of a number of different scenarios of operation (solid lines), the precise pWCET distribution (red dashed line) which is the least upper bound (i.e. the point-wise maxima of the 1 - CDF) for all of these distributions, and also some arbitrary upper bound pWCET distribution (red dotted line) which is a pessimistic estimate of the precise pWCET. Also shown (on the y-axis) is an upper bound p on the probability that any randomly selected run of the program will have an execution time that exceeds x (on the x-axis). The value x is referred to as the pWCET estimate at a probability of exceedance of p. (More formally, the least upper bound pWCET distribution is given by $\sup_{\theta\in\Theta} \bar{F}_\theta$, where $\bar{F}_\theta$ is the 1 - CDF for scenario of operation θ, and Θ is the space of all valid scenarios of operation).

Note that the greater than or equal to relation $\succeq$ between two random variables does not provide a total order, i.e. for two random variables $\mathcal{X}$ and $\mathcal{Z}$ it is possible that $\mathcal{X} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq X$. Hence the precise pWCET distribution may not correspond to the execution time distribution for any specific scenario. This can be seen in Figure 2, considering the execution time distributions $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$. It is the case that $\mathcal{X} \succeq \mathcal{Y}$, but $\mathcal{X} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq X$. By contrast, as the greater than or equal to relation for scalars ($\geq$) does provide a total order, the precise WCET does correspond to the execution time for some specific run of the program.

The WCET upper bounds all possible execution times for a program, independent of any particular run of the program. Similarly, the pWCET distribution upper bounds all possible execution time distributions for a program, independent of any particular scenario of operation.

We note that the term pWCET is open to misinterpretation and is often misunderstood. To clarify, it does *not* refer to the probability distribution of the worst-case execution time, since the WCET is a single value. Rather informally, following Definition 2, the pWCET may be thought of as the "worst-case" (in the sense of upper bound) probability distribution of the execution time for any scenario of operation.

Below, we provide some simple hypothetical examples that illustrate the meaning of the pWCET distribution.

Consider a program A running on time-randomised hardware. Further, assume that the program has two paths which may be selected based on the value of an input variable. The discrete probability distributions $\mathcal{X}$ and $\mathcal{Y}$ of the execution time of each path may be described by *probability mass functions* as follows:

$$\mathcal{X} = \begin{pmatrix} 10 & 20 & 30 & 40 \\ 0.4 & 0.3 & 0.2 & 0.1 \end{pmatrix} \mathcal{Y} = \begin{pmatrix} 20 & 30 & 40 & 50 \\ 0.8 & 0.15 & 0.04 & 0.01 \end{pmatrix}$$

Indicating, among other things, that there is a probability of 0.1 that the execution time of the first path is 40 and a probability of 0.01 that the execution time of the second path is 50.

The *Complementary Cumulative Distribution Function* (1-CDF) or *Exceedance Function* defined as $\bar{F}_{\mathcal{X}}(x) = P(\mathcal{X} > x)$ is given by:

$$\bar{F}_{\mathcal{X}} = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 \\ 1 & 0.6 & 0.3 & 0.1 & 0 \end{pmatrix} \bar{F}_{\mathcal{Y}} = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 1 & 1 & 0.2 & 0.05 & 0.01 & 0 \end{pmatrix}$$

The precise pWCET distribution for the two paths can be found by taking the point-wise maxima of their exceedance functions. Hence:

$$pWCET = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 & 50 \\ 1 & 1 & 0.3 & 0.1 & 0.01 & 0 \end{pmatrix}$$

Note that the above pWCET distribution is precise on the assumption that repeatedly executing one path or the other or any combination of them is a valid scenario of operation, otherwise it may be that it is an upper bound rather than the precise pWCET.

Observe that the program has a WCET of 50, which equates to the last value in the pWCET distribution (where the 1-CDF becomes zero). The pWCET distribution gives more nuanced information than this single value, as it upper bounds the probability of occurrence of the extreme execution time values (e.g. 30, 40, and 50) that occur rarely and form the tail of the distribution.

Execution on time-randomised hardware is not the only way in which a non-degenerate[6] pWCET distribution can arise. As an alternative, consider another program B that implements a software state machine with four states and hence four paths that runs on time-predictable hardware. Here the main factor which affects the execution time is the path taken, which is determined by the value of the software state variable. For this program, all valid scenarios of operation involve the software state variable cycling through its four possible values in order, and hence the four possible paths executing in order on any four consecutive runs of the program. Further, assume that there is a small variability in the execution time of each path depending on the value of some input variable, which may take any value on any run. Hence the execution

[6] A degenerate distribution has only a single value.

times of the different paths are 10 ± 2, 20 ± 2, 30 ± 2, and 40 ± 2, each with a probability of 0.25. For this program, the pWCET distribution valid for any scenario of operation is:

$$pWCET = \begin{pmatrix} 0 & 12 & 22 & 32 & 42 \\ 1 & 0.75 & 0.5 & 0.25 & 0 \end{pmatrix}$$

Finally, consider a program C which again runs on time-predictable hardware. This program has an input variable v which may take one of four values selecting one of four different paths. The execution times of the paths are 10, 20, 30, and 40. Further, we are given some additional information about all the valid scenarios of operation, over a large number of runs of the program, the first 3 input values occur with the same probability, while the 4th is a fault condition that occurs at most 1% of the time. The pWCET distribution is as follows:

$$pWCET = \begin{pmatrix} 0 & 10 & 20 & 30 & 40 \\ 1 & 0.67 & 0.34 & 0.01 & 0 \end{pmatrix}$$

We note that in all three examples, the pWCET distribution upper bounds the execution time distribution for a randomly selected run of the program in any scenario of operation. However, it is not always the case that the pWCET distribution is *probabilistically independent* of the value realised for the execution time of previous runs of the program. For example in the case of program B, the pWCET distribution does not provide a valid upper bound on the execution time distribution of the program *conditional* on specific execution times having occurred for previous runs. (If the previous execution time of the program was 30, then the execution time of the current run has a probability of 1 of exceeding 32, since the state variable will increment by 1 and the longest path will be selected with an execution time of 40 ± 2). Further, in the case of program C, although the fault condition may occur only 1% of the time, there may well be a cluster of faults. Hence for programs B and C, it is not valid to compose the pWCET distributions using basic convolution to obtain a bound on the interference (total execution time) of two or more runs of the program. This has implications for how the pWCET distribution may be used in probabilistic schedulability analysis, which are discussed in the companion survey on that topic [38].

▶ **Definition 4.** Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are *probabilistically independent* if they describe two events such that knowledge of whether one event did or did not occur does not change the probability that the other event occurs. Stated otherwise, the joint probability is equal to the product of their probabilities $P(\{\mathcal{X} = x\} \cap \{\mathcal{Y} = y\}) = P(\mathcal{X} = x) \cdot P(\mathcal{Y} = y)$. (In this context, the events are the execution times of runs of the program taking certain values).

We note that while the above simple examples are useful to illustrate the concept of a pWCET distribution, in practice the exceedance probabilities of interest are very small, typically in the range 10^{-4} to 10^{-15}. These probabilities derive from the acceptable failure rate per hour of operation for the application considered. (Note, the relationship between failure rates per hour of operation and probabilities of timing failure depend on various factors considered in fault tree analysis, including any mitigations and recovery mechanisms that may be applied in the event of a timing failure [50], see the companion survey [38] for further discussion).

It is interesting to consider the use and interpretation of the pWCET distribution for a program. Let us assume that the program will be run repeatedly a potentially unbounded number of times, and that a fixed execution time budget of x applies to each run. Further, we assume that this budget is enforced by the operating system, and therefore that any run of the program which has not completed within an execution time of x is terminated and assumed to have failed. The pWCET distribution provides the following information (by reading off the probability of exceedance p associated with the execution time budget x, see Figure 2):

(i) An upper bound p on the probability (with a long-run frequency interpretation) equating to the number of runs expected to exceed the execution time budget x divided by the total number of runs in a long (tending to infinite) time interval.

(ii) An upper bound p on the probability that the execution time budget x will be exceeded on a randomly selected run. (This is broadly equivalent to the above long-run frequency interpretation).

Contrast this with the binary information provided by the WCET. If x is greater than or equal to the WCET, then we can expect the budget to never be exceeded. However, if x is less than the WCET, then we expect the budget to be exceeded on some runs, but we have no information on how frequently this may occur. Hard real-time systems in many application domains can in practice tolerate a small number of consecutive failures of a program to meet its execution time budget, but cannot tolerate long black-out periods when every run fails to complete within its budget. The problem of reconciling requirements on the length of potential black-out periods and a probabilistic treatment of execution times has, as far as we are aware, received little attention in the literature. We note that calculation of the probability of such black-out periods occurring requires information about the dependences (or independence) of the pWCET distributions for consecutive runs of the program. This topic is discussed further in the companion survey on probabilistic schedulability analysis [38].

We note that some researchers have interpreted the pWCET distribution as giving the probability or confidence $(1-p)$ that the WCET does not exceed some value x. This interpretation can be confusing, since the meaning of the WCET is normally taken to be the largest possible execution time that could be realised on any single run of the program, as in Definition 1. Instead, in line with Definition 2 and (i) above, we view the 1 - CDF of the pWCET distribution as providing, for any chosen value x for the execution time budget, an associated upper bound probability p (with a long-run frequency interpretation) equating to the number of runs of the program expected to exceed the execution time budget x divided by the total number of runs of the program in a long time interval.

2.2 Overview of Static Probabilistic Timing Analysis (SPTA)

The aim of Static Probabilistic Timing Analysis (SPTA) methods is to *construct* an upper bound on the pWCET distribution of a program by applying static analysis techniques to the code of the program (supplemented by information about input values) along with an abstract model of the hardware behaviour. Typically, a precise analysis is not possible due to issues of tractability, and thus over-approximations are made which may lead to pessimism in the upper bound pWCET distribution computed. For static analysis to produce a non-degenerate pWCET distribution there has to be some part of the system or its environment that contributes random or probabilistic timing behaviour.

SPTA methods for programs running on time-randomised hardware (e.g. with a random replacement cache) effectively consider each path through the code. For each path, these methods construct a pWCET distribution that upper bounds the probability distribution of the execution time for that path, considering all possible initial hardware states and all possible input states that cause execution of the path. For simple hardware, it is sufficient to consider an empty cache (the worst-case hardware state) and a single input state that drives the path, since there is no difference in execution times for different inputs that select the same path. Nevertheless, significant approximations need to be made in the analysis, since the problem of determining the possible cache states and their probabilities at each program point is intractable. The upper bound pWCET distributions for every path are then combined using an envelope function (taking the point-wise maxima over the 1-CDFs) to determine an upper bound on the pWCET distribution

for the program that is valid *independent* of the path taken. (More sophisticated SPTA methods analyse sub-paths and use appropriate join operations at path convergence to compute tighter upper bounds on the pWCET distribution of the program). While the results provide a valid upper bound on the pWCET distribution, they are not necessarily tight even for simple architectures.

Note that SPTA methods typically do not explicitly consider different valid scenarios of operation, but rather they effectively assume that any scenario (sequence of input states and hardware states) could occur, and thus over-approximate.

2.3 Overview of Measurement-Based Probabilistic Timing Analysis (MBPTA)

The aim of Measurement-Based Probabilistic Timing Analysis methods is to make a *statistical estimate* of the pWCET distribution of a program. This estimate is derived from a sample of execution time observations obtained by executing the program on the hardware or on a cycle accurate simulator according to an appropriate *measurement protocol*. The measurement protocol executes the program multiple times according to some sequence(s) of feasible input states and initial hardware states, thus sampling one or more possible scenarios of operation.

Provided that the sample of execution time observations passes appropriate statistical tests (see below), then *Extreme Value Theory* (EVT) can be used to derive a statistical estimate of the probability distribution of the *extreme values*[7] of the execution time distribution for the program, i.e. to estimate the pWCET distribution. By modelling the shape of the distribution of the extreme execution times EVT is able to predict the probability of occurrence of execution time values that exceed any that have been observed.

Early results in Extreme Value Theory required the sample of observations used to be *independent and identically distributed (i.i.d.)*; however, later work by Leadbetter [81] showed that EVT can also be applied in the more general case of a series of observations which are *stationary*, but are not necessarily independent. Both i.i.d. and stationary properties can be checked using appropriate statistical tests.

▶ **Definition 5.** A sequence of random variables (i.e. a series of observations) are *independent and identically distributed (i.i.d.)* if they are mutually independent (see Definition 4) and each random variable has the same probability distribution as the others.

▶ **Definition 6.** A sequence of random variables (i.e. a series of observations) is *stationary* if the joint probability distribution does not change when shifted in time, and hence the mean and variance do not change over time.

We note that simple software state machines that produce a cyclically repeating behaviour typically result in a stationary series of execution time observations.

In order for the estimated pWCET distribution derived by EVT to be valid for a future scenario of operation, then the sample of input states and initial hardware states used for analysis must be *representative* of those that will occur during that scenario.

▶ **Definition 7.** A sample of input states and initial hardware states used for analysis is *representative* of the population of states that may occur during a future scenario of operation if the property of interest (i.e. the pWCET distribution) derived from the sample of states used for analysis matches or upper bounds the property that would be obtained from the population of states that occur during the scenario of operation.

[7] By extreme values we mean large values towards the tail of the distribution, which have a small probability of occurrence.

▶ **Definition 8.** As determined by MBPTA, the estimated pWCET distribution for an entire program (or path through a program) is a statistical estimate of the probability distribution of the *extreme values* of the execution time of that program (or path), valid for any future scenarios of operation that are properly represented by the sample of input states and initial hardware states used in the analysis.

Ideally, MBPTA would provide a pWCET distribution which is valid for any of the many possible scenarios of operation; however, an important issue here is that there may not be one single distribution of input states and hardware states that is representative of all possible future scenarios of operation. This issue of *representativity* is a key open problem in research on the practical use of MBPTA.

The difficulty in ensuring representativity can be ameliorated by taking steps to make regions of the input state space equivalent in terms of execution time behaviour, similarly regions of the hardware state space. As a simple example, a floating point operation may have a variable latency that depends on the values of it operands; however, if a hardware test mode is used whereby the floating point operation always takes the same worst-case latency regardless of these values, then any arbitrary values can be chosen, and they will be representative (as far as execution times are concerned) of any other possible values in the input state space. Similarly, a fully associative random replacement cache can make many, but not necessarily all, patterns of accesses to data at different memory locations have equivalent execution time behaviour. Unfortunately, these steps typically require modifications to the hardware used. The problem of representativity is most acute for COTS (Common Off The Shelf) hardware. In particular, it is challenging to ensure representativity with hardware that has complex deterministic behaviour based on substantial state and history dependency. Examples include LRU/PLRU caches, and caches with a write-back behaviour. Here, the latency of instructions that access memory can vary significantly based on the specific history of prior execution, i.e. the specific sequence of addresses accessed. Further, it is hard to determine which regions of the input state space are equivalent in terms of execution time behaviour. This makes it very difficult to ensure that the input data used is representative .

Two EVT theorems and associated methods have been employed in the literature on MBPTA: the *Block Maxima* method based on the Fisher-Tippett-Gnedenko theorem, and the *Peaks-over-Threshold* (PoT) method based on Pickands-Balkema-de Haan theorem (see the books by Coles [32] and Embrechts et al. [46] for details of these theorems).

The Block Maxima method can be summarised as follows:

- Obtain a sample of execution time observations using an appropriate measurement protocol.
- Check, via appropriate statistical tests, that the execution time observations collected are analysable using EVT.
- Divide the sample into blocks of a fixed size, and take the maximum value for each block. (Note, in practice only the maxima of the blocks need be independent for the theory to apply, not necessarily all of the sample data).
- Fit a *Generalised Extreme Value* (GEV) distribution to the maxima. This will be a reversed Weibull, Gumbel, or Frechet distribution, depending on the shape parameter. (Alternatively fit a GEV with the shape parameter fixed to zero i.e. a Gumbel distribution).
- Check the goodness of fit between the maxima and the fitted GEV (e.g. using quantile plots).
- The GEV distribution obtained for the extreme values estimates the pWCET distribution.

The Peaks-over-Threshold method can be similarly summarised as follows:

- Obtain execution time observations using an appropriate measurement protocol.
- Check, via appropriate statistical tests, that the execution time observations collected are analysable using EVT.

- Choose a suitable threshold, and select the values that exceed the threshold. Note that *de-clustering*[8] may be required for data that is not independent.
- Fit a *Generalised Pareto Distribution* (GPD) to the excesses over the threshold.
- Check the goodness of fit between the peak values selected and the fitted GPD (e.g. using quantile plots).
- The GPD distribution obtained for the extreme values estimates the pWCET distribution.

There are two main ways of applying MBPTA, referred to as *per-path* and *per-program*:

1. *Per-path:* MBPTA is applied at the level of paths. A measurement protocol is used to exercise all feasible paths, then the execution time observations are divided into separate samples according to the path that was executed. EVT is then used to estimate the pWCET distribution for each path. The pWCET distribution for the program as a whole is then estimated by taking an upper bound (an envelope or point wise maxima on the 1 - CDFs) over the set of pWCET estimates for all paths.
2. *Per-program:* MBPTA is applied at the level of the program. A measurement protocol is again used to exercise all paths. In this case, all of the execution time observations are grouped together into a single sample. EVT is then applied to that sample, thus estimating directly the pWCET distribution for the program.

There are a number of advantages and disadvantages to the per-path and per-program approaches for MBPTA. Recall that the execution time observations analysed using EVT must be identically distributed, which can be checked using appropriate statistical tests. In the per-program case, these tests can fail if the execution times from different paths come from quite different distributions. The independent distribution (i.d.) tests could also fail in the case of observations for an individual path; however, for this to happen there must be significant differences in the execution time distributions obtained for the same path with different input states. This is less likely in practice, although it may still happen.

With the per-path approach, issues of representativity arise only at the level of paths, whereas the per-program approach raises issues of representativity at the program level. With the per-path approach, it is sufficient that the sample of input states and initial hardware states used to generate execution time observations for a given path are representative of the input states and hardware states for that path that could occur during operation. For example, for relatively simple time-randomised hardware it may be possible to obtain a single representative input state for each path by resetting the hardware on each run to obtain worst-case initial conditions (e.g. an empty cache). For more complex hardware, it becomes more difficult to ensure that the input states and initial hardware states used in the analysis of an individual path are representative; nevertheless, the problem is somewhat simpler than in the per-program case.

With the per-program approach, the frequency at which different paths are exercised in the observations used for analysis can impact the pWCET distribution obtained. For example if a path with large execution times is exercised much more frequently during operation than it was during analysis, then the pWCET distribution obtained during analysis may not be valid for that behaviour. Since the sample of input states used during analysis was not representative, the pWCET distribution obtained may be optimistic.

Despite the above disadvantages, the per-program approach may be preferable in practice, since it does not require the user to separate out the execution time observations on a per-path basis.

[8] De-clustering typically involves using only the single maximum value in any group of continuous observations that exceeds the threshold.

Further, the per-path approach loses information about the ordering and dependences between execution time observations, which may be relevant to obtaining a tight pWCET distribution.

In seeking to employ EVT, there are two questions to consider: First, are the samples of observations obtained for input into EVT *representative* of the future scenarios of operation that can occur once the system is deployed? If the answer to this question is "no", then any results produced (e.g. estimated pWCET distributions) cannot be trusted to describe the behaviour of the deployed system. Secondly, can the particular EVT method chosen be applied to the observations? This question can be answered by applying appropriate statistical tests (e.g. checking for i.i.d., goodness-of-fit etc.). It is important to note that an answer "yes" to this second question does not necessarily imply that the results produced provide a sound[9] description of the behaviour of the deployed system. They may only provide a sound description of its behaviour in precisely those scenarios used for analysis, i.e. used to produce the observations. *Representativity* is essential to extend the results obtained via EVT to the actual operation of the system.

3 Static Probabilistic Timing Analysis (SPTA)

In this section, we review analytical approaches to determining pWCETs distributions, commonly referred to as Static Probabilistic Timing Analysis (SPTA). These methods *construct* an upper bound on the pWCET distribution of a program by applying static analysis techniques to the code of the program along with an abstract model of the hardware behaviour. In order for static analysis to produce a (non-degenerate) pWCET distribution there has to be some part of the system or its environment that contributes variability in terms of random or probabilistic timing behaviour. Examples include: (i) probabilities of certain inputs occurring, (ii) probabilities of faults occurring, (iii) time-randomised hardware behaviour, such as the use of random replacement caches. In the following subsections, we review the research on SPTA relating to each of these factors.

3.1 SPTA based on Probabilities from Inputs

Initial work on SPTA considered the probability of different input values occurring, or different conditional branches being taken in the code.

The first work in this area by David and Puaut [35] in 2004 assumed that the input variables are independent and have a known probability distribution in terms of the values that they can take. In this work, a tree-based static analysis is used to compute the execution time of each path and the probability that it will be taken. This generates a probability distribution for the execution time of the program, not considering hardware effects. The method has exponential complexity, which depends strongly on the external variables. The main drawback is the requirement for the input variables to be independent, which is unlikely to be the case in practice. Further the probability distribution for each input variable must be known.

In 2008, Liang and Mitra [86] analysed the effects of cache on the probability distribution of the execution times of a program, assuming that probabilities of conditional branches and statistical information about loop bounds are provided. They introduce the concept of *probabilistic cache states* to capture the probability distribution of different cache contents at each program point, and an appropriate join function for combining these states. The analysis computes the cache miss probability at each program point enabling the effects of the cache to be included in

[9] In this survey, we use the adjective *sound* to indicate a description, an analysis, or a probability distribution that provides information about the system that is not optimistic with respect to its timing behaviour. Thus the information provided may be precise, or it may be pessimistic.

an estimation of the execution time distribution of the program. Aside from issues of tractability, the main difficulty with this approach is the assumption that values for the probability of taking different conditional branches can be provided, and the implicit assumption that these values are path independent, which they may not be.

3.2 SPTA based on Probabilities from Faults

Systems may exhibit probabilistic behaviour due to the occurrence of faults, either external to the microprocessor, for example due to sensor failures, or internal to it, for example due to faulty cache lines.

The initial work in this area by Hofig [62] in 2012 introduced a methodology for *Failure-Dependent Timing Analysis*. This combines static WCET analysis of the code with probabilities propagated from a safety analysis model. Thus a set of WCET bounds are obtained that are associated with particular situations, for example a combination of sensor failures, and probabilities that those situations will occur. These values can then be used to determine a valid WCET at any given acceptable deadline miss probability. The work assumes that sensor failures are independent. The authors argue that when failures are dependent then this dependence can be captured in the safety analysis resulting in revised probabilities for the different situations considered.

In 2013, Hardy and Puaut [58] presented a SPTA for systems with deterministic instruction caches in the presence of randomly occurring permanent faults affecting the RAM cells that implement the cache lines. This method first computes the WCET assuming no faults, and then determines an upper bound on the probabilistic timing penalty due to the additional fault induced cache misses. The timing penalty is derived independently for each cache set, and the results combined to obtain the overall penalty. The motivation for this work is that as microprocessor technology scales decrease so the probability of permanent RAM cell failures will increase. A journal extension by the same authors [59] examines the scalability of the method to larger cache sizes, and also covers the effect of different values for the failure probability of memory cells, reflecting different technology scales.

More recently in 2016, Chen et al. [31] presented a SPTA for systems with a random replacement cache subject to both transient and permanent faults. This approach uses a Markov Chain model to capture the evolution of cache states and their probabilities. The cache states are modified taking account of the impact of faults, and hence the resulting pWCET distributions obtained reflect the fault rates specified as well as the random replacement policy of the cache. The evaluation shows that the method produces tight bounds with respect to simulation results. The cache assumed is however only 2-way, hence it is not clear whether the method scales to larger cache associativities. In a subsequent paper, also in 2016, Chen et al. [29] addressed an issue with their previous work whereby increasing permanent fault rates substantially degrades the pWCET. In this work they compare the previous rule-based detection mechanism with a more complex method that uses Dynamic Hidden Markov Model detection. The former is simple to implement, but the latter is significantly more effective, providing better pWCET estimates for high fault rates.

3.3 SPTA based on Probabilities from Random Replacement Caches

Caches are small fast memories used to bridge the speed difference between the processor and main memory. Access times to cache can be in the region of 10 to 100 times faster than accesses to main memory, thus cache often has a significant impact on the execution time of programs. Much of the work on SPTA (and indeed MBPTA) has focussed on caches that use a random replacement policy. Before reviewing this work, we outline some fundamentals about caches and their operation.

Main memory, used to store both instructions (the program) and data, is logically divided up into memory blocks, which may be cached in cache lines of the same size. When the processor requests a memory block, the cache logic has to determine whether the block already resides in the cache, a *cache hit*, or not, a *cache miss*. To facilitate efficient look-up each memory block can typically only be stored in a small number of cache lines referred to as a *cache set*. The number of cache lines or *ways* in a cache set gives the *associativity* of the cache. A cache with N-ways in a single set is referred to as *fully-associative*, meaning a memory block can map to any cache line. At the other extreme, a cache with N sets each of 1-way is referred to as *direct-mapped*; here each memory block maps to exactly one cache line. With set-associative and direct mapped caches, a *placement policy* is used to determine the cache set that each memory block maps to. The most common policy is modulo placement which uses the least significant bits of the block number to determine the cache set. Since caches are usually much smaller than main memory, a *replacement policy* is used to decide which memory block (i.e. cache line in the set) to replace in the event of a cache miss. Replacement policies include Least Recently Used (LRU), Pseudo LRU (PLRU), FIFO, and Random Replacement. Early work by Smith and Goodman [121, 122] in 1983 considered FIFO, LRU, and random replacement caches, concluding that the performance of random replacement is superior to FIFO and LRU, when the number of memory blocks accessed in a loop exceeds the size of the cache. LRU is superior when it does not.

The origins of SPTA for systems with random replacement caches can be traced to initial work by Quinones et al. [106] and Cazorla et al. [26] which provided a simple analysis restricted to single-path programs. Subsequently, Davis and Altmeyer and their co-authors Griffin and Lesage developed more sophisticated and precise analysis that also covers multi-path programs [39, 7, 6, 52, 83, 82].

The early work of Quinones et al. [106] in 2009 explored, via simulation, the performance of caches with a random replacement policy compared to LRU. The evaluation involves programs with a number of functions (greater than the cache associativity) called from within a loop. The different cases explored correspond to different memory layouts for the functions. Here, a small number of layouts result in pathological access patterns (referred to as *cache risk patterns [95]*) where, assuming LRU replacement, each function evicts the instructions for the next function from the cache. The simulation results show that random replacement has better performance than LRU in these cases, and lower variability in performance over the range of memory layouts explored. Further, when the code is too large to fit in the cache, then LRU has relatively poor performance, with random replacement providing better cache hit rates and less variability across different layouts. The authors suggest a simple method of statically computing the probability of pathological behaviour in the case of a random replacement cache; however, this formulation was later shown to be optimistic (i.e. unsound) by Davis et al. [39], since random replacement does not eliminate all of the dependences on access history.

In 2011, Burns and Griffin [23] explored the idea of predictability as an emergent behaviour. They show that if components are designed to exhibit independent random behaviour, then an execution time budget with a low probability of failure can be estimated that is not much greater than the average execution time. Their experiments examine hypothetical programs made up of instructions with execution times that are independent and identically distributed (i.i.d). The execution of each instruction is assumed to take either 1 cycle (representing a cache hit) or 10 cycles (representing a cache miss), with a 10% probability of the latter. Thus the average execution time of an instruction is 1.9 cycles. The authors found that for programs of more than 10^5 instructions, the execution time budget at an exceedance probability of 10^{-9} was only a few percent larger than the average execution time. They note, however, that current hardware does not result in instructions with random execution times.

A simple SPTA for caches with an *evict-on-access*[10] random replacement policy based on reuse distances was presented by Cazorla et al. [26] in 2013. This analysis upper bounds the probability of a miss on each access in a way that is independent of the execution history, thus enabling the distributions for each access to be combined using basic convolution to produce a pWCET distribution for the execution of a trace of instructions, i.e. a single path through the program. Comparisons are made with analysis for a LRU cache, showing that random replacement is less sensitive to missing information. Reineke [108] later observed that the formula given for a random replacement cache reduces to zero whenever the re-use distance is equal to or exceeds the cache associativity. Hence, in a like-for-like comparison, analysis for LRU strictly dominates that for random replacement presented in this paper. We conclude that the results given by Cazorla et al. are somewhat misleading; they are an artefact of the difference in associativity rather than a difference in replacement policy, since 2-, 4-, and 8-way LRU caches are compared to a fully-associative random replacement cache. We note; however, that when more precise analysis of a random replacement cache is used there are some circumstances when it can outperform state-of-the-art analysis for LRU, as shown by Altmeyer et al. [6] in 2015.

In 2013, Davis et al. [39] introduced a SPTA technique based on re-use distances that is applicable to random replacement caches that use an *evict-on-miss policy*[11]. This dominates the earlier analysis of Cazorla et al. [26] which assumes evict-on-access. The authors show that it is essential that any analysis providing probability distributions per instruction that are then convolved to form a pWCET distribution for the program must provide distributions for each instruction that are *independent* of the pattern of previous hits or misses, otherwise the analysis risks being unsound (i.e. optimistic). The authors extend their analysis to cover multi-path programs and the effect of preemptions. By taking a simplified view of preemption, effectively considering that it flushes the cache, they derive analysis that bounds the maximum impact of multiple preemptions on a program and show how computation of probabilistic cache related preemption delays (pCRPD) can be integrated into SPTA. We note that although the analysis is presented for fully associative caches, it is also applicable to set-associative caches, since each cache set operates independently.

Subsequently in 2014, Altmeyer and Davis [7] introduced effective and scalable techniques for the SPTA of single path programs assuming random replacement caches that use the evict-on-miss policy. They show that formulae previously published by Quinones et al. [106] and Kosmidis et al. [68] for the probability of a cache hit can produce results that are optimistic and hence unsound when used to compute pWCET distributions. In contrast, the formula given by Davis et al. [39] is shown to be optimal with respect to the limited information it uses (reuse distances and cache associativity), in the sense that no improvements can be made without requiring additional information. The authors also introduce an improved technique based on the concept of *cache contention* which relates to the number of cache accesses within the reuse distance of a memory block that have already been considered as potentially being a cache hit. An exhaustive approach is also derived that computes exact probabilities for cache hits and cache misses, but is intractable. They then combine the exhaustive approach focussing on a small number of the most *relevant* memory blocks with the cache contention approach for the remaining memory blocks.

Altmeyer et al. [6] extended their earlier work [7] in a journal publication in 2015. In this paper, they introduce a new formula for the probability of a cache hit based on stack distance, correct an error in the formulation of basic cache contention, and provide an alternative cache

[10] The evict-on-access random replacement policy evicts a randomly chosen cache line whenever an access is made to the cache, irrespective of whether that access is a cache hit or a cache miss.

[11] The evict-on-miss random replacement policy evicts a randomly chosen cache line whenever an access is made to the cache and that access is a cache miss.

contention approach based on the simulation of a feasible evolution of the cache contents. Both cache contention approaches are formally proven correct, and are shown to be incomparable with the new stack distance approach. They also present an alternative more powerful heuristic for selecting relevant memory blocks, with blocks no longer considered as relevant once they are no longer used. This improves accuracy, while ensuring that the analysis remains tractable. The evaluation shows that the cache contention techniques improve upon simple methods that rely on reuse distance or stack distance, and that the combined approach with 4 to 12 relevant memory blocks makes further substantial improvements in precision. Specific comparisons with LRU show that the simple reuse distance and stack distance approaches are always outperformed, in fact dominated, by analysis for LRU. In contrast, the sophisticated combined analyses for random replacement caches are incomparable with those for LRU. LRU is more effective when the number of memory blocks accessed in a loop does not exceed the associativity of the cache, whereas random replacement is more effective when they do.

In 2014, Griffin et al. [52] applied lossy compression techniques to the problem of SPTA for random replacement caches. They built upon the exhaustive collecting semantics approach developed by Altmeyer and Davis [7], exploiting three opportunities to improve runtime while trading off some precision: (i) memory block compression, (ii) cache state compression, and (iii) history compression. Two methods of memory block compression were considered. The first excludes memory blocks with a hit probability below some threshold, while the second excludes those with a forward reuse distance that exceeds a given threshold. Both cache state and history compression are applied via the use of fixed precision fractions. The lossy compression techniques are highly parameterisable enabling a trade-off between precision and runtime. Further, the runtime is linear in the length of the address trace, compared to quadratic complexity for the combined technique derived by Altmeyer and Davis [7].

An effective SPTA for multi-path programs assuming a random replacement cache was developed by Lesage et al. [83] in 2015. This work adapts the cache contention and collecting semantics approach derived by Altmeyer and Davis [7] to the multi-path case. It also introduces a conservative join function which provides a sound over-approximation of the possible cache contents and pWCET distribution on path convergence. The analysis makes use of the control-flow graph. It first unrolls loops, since this allows both the cache contention and collecting semantics to be performed as simple forward traversals of the control-flow graph. Approximation of the incoming cache states on path convergence, via the conservative join operator, keeps the analysis tractable. The distributions obtained via the cache contention and collecting semantics are then combined using convolution. The main analysis technique is supplemented by *worst-case execution path expansion*. This method uses the concept of *path inclusion* to determine when one path includes another, necessarily resulting in a larger pWCET distribution. This enables the included path to be removed from the analysis. This concept simplifies the analysis of loops, since only the maximum number of loop iterations need be considered. The evaluation shows that this multi-path SPTA reduces the number of misses predicted for complex control flows by a factor of 3 compared to the simple path merging approach proposed earlier by Davis et al. [39]. Incomparability between analysis for LRU and sophisticated analysis for random replacement caches is also demonstrated on multi-path programs. The authors also investigate the runtime of their methods, showing that it is tractable with 4-12 relevant memory blocks. To reduce the runtime for long programs they also consider splitting such programs into Single Entry Single Exit regions[12] which can be analysed separately, with the pessimistic assumption of an empty cache at the start of each region. This approach is shown to be effective, permitting the use of more relevant memory blocks and

[12] An idea first suggested by Pasdeloup [104].

hence in some cases improving precision. In a journal extension, Lesage et al. [82] incorporate advances in SPTA techniques for single path programs into the analysis framework for multi-path programs.

In 2017, Chen and Beltrame [28] derived a SPTA for single path programs running on a system with an evict-on-miss random replacement set-associative cache. They use a non-homogeneous Markov chain to model the states of the system. For each step (i.e. memory access in the trace of the program), the method computes a state occurrence probability vector for the cache and a transition matrix which determines how the state vector is transformed. The computational complexity of the basic method increases as a polynomial with a large exponent given by the number of memory addresses. To contain the complexity and make the approach tractable, the authors adapt the method as follows. For the first n addresses, the state space is constructed using the Markov chain method as described above. Then when a new memory address is accessed that is not in the current state, another memory address that is part of the state and is either not used in the future, or has the longest time until it is used in the future, is discarded. This is a form of lossy compression that ensures the number of states does not increase further. The authors show how the method can be adapted to write-back data caches, and compare its performance with the SPTA method of Altmeyer at al. [6]. The evaluation shows that the adaptive Markov chain method provides improved performance, with the geometric mean of the execution times at an appropriate probability of exceedance reduced by 11% for the set of Mälardalen benchmarks studied. The runtime of the two methods is shown to be similar.

Also in 2017, Chen et al. [30] developed a SPTA for random replacement caches with a detection mechanism for permanent faults. The detection mechanism periodically checks the cache for faulty blocks and disables them. The analysis builds upon the combined approach of Altmeyer et al. [6]. The authors derive formula for two operating modes: with and without fault detection turned on. By combining these two modes, the method provides timing analysis accounting for the integrated fault detection. The experimental evaluation provides a comparison with simulation for the Mälardalen benchmarks, assuming a 2-way, 1 KByte instruction cache, with 16 byte cache lines, and a permanent fault rate that equates to a mean time between failures of 5 years. Other experiments consider higher fault rates, larger cache lines and a 4-way cache. The results show that when sufficient relevant blocks are used, SPTA provides results that are close to those obtained via simulation.

3.4 Summary and Perspectives

Static Probabilistic Timing Analysis (SPTA) for systems using random replacement caches has matured considerably since its origins in the work of Quinones et al. [106]. State-of-the-art techniques by Altmeyer et al. [6], Chen and Beltrame [28] and Lesage et al [83, 82] provide effective analysis for single and multi-path programs respectively on systems using set-associative or fully-associative random replacement caches. This analysis is however limited to systems with single-level private caches. As far as we are aware SPTA techniques have not yet been developed for multi-level caches or for multi-core systems where the cache is shared between cores. A preliminary discussion of these open problems can be found in the work of Lesage et al. [84] and Davis et al. [40] respectively.

4 Measurement-Based Probabilistic Timing Analysis (MBPTA)

In this section, we review Measurement-Based Probabilistic Timing Analysis (MBPTA) methods. These methods use statistical techniques based on Extreme Value Theory (EVT) to derive an estimate of the pWCET distribution of a program from a sample of execution time observations.

These observations are obtained by executing the program on the hardware or a cycle accurate simulator under a measurement protocol. The measurement protocol executes the program multiple times according to a set of test vectors and initial hardware states, thus sampling one or more possible scenarios of operation.

Whether or not useful results can be determined using MBPTA methods depends crucially on the sample of execution time observations obtained and their properties. Early work in this area required execution time observations to be independent and identically distributed (i.i.d.). Later work relaxed this assumption, but still required that the sequence of execution time observations exhibit stationarity and weak dependences. Further, the results are only valid for future scenarios of operation for which the sample of observations is representative (see Section 2.3).

In the following subsections, we review: (i) early research into MBPTA that requires execution time observations to be i.i.d., (ii) subsequent research that relaxes the i.i.d. assumption, (iii) research which focusses on issues of representativity.

4.1 EVT and i.i.d. observations

In this section, we review early work on MBPTA based on the application of Extreme Value Theory that assumes the execution time observations are i.i.d. This work began with a number of papers by Burns and co-authors Edgar and Griffin [22, 45, 51], and culminated with the work of Cucu-Grosjean et al. [34] which inspired a substantial body of subsequent research into MBPTA.

In 2000, Burns and Edgar [22] introduced the use of EVT (the *Fisher-Tippett-Gnedenko* theorem) in modelling the maxima of a distribution of program execution times. They motivate the work by noting that advanced processor architectures make it prohibitively complex or pessimistic to estimate WCETs using traditional static analysis techniques. The following year, Edgar and Burns [45] introduced the statistical estimation of the pWCET distribution for a task. They use measurements of task execution times to build a statistical model. These observations are assumed to be *independent*, obtained over a range of environmental conditions, and of sufficient number for generalisations to be applied. The method they propose involves fitting a Gumbel distribution directly to the observations, using a χ-squared test to determine the scale and location parameters. Such direct fitting is however not strictly correct, since the distribution is used to model the *maxima* of subsets of values, not all of the values themselves, as noted by Hansen et al. [57]. Further, there are also issues in using a χ-squared test, which is not well suited to this purpose, as noted by Cucu-Grosjean et al. [34]. The authors show that if all tasks are executed with execution time budgets that must sum to some fixed total, then the optimal setting for the budgets achieves that total with an equal probability of each task exceeding its budget.

In 2009, Hansen et al. [57] considered the use of EVT to estimate pWCET distributions with an appropriate level of confidence. They correct a key issue in the work of Edgar and Burns [45] by using the *Block Maxima* method. Here the observations of execution times are first grouped into blocks, then the maximum value is taken for each block. The Gumbel distribution is then fitted to the distribution of the block maxima, with a χ-squared test used to determine goodness of fit. The experimental evaluation presented involves over 100 tasks running on a commercial PowerPC-based device using the VxWorks operating system. The block size is initially set to 100, and doubled repeatedly if the χ-squared test does not indicate a sufficiently good fit. (We note that this doubling may have been sufficient to capture stationarity in the observations, see the later work of Santinelli et al. [112]). The Gumbel parameters were obtained by using linear regression on a QQ-plot. The results produced using EVT were validated against additional measurements (over 2 million in the case of one task reported upon in detail). These results show that the EVT method provides a much better estimate of the extreme values with respect to subsequent observations, than simply taking the maximum observed value and assuming that it is

the WCET. Nevertheless, there were some tasks where the rate at which execution times (captured during the validation stage) exceeded some large value was greater than the exceedance probability predicted by the Gumbel distribution, i.e. the pWCET distribution was optimistic.

The use of EVT to model the extreme execution times of programs was considered by Griffin and Burns [51] in 2010. They provide a critique, discussing a number of issues that need to be addressed in order for the results produced to be sound. These include issues relating to the use of upper bounds (e.g. Gumbel distribution) that are continuous when the possible execution times of a program may have large discrete steps, and the need for the observations used to be *independent and identically distributed* (i.i.d). They describe various ways in which observations may be dependent (e.g. via internal state such as cache contents, and also via external factors such as input variables that are affected by the previous outputs of the system leading to a history dependence). They also note that different paths through the program lead to different execution time distributions, and hence issues arise with the assumption that all the observations are identically distributed. Some possible solutions are suggested, such as shifting the 1 - CDF of the continuous distribution so that it upper bounds its discrete counterpart at all execution time values, including those that cannot actually be obtained. Possible solutions to issues with the i.i.d. assumption include resetting the system state between observations, and determining the set of potential worst-case paths by some other means, and then analysing each of those paths separately, i.e the per-path approach (discussed in Section 2.3).

In 2011, Lu et al. [89, 90] examined issues with the application of EVT highlighted by Griffin and Burns [51]. They address the requirement that the observations must be i.i.d. They note that a Gumbel distribution should not be directly fitted to the observations as done by Edgar and Burns [45], since the Gumbel (and other EV distributions) are intended to model distributions of the maxima (or minima) of a large number of random variables. Instead, the authors propose the use of Simple Random Sampling in terms of randomising program inputs, and dividing observations into groups of N sets, each of m observations, and only using the largest of the m observations in each set, i.e. the *Block Maxima* method. We note that Simple Random Sampling may break dependences in the observations with the potentially for pWCET estimates to then be optimistic. (The authors give no proof that re-sampling the observations in this way ensures a sound, i.e. pessimistic result).

In a seminal paper published in 2012, Cucu-Grosjean et al. [34] introduced a statistically sound method of applying EVT to probabilistic timing analysis, referred to in the paper as Measurement-Based Probabilistic Timing Analysis (MBPTA). This method uses end-to-end execution time measurements obtained from the system during testing. For the method to be applicable, the observations must be i.i.d. Statistical tests (two sample Kolmogorov-Simirnov test, and Runs test) are used to check that this is the case. The Block Maxima method is used to group the observations, and the Exponential Tail test is used to check if the distribution of the maxima fits the Generalised Extreme Value (GEV) distribution, in particular, a Gumbel distribution. The method also proposes an approach for determining the number of observations required for each program path. In applying MBPTA to multi-path programs, the authors note that *path coverage* is required, otherwise the requirements for i.i.d. observations would be broken, and give an example of how the results could be unsound in this case. With full path coverage; however, the method appears to be relatively insensitive to the number of observations on each path, provided there are sufficient of them. The authors further note that for the i.i.d. property to be preserved for multi-path programs, either the inputs can be selected randomly when making measurements and the observations grouped sequentially, or testing can be done with all possible inputs and then observations selected via random sampling without replacement when performing the grouping into blocks. We note that this may bias the block maxima towards the worst-case,

since if high execution times are grouped, random sampling may increase the likelihood that a block contains a high value. This can potentially result in both pessimism and optimism in the pWCET distribution, since it changes the shape of the distribution of the block maxima. The authors evaluate the method for both single-path and multi-path programs running on a simulated microprocessor with a random replacement cache. Comparisons against the simple SPTA method of Cazorla et al. [26] show that the pWCET distribution obtained via MBPTA typically overestimates that derived via SPTA by 3-15% at small probabilities. We note that since later work by Altmeyer et al. [6] shows that the SPTA method described by Cazorla et al. [26] typically produces results that have substantial pessimism, substantive conclusions cannot be drawn about the absolute accuracy of the MBPTA method from these comparisons.

4.2 EVT and observations with dependences

The initial work on MBPTA by Cucu-Grosjean et al. [34] in 2012 resulted in increased interest in this area of research. Subsequent developments have focussed on relaxing the i.i.d. assumption and thus facilitating analysis of systems where execution time observations exhibit dependences. Much of the work in this area emanates from Santinelli and his co-authors Melani, Berezovskyi, and Guet [93, 112, 16, 53, 54]. They leverage early work by Leadbetter et al. [81] in 1978 and Hsing [63] in 1991 showing that EVT can be applied to stationary and weakly dependent time series.

In 2013, Melani et al. [93] investigated the use of statistical methods based on EVT when execution time observations show dependences on some other event in the system. For example preemptive rather than non-preemptive scheduling, or different preempting tasks that either heavily load the processor or the cache. They suggest a characterisation of dependences by effectively shifting, by an amount Δ, the pWCET distribution for a program as obtained in isolation. Evaluation shows that this method is effective for code from the Maladarlen benchmark suite [56] running on an Intel Xeon processor with 3 levels of cache. A small shift, compared to the overall execution time, is sufficient to account for the effect of the events which the execution times are dependent on. We note that shifting the pWCET distribution in this way equates to adding a fixed overhead to account for the extra interference due to preemption and scheduling.

The case for applying EVT when there are dependences between observations was examined by Santinelli et al. [112] in 2014. They note the prior work by Leadbetter et al. [81] and investigate less constraining hypotheses than independence such as stationarity and extremal dependence. To verify stationarity, autocorrelation is computed using lag plots. Experimental evaluation on an Intel Xeon processor with four cores (per socket) and 3 levels of cache shows that there is sufficient execution time variability for EVT to be applied. Observations from multi-path code exhibit stationarity, but pass the tests required in order for EVT to be applied. This indicates that the use of time-randomised hardware (e.g. random replacement caches) is not necessary in order to apply EVT. They also discuss dependence between extreme samples. This can be depicted using an *extremogram* (described by Davis and Mikosch [36]), which shows whether extreme samples are correlated in terms of their separation, i.e. whether they are clustered, or distributed throughout the trace of observations. Further, the authors examine the effect on the pWCET distributions of choosing different values for the size of the blocks in the Block Maxima method and different thresholds in the Peaks-Over-Threshold (PoT) method. Significant sensitivity is demonstrated to these values, with a degradation in performance at larger block sizes. It is not clear whether this is because the total number of samples is kept constant and thus the larger block sizes equate to too few blocks. Similarly, using too high a threshold (equating to the 98 percentile) results in a very large pWCET estimation. Again it may be that too few samples remain. We note that while this work provides useful insight into the importance of appropriate parameter selection, no guidance is given on how such parameters should be selected to achieve accurate results.

In 2014, Berezovskyi et al. [16] investigated the use of MBPTA techniques based on EVT to estimate the pWCET of CUDA kernels running on an NVIDIA GPU. They discuss the use of the Generalised Extreme Value (GEV) distribution, and developments that show that independence of observations is not necessary for the application of EVT. They note the prior work by Leadbetter et al. [81] and Hsing [63] and recount theorems relating to EVT for sequences with *long range independence* and *extremal independence* [81], and note that randomness is not sufficient in itself to show independence in these cases. Further, the Runs test previously used by Cucu-Grosjean et al. [34] does not suffice; rather time series[13] tests based on auto-regression and autocorrelation are needed. The authors suggest the use of lag plots to determine autocorrelation. They use autocorrelation tests together with the notion of stationarity to quantify statistical independence. They also use an autoregressive model to determine the stationarity of a trace of observations. The Ljung-Box test is used to look for correlations between lags. Finally, extremeograms are used to estimate dependence at the extreme values. The evaluation uses a Kepler GK104 GPU. Tests on the trace of observations for the GPU computations show them to be independent. By contrast, the observations of the execution times including data transfer to and from the host processor are not independent, but they are stationary. Here, the Runs test would have concluded independence; however, in fact there is a strong stationary relationship, but since this does not reflect as dependence in the extreme observations, EVT can still be applied.

In 2016, Guet et al. [53] proposed a logical work-flow (embedded in a tool called DIAGXTRM) that checks the applicability of EVT for the pWCET estimation problem. This work discusses Pickands-Balkema-de Haan Theorem [105], the Generalised Pareto Distribution (GPD), and the Peaks-Over-Threshold (PoT) method. The authors again note the prior work by Leadbetter et al. [81] showing that EVT is applicable in the more general stationary case without independence. Here, the hypotheses to check on the trace of observations are (i) stationarity, (ii) short range dependence, (iii) local independence of the peaks, and (iv) that the empirical peaks over the threshold follow a GPD. The efficiency of the proposed logical work-flow is likely affected by the relation between theses hypotheses, which are not independent. Moreover, the authors motivate the introduction of the hypothesis of stationarity based on the impossibility of checking the identically distributed hypothesis when the bounds on the execution time of a program have unknown probability distributions. Here, we note that the arguments are not correctly used as stationarity is an alternative to the hypothesis of statistical independence, rather than an alternative to the identically distributed hypothesis, further EVT is itself applied only to those cases where the measurements follow unknown probability distributions. To evaluate stationarity, the authors use the Kwiatowski–Philips–Schmit–Shin (KPSS) test. Further, the Brock–Dechert–Scheinkman test is used to evaluate short range dependence, by examining the correlation between different sub-sequences of the same length. The reliability of EVT also depends on the independence of extreme observations. Here, the *Extreme Index* (see Embrechts et al. [46]) is used to give the probability that peaks are far enough apart to be considered independent, as opposed to constituting a burst relating to a single rare event. The threshold selection in the PoT method is a source of uncertainty, since different thresholds may lead to different pWCET estimates. Reviews of threshold selection methods by Scarrott and MacDonald [114] show that there is no recognised systematic process for selection. The authors therefore propose an approach that aims to provide an appropriate tail sample. They evaluate their approach using execution time measurements obtained from an Intel Xeon processor with 4 cores and 3 levels of cache. Code from the Maladarlen benchmark suite is run in various configurations including isolation, alternately with a program that makes heavy use of the cache, and with that program running on the other

[13] The time series here is the observations of execution times in the order in which they were made.

cores. All the traces of observations were found to be stationary with high confidence. Short range and extreme independence was also verified. Note, Berezovskyi et al. [15] later applied the PoT approach proposed by in this paper to the NVIDIA Kepler GK104 GPU system that they had previously investigated using the Block Maxima method [16].

Later in 2016, Guet et al. [54] investigated using measurement based probabilistic analysis to estimate the number of cache misses for programs running on a COTS multi-core processor with two Intel Xeon E5620 sockets, using 3 levels of deterministic PLRU caches, with the first two levels private to each core and the last level shared by cores on the same socket. Their aim was to estimate the worst-case number of cache misses for programs under different configurations: (i) isolation, (ii) on a single core alternating with another program on the same core, (iii) on one core with another program executing simultaneously on a different core, and (iv) combining (ii) and (iii). (Note, the other program makes a large number of memory accesses). The authors applied EVT to the problem using the Block Maxima method. They note that the observations of cache misses are unlikely to be independent, nevertheless, they show that the observations for most of the programs examined from the Mälardalen benchmark suite exhibited stationarity and/or extremal independence and so can be analysed using EVT.

In 2017, Fedotova et al. [47] applied the PoT method of EVT to estimate the upper bound values for a timer acquisition task. This involves setting two consecutive timers using the *HighPerTimer* library and calculating the time difference between them. The platform used runs the standard Linux kernel on ARM Cortex-A5 hardware. The authors discuss how the choice of threshold to use in the PoT method is a compromise between precision and bias, and make use of two graphical approaches for threshold selection: (i) mean residual life, which plots the mean excess against the threshold, and (ii) parameter stability, which plots the shape and modified scale parameters of the GPD against the threshold. Using these methods, they select an appropriate threshold which obtains 24 extreme values from the trace. They show that both Gumbell and Frechet distributions fit the data and compute pWCET estimates with various probabilities of exceedance for both cases. Due to the different shape parameters for the two distributions, these pWCET estimates exhibit large differences e.g. 7.7ms v. 5900ms at a probability of exceedance of 10^{-9}. These large differences raise questions about the variability of results that can be obtained from the same data, and hence the confidence in them. We note that 10^{-9} represents a large amount of extrapolation from the sampled data, with the pWCET estimates much closer together at a probability of exceedance of 10^{-7} (2.5ms v. 11.3ms).

Also in 2017, Lima and Bate [87] proposed a technique called IESTA (Indirect Estimation in Statistical Time Analysis) to support the application of EVT without the need for hardware or software randomisation. The basic idea is to add a randomised component (e.g. from a normal distribution with values in the range $[a, b]$) to the set of execution time observations. This additional component removes some of the discreteness from the original distribution, and makes the new values more likely to pass the statistical tests required for the application of EVT. Once the random component has been added to the observations, then the maxima are sampled, using either PoT or Block Maxima methods, and the pWCET estimated by fitting to a GEV distribution. Goodness-of-fit tests indicate if the estimate is a close match to the empirical distribution of the maxima or peaks over the threshold. If it is not, then the dispersal of the random component can be increased, which makes it more likely that EVT can be applied and the goodness-of-fit tests passed. The authors explore two benchmarks, one from the Mälardalen benchmark suite where EVT could not be applied even with time-randomised hardware, and a second using data from an aircraft control application from Rolls-Royce running on entirely time-predictable hardware. The latter has significant dependences between execution times, as shown in autocorrelation plots with lags of up to 40. For both benchmarks, the IESTA method is able to add a random component

enabling EVT to be applied. Both the random padding and a high threshold selection help make it unlikely that dependences appear in the sample of the maxima. The evaluation shows that even when the dispersal of the random component is large, then the increased pessimism in the analysis remains small (i.e only a few percent). It remains an open question whether this method may result in optimistic pWCET estimations.

4.3 EVT and representativity

Representativity is a fundamental issue in applying statistical methods (i.e. MBPTA) to estimate the pWCET distribution of a program. The problem is that the results obtained are only valid for those scenarios of operation for which the sample of observations used in the analysis is representative (see Section 2.3). Often it may not be possible to devise a measurement protocol that provides a single sample of observations that is representative of all possible future scenarios of operation. Research considering the problems of representativity are reviewed below.

In 2016, Lima et al. [88] took a careful look at the use of EVT in determining pWCET bounds for programs running on both time-predictable (deterministic) and time-randomised architectures (i.e. with random replacement caches). They point out that execution time observations depend on the probability distribution describing how often different input values occur. Thus there may not be a single distribution that describes the execution time behaviour. The authors therefore introduced the concept of a *weak pWCET* which is valid for a particular distribution $D(\tau_i)$ describing the frequency of occurrence of input values. This raises the problem of *representativity* of input data. A simple experimental example shows how the estimated pWCET distribution produced depends on the distribution of input values, and hence that if the input data distribution used for analysis does not match that occurring during operation of the system, then the results obtained may be unsound. Applying uniformly distributed input values during analysis may also result in poor or unsound estimates. The authors note that in some cases it is not possible to estimate a reliable Generalised Extreme Value (GEV) model and they cite appropriate extreme value condition tests that can be used to determine if this is the case. They show that the GEV distribution should be used rather than assuming a specific distribution (e.g. Gumbel). The models determined for various experiments belong to all three classes of EV distribution (reversed Weibull, Gumbel, and Frechet) not just Gumbel. There were also cases where the models belonged to none of them and EVT could not be used to provide sound results. The experimental results reported in this work also show that EVT can be successfully applied on time-predictable platforms, thus indicating that hardware randomisation is not necessary for the application of EVT. They also found that hardware randomisation is also not in general sufficient to ensure that EVT can be applied. Random replacement caches that were either too large or too small were observed to make estimation of the pWCET distribution either harder or not possible at all.

Issues of *reproducibility* and *representativity* with respect to MBPTA methods were discussed by Maxim et al. [92] in 2016. They consider two separate steps: (i) the measurement protocol, i.e. obtaining execution time observations and (ii) the method used to estimate the pWCET distribution. They note that to be useful, the estimation method must be reproducible in the sense that it must provide, within a close approximation, the same pWCET distribution, given the same set of observations. Further, the measurement protocol must also be reproducible, in the sense that the two sets of observations that it provides for two different uses of the method, starting from the same conditions (processor state, external inputs etc.) must result in the same or sufficiently close pWCET distributions. The authors also consider representativity. They note that a measurement protocol is representative if there is some value of k (the number of observations) for which taking any larger number of observations results in a pWCET distribution that closely approximates the distribution that would be obtained if the whole domain of possible observations

were considered. In other words after k observations the method *converges* on a sound pWCET distribution. The properties of reproducibility and representativity are shown to be mandatory for convergence and hence required for any MBPTA method intended for use in practice. How to obtain these properties is; however, left as an open issue.

MBPTA techniques were revisited by Santinelli et al. [110] in 2017. In this work, they apply EVT to a case study comprising traces of observations from various example systems using both time-randomised and time-predictable hardware. They discuss the validity of EVT with respect to different execution conditions (similar to the issues of representativity raised by Lima et al. [88]) and suggest two ways of integrating all possible execution conditions into EVT: (i) trace merging and (ii) the use of an envelope (i.e. upper bounding the results for each separate execution condition). We note that it is not clear whether trace merging always produces valid results. The authors further propose a reliability measure based on the confidence levels of each of the hypotheses. They show that confidence is lower if the distribution is artificially forced to fit a Gumbel distribution rather than obtaining the best fit for the shape parameter. They also discuss how to choose the threshold in the PoT method, suggesting that a threshold should be selected that maximises confidence in the matching hypothesis with the largest amount of independent peaks.

In 2017, Santinelli and Guo [111] proposed a probabilistic representation framework, modelling tasks via multiple pWCET distributions. They noted that the pWCET distributions obtained via EVT strongly depend on the execution conditions used for analysis. These execution conditions describe aspects such as the environment, inputs, task mapping, presence of faults etc. The authors therefore propose describing each task via a *Worst-Case Set* which is a collection of pWCET distributions obtained via EVT for different execution conditions. They note that the number of different execution conditions is finite, and a partial ordering may be possible between some of them, indicating dominance relationships (in the sense of the greater than or equal to operator $\succeq$ defined by Diaz et al. [44]). They note that enumerating all possible execution conditions is a complex problem (since there may be very many of them); however, tasks could be represented by pWCET distributions which bound those for collections of execution conditions with incomparable pWCET distributions. The authors propose the use of their model for mixed criticality systems, with different execution conditions for different criticality levels, and for systems with and without faults.

The problems of applying EVT when execution time observations have a *mixture distribution*, resulting in a step-like curve on an exceedance (1 - CDF) graph, were discussed by Abella et al. [3] in 2017. (Note, mixture distributions can occur as a result of caches that employ random placement, discussed in Sections 6.1 and 6.2). In this case, it is important that sufficient observations are obtained and that the threshold (PoT method) or block size (Block Maxima method) is set appropriately, such that sufficient values from the actual tail of the distribution are passed to EVT, as opposed to including too many values from other parts of the distribution. The authors present a method for selecting the observations required and suitable EVT parameters. First, they argue that in real-time systems, programs have finite execution times and so heavy tailed GEV/GPD distributions may be discarded. Further, since somewhat pessimistic pWCET estimates are acceptable, but optimistic estimates are not, they argue that it is sufficient to consider distributions with exponential tails (i.e. Gumbel distributions) as a bound for all potential light-tailed distributions. They propose a method of determining appropriate parameters based on considering the Coefficient of Variation (CV) of the residual distribution, i.e. the distribution of the excess over the threshold in the PoT method. (Note, the CV is given by the standard deviation of a distribution divided by its mean). The method employs a plot of the CV versus the number of samples over the threshold to find a suitable number of samples in the tail distribution, for which the assumption of an exponential tail is not rejected by statistical tests and the CV

is closest to 1, implying an exponential tail. The approach is evaluated using a number of the EEMBC benchmarks, but considering only single paths. Execution time observations are obtained by making runs in isolation on a cycle accurate simulator for the Cobham Gaisler Next Generation Multicore Processor (NGMP) with modifications such that all caches use random placement and random replacement (see Section 6). The CV-plots show that in some cases 2000 or 3000 observations are required, compared to the default of 1000. Comparison with empirical distributions for 10^7 runs (generated on a large compute cluster) show that the pWCET estimates are 1% to 25.8% larger than the maximum observed values at a probability of exceedance of 10^{-7}.

In 2017, Milutinovic et al. [101] discussed issues relating to representativity with respect to the use of MBPTA in an industrial context. They argue, citing the above work of Abella et al. [3], that for real-time programs (with a finite but unknown WCET and non-degenerate behaviour) it is sound to fit a Gumbel distribution, since for this class of program using a Gumbel distribution may result in an over-estimate at probabilities of exceedance that are of interest, but not an under-estimate. Further, the authors point out, as was done by Griffin and Burns [51] in 2010, that taking observations from multiple paths as input into MBPTA (i.e. the per-program approach, see Section 2.3) may produce untrustworthy results. The problem being that the observations may no longer be identically distributed (particularly if the different paths are quite distinct modes of operation), and also the frequency of occurrence of the different paths may not match that which occurs during operation. The authors give a concrete example of this problem which shows that combining observations for two paths from a simplified European Train Control System can result in a pWCET distribution which is below the pWCET distribution for either path considered alone. (We note that no comparison is made to the empirical distribution, so it is unclear if any of the three pWCET distributions are actually under-estimates). The authors suggest that the per-path approach (see Section 2.3) can be used instead to provide a solution.

Also in 2017, Guet et al. [55] considered two issues relating to representativity. First, how traces of observations are handled when there are dependencies between execution times. In this case, the authors show that re-sampling the traces of observations can reduce the degree of dependency which is apparent and this may in turn result in lower pWCET estimates which can be unsound (i.e. optimistic). Secondly, they consider the use of EVT with multi-mode tasks, where different modes of operation have distinct execution time distributions. Here, they show via examples that combining all of the observations into a single sample (i.e. covering all modes) can result in cases where either EVT cannot be applied, or it can result in unsound pWCET estimates. They infer that it is necessary to apply EVT to each mode separately and then form an envelope upper bounding the pWCET distributions for each mode to provide an overall pWCET distribution that is sound.

4.4 Summary and Perspectives

MBPTA methods have matured considerably since the early work of Burns and Edgar [22, 45]. In particular, we note the work of Cucu-Grosjean et al. [34] which spawned a large number of subsequent publications on supporting mechanisms and techniques (see section 6), and prompted further research into the application of EVT to the probabilistic timing analysis problem. The recent state-of-the-art has shown that time-randomised architectures (e.g. with random replacement caches) are neither sufficient nor necessary for the application of MBPTA methods [112, 88, 87]. Further, EVT can be applied when there are dependences between the observed execution times, provided that the series of observations exhibits stationarity and/or extremal independence [112, 16, 53, 54]. These are useful advances, since industry has a strong preference for methods that can be applied to Common-Off-The-Shelf (COTS) processors, rather than requiring specific (custom) hardware architectures. We note however, that there are significant hazards in applying MBPTA

methods. If potential sources of significant execution time variability do not exhibit this variability during analysis, i.e. such variation does not appear in the observations, then they will not be accounted for in the estimated pWCET distribution produced, which may as a result be optimistic (i.e. unsound).

A major open issue that remains is the problem of *representativity*. It is essential that the measurement protocol is representative of the future scenarios of operation that could occur in practice. In general this requires suitable coverage of the different input states, hardware states, and the worst-case path(s) through the program. While worst-case path coverage may be possible for relatively simple and well structured programs in domains such as aerospace, in general, the problem of identifying and exercising the worst-case path(s) cannot be left to the user. Research aiming at a solution to this problem is reviewed in Section 5.

The main rationale for using time-randomised hardware (e.g. random replacement caches, random permutation buses etc.) as well as software time-randomisation techniques (e.g. random placement) is to make it easier to provide an argument that the measurement protocol used during analysis is representative of the scenarios of operation that could occur in future. These enabling mechanisms and techniques are reviewed in Section 6.

We note that there remains significant scope for work on the application of EVT to the problem of estimating pWCET distributions. Currently, there are differing views on whether it is sufficient to assume that any program in a real-time system will have an execution time distribution which can be modelled as having a light or exponential tail, and hence a Gumbel distribution can be used as suggested by Abella et al. [3], or whether it is necessary to fit to a GEV distribution, since the execution time of some programs may exhibit heavy tails as shown by Lima et al. [88]. There has also been little work on the reproducibility of the method: whether small changes in the set of observations may propagate to large differences in the resulting pWCET distributions, and thus on how far the pWCET distributions can realistically be extrapolated to very low probabilities e.g. 10^{-9}, 10^{-12}, 10^{-15} and so on, while retaining confidence in the results.

5 Hybrid Techniques for Probabilistic Timing Analysis (HyPTA)

A number of researchers have sought to combine the advantages of static analysis and measurement based methods. The main advantage of measurement-based methods is that measurements record the precise timing behaviour of the real system, whereas static analysis relies on a model of that behaviour. For advanced processors, obtaining a precise model or even one that does not introduce significant pessimism may be very difficult. The main disadvantage of measurement-based methods is the difficulty in covering the worst-case behaviour, including covering the worst-case path(s) through the code. The MBPTA methods based on EVT provide a pWCET distribution which estimates the extreme execution time behaviour of future scenarios of operation, sample(s) of which were exercised during an analysis phase. It is however up to the user to provide the necessary input vectors to test all relevant paths, including the worst-case path. Typically, neither manually determining the worst-case path, nor obtaining full path coverage are feasible in practice for complex programs. Further, as shown by Lesage et al. [85], the analysis rapidly becomes optimistic when some paths are omitted. In this section, we review hybrid methods which seek to address this problem of *path coverage*.

5.1 HyPTA and the Path Coverage Problem

In a series of papers from 2002–2005, Bernat et al. [19, 18, 17] addressed the problem of path coverage in measurement-based execution time analysis by reducing it to the much simpler problem of basic block (i.e. statement) coverage. In these works, the authors introduce the concept of

Execution Time Profiles (ETPs) used to represent the *relative frequencies* of execution times for basic blocks of a program. They also provide a timing scheme for constructs such as conditionals and loops, and an algebra for combining *independent* ETPs to provide an ETP for the whole program. For sequential combination, this involves using the basic convolution operator. Further, they point out the importance of accounting for dependences between basic blocks when combining ETPs otherwise unsound results could be obtained. Potential sources of dependence include: low-level hardware features such as caches and pipelines, and high-level path dependences. They suggest using biased convolution[14] to account for unknown dependences. (We note that Ivers and Ernst [64] later showed that biased convolution is insufficient to produce a distribution that results in the worst-case exceedance probability for every possible value of the execution time budget). A tool set is described by Bernat et al. [18] that implements the mechanisms described above. The main components are: instrumentation, trace generation, trace analysis, structural analysis, and timing program generation. This tool set was the precursor and prototype for the RapiTime tool from Rapita Systems[15]. Later in 2005, Bernat et al. [17] examined the problem of combining the ETPs for two basic blocks of code A and B when there are dependences between them. Here, the authors propose applying *copulas* (representing the dependences) to the marginal (i.e. separate) distributions of A and B to obtain the joint distribution.

In 2014, Kosmidis et al. [70] introduced the idea of Path Upper Bounding (PUB) for time-randomised hardware with a random replacement cache. With PUB, the program is modified so that it retains the same functionality, but has the same timing behaviour for every path, and that timing behaviour upper bounds that of the original program. PUB inspects each conditional in the program recursively and adds accesses such that all sub-paths of a particular conditional contain the same set of accesses. Once PUB has been applied, then the modified program is subject to MBPTA, with no particular attention needed to the input vectors used or the paths exercised, since all will result in a timing behaviour that upper bounds that of the original program. Evaluation, using the EEMBC benchmarks, shows that PUB increases code size by 20-100% depending on the amount of nesting and conditionals present, with the pWCET estimate at an appropriate probability of exceedance increased by up to 50%. We note that the application of PUB requires a trusted or certified implementation, since it modifies the code of the original program to create the version that is tested for timing correctness. Further, the method only works for systems that use a single-level random replacement cache.

The Extended Path Coverage (EPC) approach proposed in 2015 by Ziccardi et al. [136] also addresses the issue of path coverage via a hybrid approach. EPC requires that execution time measurements are made at the basic block level, while also recording the path taken, and that sufficient coverage is obtained to provide an Execution Time Profile (ETP) for each basic block. The ETPs are then modified to discount any benefit that may have accrued due to the path taken when the measurements were made. This is done by computing a probabilistic padding via the SPTA techniques developed by Altmeyer and Davis [7], considering the path actually taken and the memory accesses in the immediately preceding basic blocks. Synthetic end-to-end observations for all feasible paths can then be generated by randomly sampling the ETPs for the relevant basic blocks along each path. These synthetic observations are then fed into MBPTA. The authors evaluated EPC compared to directly applying MBPTA. For single path code there was no difference. For multi-path code with a known worst-case path, the EPC method resulted in pessimism of up to 30%. Comparisons with PUB [70] showed that EPC achieved on average

[14] Biased convolution combines values from two distributions assuming perfect correlation with respect to the percentile of the execution time, i.e. largest correlated with largest, smallest with smallest etc.
[15] `https://www.rapitasystems.com/products/rapitime`

similar performance with a +/-30% variation apparent between the two methods. EPC has the advantage with respect to directly applying MBPTA that it reduces the burden of path coverage, and the advantage over PUB that it does not require changes to be made to the executable code in order for measurements to be taken. EPC does however inherit some of the disadvantages of SPTA in that it needs to know the addresses of accesses to determine which cache sets they map to, and so compute the padding. Further, fine grained observations are needed at the basic block level, and accompanying structural analysis to re-construct the possible paths. EPC was developed for systems with separate, single level data and instruction caches, given the current lack of SPTA for multiple levels of cache (discussed in Section 3.4), it is not clear if the method could be extended to more complex memory hierarchies. Like PUB, EPC only applies to systems with random replacement caches.

5.2 Summary and Perspectives

A limited amount of research has aimed at addressing the path coverage problem via the use of hybrid methods [19, 70, 136]. However, these methods have some drawbacks, initial work by Bernat et al. [19] recognises the key issue of dealing with dependences between the measurements obtained for different basic blocks, but is unable to provide a practical approach for dealing with them. Later work on Path Upper Bounding (PUB) [70] requires substantial changes to the executable code, inflating code sizes by up to 100% and execution times by up to 50%, and requiring a trusted or certified implementation. Finally, the Extended Path Coverage (EPC) method [136] makes use of SPTA to compute a probabilistic padding to account for the dependences between the execution times of basic blocks due to the random replacement cache. EPC thus inherits some of the disadvantages of SPTA in that it needs to know the addresses of accesses to determine which cache sets they map to, and so compute the padding. Both PUB and EPC methods only work with random replacement caches.

In general, the issue of path coverage in relation to MBPTA remains unsolved. One approach for simple systems with few paths is to apply EVT on a per-path basis, i.e. to traces of observations taken for each specific path separately, thus producing a pWCET distribution for each path. The overall pWCET for the program can then be obtained as a tight upper bound on the pWCET distributions for all of the constituent paths. This *envelope* approach is well-suited to systems where there are a limited number of paths and the user can provide input vectors which exercise all of them. For many systems this is however unrealistic. The alternative is to apply EVT per-program i.e. to a sample of observations that cover all of the different paths. (Note, all paths need to be covered or the pWCET estimates can quickly become unsound as shown by Lesage et al. [85]). This approach raises difficulties in applying MBPTA, since the resulting observations will most likely form a mixture distribution (with the different distributions from the different paths contributing to the mixture), and thus care needs to be taken to ensure that sufficient observations are obtained from the tail of the distribution, i.e the worst-case path(s). This can be heavily impacted by the frequency of occurrence of different paths during analysis, and thus gives rise to issues regarding representativity. Further research is needed in this area.

6 Enabling Mechanisms and Techniques

MBPTA methods [34] based on EVT work well when the observations are able to characterise the different sources of execution time variability and their probability of occurrence. As an example, consider a program running on hypothetical time-randomised hardware where each instruction takes an random amount of time to execute, from 1 to 6 cycles, independent of all of the other instructions. Assume the program has 10 instructions. The overall execution time will behave

like the sum of the values of 10 fair dice. While the worst-case (i.e. 10 sixes) is unlikely to be observed (probability $\approx 10^{-8}$), EVT is able to estimate the tail of the distribution from a relatively small number of observations (e.g. 1000). Contrast, this with a hypothetical time-predictable system, here the execution time might depend on say 10 different input variables (each with values from 1-6). Let us assume that there is some small variability in overall execution time which depends on the input values themselves e.g. the overall execution time is 10, 20, 30, 40, 50, or 60; however, if all 10 inputs take a specific combination of values then there is some conflict in the hardware which slows processing down and the execution time is 600. The probability of observing this via random testing of the inputs is $\approx 10^{-7}$. In this case, applying EVT on 1000 observations would most likely result in a prediction based on the small variability that has been observed, but would obviously not account for the much larger variability that has gone unobserved. As a result, the pWCET estimate would be unsound. This is an example of a *needle-in-a-haystack* problem in testing. The intent of using time-randomised hardware is to avoid hazards similar to the one described above. These can potentially occur with time-predictable hardware such as LRU caches, when the unlikely, but not impossible, scenario of multiple input-data dependent accesses mapping to the same cache set occurs. We note that such hazards, referred to as *cache risk patterns*, can also occur with time-randomisation techniques such as random placement (see Section 6.3).

Since the work of Cucu-Grosjean et al. [34], considerable efforts have been expended to support MBPTA methods via the use of additional hardware and software time-randomisation mechanisms. These include hardware and software random placement for set-associative random replacement caches, random permutation buses, degraded test modes, and better random number generators. In this section, we review the supporting work in these areas.

6.1 Caches and Hardware Random Placement

Fully-associative caches are slower and use more energy than set-associative caches of the same overall size but a much smaller number of ways. This is due to the extra logic needed to check if a memory block is in any, as opposed to a small number of, cache lines. Because of this, in practice caches tend to either be direct mapped or to have 2, 4, 8 or 16 ways. The idea of using a random placement policy is to provide more randomisation with a set-associative random replacement cache than is possible with modulo placement, while improving access times and energy consumption compared to a fully-associative random replacement cache. Random placement can be achieved in hardware (via a randomised hash function used to control the mapping of memory blocks to the cache) or via software (through the use of compiler techniques and runtime support that randomise the positioning of code and data in memory). In each case, a random placement is selected at the start of each run of the program and remains in place for the duration of that run. The cache is then flushed between runs ready for a different random placement to be used on the next run.

The idea of random placement has its origins in the 1990s. In 1993, Schlansker et al. [115] first proposed random placement as a means of improving the cache miss ratio of matrix operations. In 1999, Topham and Gonzalez [124] proposed the use of polynomial modulus functions (operating on memory addresses) as a means of pseudo-randomising cache placement, in order to reduce the number of conflicts. We note that while these methods can avoid systematic conflicts due to particular code structures, they provide a deterministic mapping that depends on the memory addresses, and thus always produce the same placement for a given memory layout. (The placement does not change on each run of the program).

The majority of the work on hardware random placement for random replacement caches was developed in a series of papers from 2013–2016 by Kosmidis and co-authors including Abella, Cazorla, Quinones, and later Hernandez [68, 69, 67, 61].

In 2013, Kosmidis et al. [68] proposed the use of a hardware random placement policy as part of the cache design. The parametric hash function used aims to avoid systematic collision of two memory blocks in the same cache set. The evaluation considers random placement applied on top of set-associative random replacement caches, thus it is difficult to quantify the contribution of random placement versus random replacement. Where these factors can be separated e.g. for a 1 way – 256 set (i.e. direct mapped) cache with random placement, and a 256 way – 1 set (i.e. fully associative) random replacement cache, the results show that random placement results in a factor of 2.6 degradation in execution time performance (measured in terms of the average number of instructions per cycle) compared to modulo placement. By comparison, in the fully associative case, random replacement shows approximately 10% degradation in performance compared to LRU. In a journal extension in 2014, Kosmidis et al. [67] provided a detailed analysis of the quality of the hash function used for random placement, as well as more detailed evaluation results, including an investigation into energy consumption and cache access times.

As part of an analysis for caches with random placement and random replacement, Kosmidis et al. [68] gave formulae for the cache hit probability of a given access in a set-associative random replacement cache; however, as shown by Davis et al. [41], this formula cannot be used in SPTA since it may result in a pWCET distribution that is optimistic (i.e. unsound). Similarly, the formulae given for the cache hit probability assuming caches using random placement, or both random placement and random replacement also cannot be used in SPTA.

In a number of works, Kosmidis et al. [68, 74, 67]) describe the concept of an Execution Time Profile (ETP) as defining the different execution times of a program (or instruction) and their associated probabilities. They state that *"the existence of an ETP ensures that each potential execution time of the program (for MBPTA) or instruction (for SPTA) have an actual probability of occurrence, which is a sufficient and necessary condition to achieve the desired probabilistic i.i.d. execution time behaviour"* so that MBPTA can be applied. However, this is not entirely correct. The argument given in more detail in a white paper by Abella et al. [1] shows that the time-randomised architecture proposed ensures the existence of an ETP for a single path through the program starting from a fixed initial software and hardware state. Thus the observations *for that path and initial state* will be *independent and identically distributed* (i.i.d.). However, for a complete program, with multiple paths, this does not necessarily mean that the execution time observations collected over multiple paths for use in MBPTA will be i.i.d. As a simple example, consider a program that implements a state machine with states 1-5 that it cycles through in order. Each state may have an execution time that is quite different from the others e.g. 100 ± 20, 200 ± 20, $\ldots 500 \pm 20$, where ± 20 represents the random variation and the first value is determined by the state variable. Now when multiple runs of the program are performed, it cycles through the states, and hence the execution times observed are *not independent*, rather there is a strong correlation between them with a lag of 5. In cases such as this MBPTA may or may not be applicable; appropriate statistical tests are needed to determine if the execution times observed are actually i.i.d., as there is no relation between the statistical properties of the execution times of an instruction and the statistical properties of the execution times of an entire program containing that instruction. The architecture alone cannot ensure that the execution times will be i.i.d. for all real-time programs.

Later in 2013, Kosmidis et al. [69] applied MBPTA [34] to a system with multiple levels of cache that use random replacement along with a random placement policy implemented in hardware [68]. Here, separate L1 instruction and data caches are assumed, with a unified L2 cache. Assuming latencies for the L2 cache and memory of 10 and 100 cycles respectively, the evaluation shows that using a second level of cache improves the pWCET estimate at an appropriate probability of exceedance for the EEMBC benchmark code by 10-90% depending on the size of the working set.

The authors also give a series of formulae that approximate the probabilities of cache misses for different cache arrangements. These formulae relate to those from earlier papers [68, 67], which can be optimistic by orders of magnitude as shown by Davis et al. [41].

Building on the work of Kosmidis et al. [68], in 2014 Slijepcevic et al. [120] considered a time-randomised (i.e. both random placement and random replacement) last-level cache that is shared between cores in a multi-core system. The basic idea proposed is that a shared time-randomised cache makes the cache interference suffered by a task due to co-runners depend only on the frequency of co-runner cache misses (i.e. evictions) and not on the precise cache lines that are accessed. The authors present a hardware mechanism that limits the eviction frequency by enforcing a minimum inter-eviction delay for each core. The pWCET distribution of each task is then estimated at analysis time with this mechanism limiting its rate of cache misses (i.e. stalling execution if two misses would otherwise occur too close together), and synthetic co-runners causing the maximum possible rate of evictions (i.e. separated by exactly the minimum inter-eviction delay). To avoid problems due to systematic interleaving, the minimum inter-eviction delay is itself set to a uniform random value with the desired mean. Cache sharing with the proposed mechanism limiting contention is compared to cache partitioning into 4 partitions of 2 ways each. The latter results in estimated pWCETs at an appropriate probability of exceedance that are on average 56% higher.

In 2015, Anwar et al. [9] developed a VHDL implementation of random replacement and the hardware random placement policy proposed by Kosmidis et al. [68] for instruction and data caches, and integrated it with an Ion MIPS32 processor core on an FPGA. They used the Mersenne Twister algorithm, which is considered a good hardware solution for random number generation, replacing the Multiply With Carry random number generator previously proposed [68]. For a set of benchmarks operating on arrays and matrices, with nested loops, the authors claim that the time-randomised hardware gives an improvement of 6% and 19% over LRU for 8-way and direct mapped caches respectively, when comparing observed execution times. We note; however, that these figures use the worst-case execution times observed in 30 runs with the time-randomised hardware; whereas the predicted pWCET at an exceedance probability of 10^{-3} is consistently larger than the measurements shown for LRU.

An improved hardware random placement policy called *random modulo* was described by Hernandez et al. [61] in 2016. With the parametric hash function for random placement introduced by Kosmidis et al. [68] there is a finite probability that adjacent memory blocks will be mapped to the same cache set in some placements and thus conflict with each other. This degrades performance, particularly for an instruction cache with respect to loop constructs that span a number of adjacent memory blocks. The random modulo approach seeks to avoid this problem. It effectively creates a random permutation which maps from addresses in a memory segment (the same size as a cache way) to cache sets. This mapping retains the property of modulo placement, whereby memory blocks separated by less than the size of a cache way will not conflict in the cache. The hardware implementation also has a number of advantages over the parametric hash function method of Kosmidis et al. [68]; it uses less than 10% of the silicon area and can operate at a higher frequency. The evaluation of random modulo for instruction and data caches shows that it provides a significant reduction in the pWCET estimate at an appropriate exceedance probability, compared to the parametric hash function, with an advantage of 25-62% across the different EEMBC benchmarks studied.

Later in 2016, Trillia et al. [127] improved the resilience of caches implementing the random modulo random placement policy described by Hernandez et al. [61]. They note that the original form of this policy does not result in an even distribution in terms of how often the cache lines are used. This is due to the fact that the index bits that are used are entirely dependent on the access

pattern of the program. By the simple expedient of XORing part of the selected random seed with these index bits, homogeneity of cache line use can be achieved. This has the desirable effect of making the cache more reliable, since one of the main sources of transistor degradation (called Hot Carrier Injection) is proportional to the amount of use. The additional XOR gate has no effect on the maximum operating frequency of the cache, since it is not on the critical path.

In 2018, Benedicte et al. [11] considered the use of random replacement policies in multi-level caches. They show that performance, in terms of pWCET estimates at an appropriate probability of exceedance, can be improved by using different policies in the L1 and L2 caches. They explore the use of random modulo placement [61] in the L1 cache and the use of a parametric hash function [68] across L2 cache segments combined with either modulo or random modulo placement within L2 cache segments. These approaches provide an improvement in performance of approx. 30% compared to using parametric hash functions at both levels.

6.2 Caches and Software Random Placement

The majority of the work on software random placement for random replacement caches was also developed in a series of papers from 2013–2016 by Kosmidis and co-authors including Abella, Cazorla, and Quinones [72, 73, 71, 78].

In 2013, Kosmidis et al. [72] proposed the use of compiler techniques and runtime support to randomise the layout of both code and data in memory; effectively providing a software means of random placement applicable to hardware with direct mapped caches or set-associative caches using LRU replacement. The code and data layout in memory is changed before the start of each run of the program. Due to the deterministic mapping to cache, this has the effect of randomising the cache lines at which code and data objects start in the cache, thus randomising conflicts between objects, but not within them. Evaluation on examples from the EEMBC benchmark suite produce observations of execution times that pass the i.i.d. tests, thus allowing MBPTA to be applied. Software random placement does however have a number of drawbacks. The results show that the overheads increase the pWCET estimate at an appropriate exceedance probability by a factor of 10 for code that contains a loop that is repeated 100 times, and by a factor of 2 if the loop repeats 1000 times. We note that re-arranging code and data objects at runtime may be unacceptable in many industry sectors, such as automotive and aerospace. Such a re-arrangement means that deployed systems could run code and data layouts that have *never* been tested. This may reveal some subtle bug in functionality which was dormant in tested configurations, and could be very difficult to reproduce.

Subsequently, in 2014 Kosmidis et al. [73] acknowledged issues with their software random placement scheme [72]. This scheme conflicts with the design principles of the ISO26262 standard for the development of automotive software: *"(i) limited use of pointers, (ii) recommendations against the use of dynamic objects, and (iii) no hidden data flow or control flow"*. The authors therefore proposed an alternative: *Static Software Randomisation*. Here, the idea is to create a series of binaries at compile time that have locations for functions and data such that their mapping to cache follows a random selection. The method chooses random offsets for each function and data object from the start of the cache. It then finds a suitable ordering of the functions in memory, with some additional spacing to achieve the desired mapping to cache via modulo placement without wasting too much memory. The locations of stack frames and global variables are similarly randomised at compile time. The authors propose creating N binaries, running each one once, measuring its end-to-end execution time, and using this data as input into MBPTA. They assume that the estimated pWCET distribution so obtained will apply to all of the binaries. Then at deployment they propose either (i) a different binary is deployed in each production unit, or (ii) a single binary is chosen and deployed in all production units. They note that (i) may

not be acceptable if each individual unit is not fully tested. The authors claim that the pWCET distributions obtained from their scheme will be valid for case (ii), however, this is *not* correct. For the MBPTA method based on EVT to give sound results, it is necessary that the observations made in the analysis phase are either directly representative of those that could occur during deployment of the system, or they upper bound a set of values that are representative of those that could occur during deployment. Neither is the case when observations are taken from different binaries, which are effectively different systems to the one deployed. In this case, the observations are not identically distributed, but rather they come from the different distributions associated with each of the different binaries.

In 2016, Kosmidis et al. [71] investigated the use of static software randomisation applied to software from an automotive cruise control system running on a single core of an AURIX multi-core system. Dynamic software randomisation is not possible for this system, since the AURIX platform, in common with many used in automotive systems, does not permit self-modifying code. Instead, the authors use static software randomisation where program code, stack and global data are allocated random locations in memory across different binaries. A pool of different binaries are used during the analysis phase to obtain observations, with a single binary selected for deployment. The authors claim this allows EVT-based methods to be applied; however, as discussed above, this approach is not valid. In particular, the "overall system" used during analysis, which operates by selecting and then running a binary to obtain a single observation, is *not* the same as the system used during operation, which runs the same binary every time. Thus the execution time observations made at the analysis stage are not identically distributed with respect to those obtained during deployment, and hence the results of applying EVT-based methods are invalid in this case. Unfortunately, the authors misunderstand the point of applying tests on the observations to determine if they are independent and identically distributed (i.i.d.). They apply these tests on observations from 1000 different binaries used during the analysis phase and then claim that a positive result enables the use of EVT. This may be the case if the deployed system were the same as the one used during analysis (i.e. it switched binary every run), but it is not.

A different approach to software random placement called TASA *Tool-Chain Agnostic Static Software randomisation Approach* was proposed by Kosmidis et al. [78] in 2016. This approach randomises the location in memory at which different objects defined in the source code are placed. To do so, TASA relies on the fact that the compiler generates code and data in the order in which they appear in the source files. Thus it adds functionally neutral padding code and data and re-orders the declarations of variables and functions to achieve a degree of randomisation. Stack frames are also randomised by adding a randomly sized array to the list of local variables and also by re-ordering those variables. Similarly, the members of compound structures are also shuffled. TASA has the advantage over previous efforts at software randomisation in that it operates at the source code level and is thus portable, depending only on the programming language. The TASA approach relies of creating N binaries to use in the analysis stage (collecting end-to-end measurements for input into MBPTA) and then deploying a single binary for which it is assumed the pWCET distribution derived via EVT will apply. Unfortunately, as discussed above this logic is faulty. The N binaries represent different systems from that which is deployed, and hence the results from applying EVT are not valid for the deployed system.

6.3 Cache Risk Patterns with Random Placement

A significant issue with both hardware and software random placement is that some randomly chosen placements may result in a *cache risk pattern* [106] whereby a number of accesses within a loop conflict in the cache, thus resulting in a large increase in the execution time for that particular configuration. Further, the probability of such a pattern occurring can be below that which might

reasonably be observed in the limited number of execution time observations used in MBPTA, but above the probability threshold (e.g. 10^{-9}) at which such long execution times can safely be ignored. Thus random placement can create a needle-in-a-haystack problem, making the results provided by MBPTA unreliable.

Research aimed at addressing the above problems was developed predominantly by the same group of researchers who investigated software and hardware random replacement methods (see Sections 6.1 and 6.2, including Abella, Benedicte, Cazorla, Kosmidis, and later Milutinovic [4, 13, 14, 98, 99, 97].

In 2014, Abella et al. [4] identified the above issue with the random placement scheme of Kosmidis et al. [68] and suggested taking observations using a smaller cache as a way of revealing cache risk patterns by making them more likely to occur in the limited number of runs employed by MBPTA. Analysis is given which aims to compute the reduction in the size of the cache needed to achieve this. This analysis makes an assumption that a cache risk pattern for a given placement will always be observed in a run that uses that placement; however, unless the code is single path, or the loop is executed on every path, then this assumption may not hold. Different runs may exercise different paths, only a few of which may cause the loop with the cache risk pattern to execute. Hence the reduction in the size of the cache necessary to achieve sound results may be substantially greater than that computed using the formulation given in the paper. We note that the analysis provided by Abella et al. [4] assumes that each unique address is re-mapped independently by the random placement mechanism. While that may be the case using a hardware approach to random placement [68], with software random placement schemes [72] code and data are subject to random placement only at the level of functions and objects. It is not clear if the method proposed by Abella et al. [4] can be applied in that case; however, it is apparent that the issue of unobserved large variations in execution time also exists with software random placement schemes.

In 2016, Benedicte et al. [13] studied the problem of cache risk patterns with hardware random placement. In particular, they review the analysis given by Abella et al. [4] which determines the probability P_{eoi} of a particular event of interest (i.e. a cache risk pattern) occurring based on an approximation using *weak compositions theory*. The authors present a precise calculation of P_{eoi} using an approach based on multinomial coefficients. This calculation shows that the value of P_{eoi} computed by Abella et al. [4] using weak compositions theory may over-estimate the precise value. Such an over-estimate leads to an *under-estimate* of the number of runs required to have confidence that the event will occur during the runs used to collect execution time observations for input into EVT, thus undermining the soundness of the estimated pWCET distribution. Calculation of the precise value for P_{eoi} takes significantly longer, and may become intractable due to the need to enumerate all possible cache allocations of interest. The method is only valid for programs with *"homogeneously accessed objects"* in other words, programs where every object (e.g. instruction) is accessed the same number of times, i.e. in one single outer control loop with no conditional branches.

The issue of cache risk patterns due to software random placement schemes was also addressed by Benedicte et al. [14] in 2016. The authors note that unlike hardware random placement, software random placement works at the level of functions and objects and thus the probability that a particular object is allocated to a given cache set is dependent on the allocation of previous objects. They also note that determining a precise model of the probability P_{eoi} of a particular event of interest (a cache risk pattern) is intractable. Given a number of objects and their sizes, Monte-Carlo simulation is therefore used to estimate P_{eoi}, and hence whether it is greater than the required threshold e.g. 10^{-9}, but nevertheless too low to have confidence that the event will occur during the runs used to collect execution time observations for input into EVT. If so, the

number of runs R is computed such that the probability that the event will not be seen in R runs is less than $P_{conf} = 10^{-9}$. This is given by the formula $R \geq log(P_{conf})/log\ (1 - P_{eoi})$. The evaluation shows that when the number of objects is in the range $[5 - 60]$ (depending on cache and object sizes), then P_{eoi} can fall into the problematic range requiring an increase in the number of runs. Some examples are given showing that with $P_{eoi} = 0.085$ then the number of runs needs to be increased to approx 2500. However, we note that for smaller values of P_{eoi} e.g. 10^{-6} which also appears in the evaluation figures for a smaller number of objects, then the number of runs required such that the probability of not observing the event is less than 10^{-9} becomes very large e.g. approx. 2.10^7, which is unlikely to be attainable in practice. Aside from the potential for requiring very large numbers of runs, the main drawback of this approach is that, as the authors note, it only works for programs with homogeneously accessed objects. This severely restricts the use of the method when realistic programs, e.g. with functions within conditionals and nested loops, are considered.

In 2016, Milutinovic et al. [98] addressed an issue with the approach of Abella et al. [4] to identifying cache risk patterns caused by random placement that have a very low probability of being captured in the observations used in MBPTA. They showed that the approach of Abella et al.[4] *"only works when the impact on execution time of mapping any subset, bigger than W (the number of cache ways) is the same"*. This is only the case if all of the accesses are made in a round-robin fashion, for example single path code within a single large loop; however, this is not the case for programs in general which contain conditionals and other constructs. The method proposed by the authors aims to solve this problem. It considers all $\binom{U}{x}$ combinations of x out the U different memory block accesses in the program, where x is varied in the range $[W + 1, U]$. The probability of a given combination of x accesses mapping to the same cache set is computed and if this is found to be in the range of interest, then cache simulations are performed with the given combination of x accesses mapped to the same cache set and other accesses mapped at random. The simulation runs are used to determine an average miss count for the combination. This information is used to obtain a set of $(miss\_count, probability)$ pairs indicating the number of misses and their probability, derived from all of the combinations simulated. These points are plotted on a graph and compared with the probabilistic Worst-Case Miss Count (pWCMC) distribution obtained using MBPTA. If the pWCMC distribution does not upper bound all of the $(miss\_count, probability)$ points, then the number of observations used in MBPTA is increased until it does. The complexity of the method makes it intractable for practical programs with large numbers of memory block accesses due to the number of different combinations involved. For example 100 distinct memory block accesses and a 4-way cache would require cases where 5 address mapped to the same cache set are considered, of which there are $\binom{100}{5} \approx 7e10^7$ combinations. The authors attempt to deal with this problem by limiting U to the 15 most heavily accessed memory blocks. In the EEMBC benchmarks used for evaluation, this restriction means that approximately two thirds of the accesses are covered. No argument is given explaining why this is sufficient to ensure that the results obtained are sound.

In an extension [99] published in 2017, to their previous work [98] from 2016, Milutinovic et al. gave an example showing how the original [4] and improved [13] analysis of cache risk patterns, given by Abella et al. and Benedicte et al. respectively, fails to provide trustworthy results in cases when accesses are made in a non-homogeneous way. Evaluation using the EEMBC benchmark suite shows that for all of the benchmarks considered, these works under-estimate the number of observations required by MBPTA leading to results (i.e. pWCET distributions) that are untrustworthy below a probability of exceedance that is in the range 0.9 to 0.001 depending on the benchmark. (By contrast, sound results would be trustworthy down to at least a probability of exceedance of 10^{-9}). Further, they also evaluate issues of trustworthiness with the earlier

approach of Milutinovic et al. [98]. They show that issues can occur for simple cases where there are more addresses accessed in a loop than are considered by the analysis. Comparing the results for $U = 10$ to those for $U = 15$ on the EEBMC benchmarks shows that limiting the number of addresses considered can lead to either an over- or an under-estimation of the number of observations required by MBPTA, with under-estimation (occurring in 3 out of the 8 benchmarks) leading to untrustworthy results at the required probability of exceedance (10^{-9}).

Prior to the above works, many papers were published assuming random placement, but as far as we can tell they did not check that the number of observations made was sufficient to avoid the hazards of cache risk patterns described above. The validity of the evaluation results for systems using random placement in the following papers is therefore questionable: [68, 72, 69, 73, 67, 71, 119, 129, 128, 60, 61, 136].

Subsequently in 2017, Milutinovic et al. [97] recognised some of the problems with their previous work [98, 99] on identifying cache risk patterns caused by random placement that have a very low probability of being captured in the observations used in MBPTA. In particular, they note that *"evaluating in the cache simulator all potentially conflictive combinations of addresses is not feasible in the general case due to its exponential dependence on the number of addresses".* For $U = 15$ different addresses, exhaustive evaluation via cache simulation is shown to require 27 hours per benchmark on a compute cluster running 100 jobs in parallel. To address this problem the authors propose a Time-aware Address Conflict (TAC) approach which aims to identify a list of address combinations that if mapped to the same cache set can result in a high miss count. The basic method is the same as that described by Milutinovic et al. [98, 99], but rather than examining an exhaustive list of conflicting address combinations, only a limited number (i.e. 20) for each number K of conflicting addresses are considered. Values of K are examined from $W + 1$ upwards, where W is the number of cache ways, stopping when the probability of K addresses mapping to the same cache set is below the cutoff probability. The address combinations on the TAC list are ranked according to the concept of *guilt*, which aims to identify those combinations of addresses which are likely to result in high cache miss counts. Guilt is a heuristic estimate formulated via an expression which reflects the re-use distance of each address access. This is used to populate an *address guilt matrix* which aims to capture the extent to which misses on accesses to some address A are caused by accesses to some other address B. The TAC approach uses the information in the address guilt matrix to determine a ranking for different combinations of address accesses. A small number of the top ranked combinations are then evaluated via cache simulation. This reduction in the number of combinations considered improves the runtime by orders of magnitude compared to the exhaustive approach. The approach is shown to be effective, giving the same results as the exhaustive approach when the number of different addresses considered is limited to 15. Further evaluation on a railway case study considers 10 different test cases defined by specific input vectors, with the address traces used as input into the method. It appears that the approach is limited in its applicability to traces of addresses and thus to single paths through a program. MBPTA can be applied on a per-path basis giving a result that is valid for a single path, and can also be applied on a per-program basis. It is not clear how, or even if TAC would work on a per-program basis. We note that the *guilt* metric is a heuristic, and there is no proof given that it is guaranteed to always ensure that the address combinations with the highest likelihood of resulting in cache misses will be examined. Thus the improvement in runtime efficiency appears to come at a cost in terms of reduced confidence that the resulting pWCET distribution is valid.

In 2018, Milutinovic et al. [100] considered the problem of cache risk patterns due to random placement policies when applying the Path Upper Bounding (PUB) technique of Kosmidis et al. [70] (see Section 5.1). Recall that PUB pads the code with extra accesses in such a way that the pWCET distribution for any path through the modified code upper bounds the pWCET

distributions for all paths through the original code. Thus only one arbitrary path through the modified code needs to be exercised to obtain a sound pWCET estimate. The authors apply the TAC approach [97] (discussed above) to the modified code produced by PUB. They show that applying PUB can make cache risk patterns due to random placement either more or less likely to occur, and may in some cases significantly increase the number of observations required by MBPTA to obtain sound estimates of the pWCET distribution (e.g. from 3000 to 500,000). They give examples where cache risk patterns lead to an under-estimate of the number of observations required when using PUB alone, resulting in an estimated pWCET distribution which is unsound compared to the empirical distribution obtained by taking a large number of observations. The use of TAC is intended to address this problem.

6.4 Buffers, Buses and other Resources

As well as random replacement caches with/without random placement, other resources have been investigated and adapted to provide randomised behaviour, with the aim of making systems more amenable to being analysed using MBPTA techniques. The majority of the work in this area was published by a group of researchers including Slijepcevic, Cazorla, Kosmidis, Jalle, Abella, Hernandez, and Quinones from 2013 – 2016 [119, 27, 74, 65, 25, 60].

In 2013, Slijepcevic et al. [119] proposed a *Degraded Test Mode* (DTM) which when combined with fault tolerant random replacement caches enables MBPTA to be used to obtain pWCET distributions which estimate the execution time behaviour of the system in the presence of a given number of hardware faults that cause some cache lines to become unavailable. The motivation for this work is that although test methods can verify that processors do not contain faults when first deployed, degradation can cause latent defects to manifest into permanent faults. Hardware mechanisms can address such failures to some extent, e.g. by disabling cache lines when a faulty bit is detected. The rate at which this occurs depends on the technology scale used. With random placement and a set-associative random replacement cache or a fully-associative random replacement cache, the authors argue that there is little dependence on the actual cache lines which are faulty. Thus the degraded test mode configures the cache to have a number of lines disabled, commensurate with the fault probability per bit over the required lifetime of the system. MBPTA can then be applied to the system in degraded mode. Due to a lack of dependence on the location of any failed lines, the resulting pWCET distribution is valid for any set of bit failures in the cache lines that could be expected during the lifetime of the system.

Also in 2013, Cazorla et al. [27] looked at the requirements placed on the *analysis phase* runs of a program on a hardware platform, with the aim of ensuring that the observations of execution times are representative of or upper bound those for any population of such values that might be obtained during operation. As discussed in Section 4 *representativity* is necessary for the estimated pWCET distribution derived via MBPTA to be valid for future scenarios of operation. The authors reinforce the point made by Cucu-Grosjean et al. [34] in 2012 that adequate path coverage is required to ensure that observations from the worst-case path are included. The authors classify a number of different sources of execution time variation that need to be controlled for. For example, division operations may take a variable time dependent on the operands. Such variation could potentially be controlled for by using worst-case values, or in hardware via a worst-case mode, where such operations always take their worst-case time to execute. Code and data placement in memory typically affect the mapping to cache, which impacts execution times. Here, the authors suggest using random placement techniques to ensure representativity; however, as discussed in Section 6.3, random replacement also has significant problems relating to representativity due to cache risk patterns.

In 2014, Kosmidis et al. [74] aimed to set out the properties or architectural features that a

processor needs to have to *guarantee* that the MBPTA method [34] can be applied. They classify hardware resources into either *jitterless*, having no execution time variability, or *jittery* resources. Jittery resources may result in latencies that depend on the execution history or the input values or a combination of the two. They propose that jittery resources should either be assumed to always take their worst-case latency or be time-randomised. In the evaluation, the authors apply MBPTA to programs running on a time-randomised architecture with both instruction and data caches using random placement and random replacement. They use the statistical tests described by Cucu-Grosjean et al. [34] to check if the execution time observations are i.i.d., and this was shown to be the case for the EEMBC benchmarks used. The authors claim that *"Both tests are passed in all cases, which proves that the example architecture meets the i.i.d. requirement."*. This claim of *proof* is in our view extended too far, rather the experiments show that the observations are i.i.d., but *only* for the particular instances of those benchmarks on the given architecture. There is no evidence that this would necessarily be the case for any program on the given architecture. Later work by Lima et al. [88] (discussed in Section 4.3) shows that using a random replacement cache is not sufficient to guarantee that observations are i.i.d., neither does an LRU cache necessarily preclude it. We note that in general any hardware resources or software variables that preserve state between runs of a program could potentially lead to dependences between execution time observations, breaking the independence property.

A *random permutation* bus was proposed by Jalle et al. [65] in 2014 as a means of connecting cores and memory in a multi-core system. With a random permutation bus, the bus arbiter produces a new random permutation (order for contenders to access the bus) every N rounds, where N is the number of contenders. The random permutation bus is compared to a lottery bus (introduced by Lahiri et al. [79] in 2001) that makes a random selection of which contender gains access to the bus on each round, and a conventional Round-Robin bus. Applying MBPTA, the authors show that the pWCET estimate at an appropriate exceedance probability is reduced by between 1.5% and 9.6% for a random permutation bus as compared to a Round-Robin bus. In the latter case, an assumption is made that every access incurs the worst-case delay.

In 2015, Panic et al. [102] showed how systems that incorporate buses and other resources with TDMA arbitration may be analysed using MBPTA. The key idea is that the largest difference between execution times that can be caused by a misalignment of any number of synchronous (blocking) requests with the TDMA cycle is $w - 1$ where w is the length of the overall cycle as shown by Kelter et al. [66]. The authors extend this result to multiple TDMA resources showing that it generalises to $LCM(w_1, w_2, \ldots) - 1$. Given these results, the simple expedient of analysing the system using MBPTA and then padding the results by adding the largest difference that could be caused by misalignment with the TDMA cycle, results in an upper bound. Since the padding is typically small compared to overall execution times, this method is effective and provides superior performance to the random permutation bus [65] or forcing all accesses to take the worst-case time. An extended version of this work was published in 2017 by Panic et al. [103], adding a discussion of the impact of timing anomalies.

The behaviour of random replacement caches, random permutation bus arbitration, and other hardware components with time-randomised behaviour depends on the quality of the underlying random number generator. In 2015, Agirre et al. [5] described a Pseudo-Random Number Generator (PRNG) designed to provide the random numbers needed to implement these hardware components. The proposed PRNG uses a Linear-Feedback-Shift-Register (LFSR) design, which provides a long period ($> 2^{60}$) for the random number sequence. The LFSR design is modified to produce a 32-bit pseudo random number on each cycle, groups of bits from which are then used as the random numbers required for different components (e.g. instruction and data cache etc.). The authors discuss a number of ways in which the basic design can be hardened to meet

the safety requirements of IEC-61508 SIL 3. These include duplicating the PRNG and checking via a voter that the outputs match, and also using a watchdog timer to check that new values have been produced. They report that the PRNG passes 187 out of the 188 tests specified by the US National Institute of Standards and Technology for assessing randomness properties. Failing only the Linear Complexity test which is used to determine if the sequence produced is complex enough to be considered random [109]. The evaluation shows that the observed execution times for the EEMBC matrix benchmark failed the i.i.d. tests used as part of the MBPTA method when obtained from a LEON 3 processor prototyped on an FPGA using the default random policy, but passed those tests when the LFSR PRNG implementation was used.

In 2015, Hernandez et al. [60] took a preliminary look at the changes potentially needed to the LEON3 multi-core processor in order to support the use of MBPTA. They identify a number of points: The existing random replacement policy, which can be configured for the caches, uses a randomisation method that is not of sufficient quality. It is replaced by the PRNG described by Agirre et al. [5]. Further, hardware enabled random placement is implemented using the method based on a parametric hash function described by Kosmidis et al. [67]. The core-to-L2 and L2 cache are modified so that requests from one core cannot cause stalls for another core. Finally, the bus and memory controller arbitration policies are round-robin and FIFO respectively. The authors suggest that these need to be modified to be either random permutation [65] or lottery bus [79]; however, this change is not made, instead benchmarks are run on only one core thus avoiding issues of contention. The brief evaluation shows that the 1000 execution time observations used for each of the two programs vary by less than 0.001% for rspeed, and by 0.04% for the matrix benchmark. This tiny variation is due to the fact that the benchmarks fit comfortably in the cache. The authors draw no conclusions, but suggest that they will in future study further benchmarks with different cache requirements.

In 2016, Cazorla et al. [25] and Kosmidis et al. [76] summarised the research and development work on support mechanisms for the MBPTA method [34] developed during the EU PROXIMA project and described in previous papers.

Also in 2016, Slijepcevic et al. [117] proposed a tree-based Network-on-Chip (NoC) for a many-core system, adapted with the aim of permitting MBPTA of tasks running on the cores. The tree-based NoC uses either Round-Robin (with a worst-case mode enabled for analysis), Lottery, or Random Permutation methods for arbitration at each level in the tree. The evaluation shows that the tree-based NoC provides higher performance than a bus for clusters of 8 or 16 cores. Slijepcevic et al. followed this work with a further paper [118] in 2017 on the use of Random Permutation methods in the routers of a wormhole NoC. The aim being to avoid the systematic worst-case behaviour which has to be accounted for in the analysis of worst-case traversal times when deterministic arbitration policies are used. Applying MBPTA, and using a probability of exceedance of 10^{-13}, the authors show that the pWCET estimates improve on the WCETs for an equivalent NoC with deterministic routing by on average 22% to 75% for 3x3 and 6x6 NoCs respectively. (This assumes that the number of in-flight requests are limited, a technique that improves the analysed performance in the randomised case, but not in the deterministic case).

An alternative approach to implementing random replacement caches was proposed by Benedicte et al. [12] in 2018. Recall that on a cache miss, the conventional random replacement policy selects at random any one of the cache lines in the cache set for replacement. With W ways this means that on a cache miss the probability of eviction is $1/W$ for each cache line in the set. Although this form of randomisation is effective in supporting MBPTA, it is also inefficient. For example, alternate accesses to two addresses that map to the same cache set may cause mutual evictions even in the case where there are 4-ways available. The authors propose a form of random permutation to avoid this problem and thus improve performance. With this Random

Permutations Replacement (RPR) method a random permutation is generated for each cache set. This permutation determines the order in which all of the ways in the set will be subject to eviction. Once all of the ways have been evicted then another random permutation is chosen and so on. This has the advantage that any repeating sequence of $K < W$ distinct address accesses can only result in a maximum of $K - 1$ evictions. (This worst-case can happen across the boundary of two permutations). The authors describe an efficient hardware implementation of the RPR mechanism which trades off the number of distinct permutations used against the number of bits required for implementation. The evaluation shows that the approach results in pWCET estimates at an appropriate probability of exceedance that are on average 24% better than those for conventional random replacement for the Mälardalen benchmarks studied, and 16% better on average for a railway case study.

6.5 Summary and Perspectives

The concept of an *"MBPTA-compliant"* platform has been pursued in many publications with the intent that any program running on the platform will be amenable to analysis using MBPTA methods. While a commendable goal, recent research indicates that there is no such panacea. Time-randomised architectures are neither necessary nor sufficient for the application of MBPTA methods based on EVT. Rather the combination of input values and hardware states, the program, and the hardware platform all influence whether MBPTA methods can be applied. Appropriate statistical tests are needed on the sample of execution time observations to determine if the method can be applied, with goodness-of-fit tests used to determine if the estimated pWCET distribution is a close match to the empirical distribution of the maxima or peaks over threshold. While time-randomised architectures may make it more likely that MBPTA methods can be applied, they cannot guarantee it. Neither do time-predictable (deterministic) architectures preclude the use of MBPTA methods [112, 16, 53, 54, 88, 87]. Time-randomised architectures may help in avoiding hazards due to pathological cases where large variations in execution time can occur very rarely (i.e. below the level at which they can reasonable be observed in testing) without being made up of a combination of smaller variations that can be observed (see the hypothetical example at the start of Section 6). However, time-randomisation does not always achieve this and can sometimes make the situation worse, as has been shown in the case of random placement.

The work on random placement policies (reviewed in Sections 6.1 and 6.2) has a number of issues. Randomising the mapping of memory blocks to cache sets across different runs of a program has been shown to lead to cache risk patterns which may not be detected during testing and analysis, but which can seriously impact execution times at much higher probabilities than are acceptable in terms of the resulting timing overruns. Despite work by Abella et al. [4], Benedicte et al. [13, 14], and Milutinovic et al. [98, 99, 97] there are as yet no viable practical solutions to this problem that can guarantee to produce trustworthy results for realistic programs. Static Software Randomisation [73, 71, 78] where a single randomly chosen placement is used in the deployed system with various different random placements used in testing and timing verification, appears to misinterpret the requirements of MBPTA methods that build upon EVT. For these methods to give sound results, it is necessary that the observations made in the analysis phase are either directly representative of those that could occur during operation, or upper bound them as noted by Cazorla et al. [27]. This is not the case when observations are taken from different binaries, which effectively constitute different systems. Such observations are not identically distributed with respect to those from the system during operation, and hence it is invalid to use the estimated pWCET distribution obtained via EVT in this case. Finally, while hardware random placement requires custom hardware [9], dynamic software random placement [72] goes counter to engineering practice, conflicting with the design principles set out in standards such

as ISO26262. Such re-arrangement of code and data objects at runtime means that deployed systems could run code and data layouts that have *never* been tested. This is unacceptable in many industries.

7 Case Studies, Benchmarks and Evaluation

Many papers surveyed in the previous sections include some form of evaluation. In this section, we review work that specifically focuses on performance evaluation, through the use of case studies, benchmarks, or evaluation frameworks. In addition, we review papers that provide a critique of SPTA and MBPTA methods and the open challenges that remain.

7.1 Critiques

In 2014, Reineke [108] made a critical technical comparison between deterministic analysis of LRU caches and probabilistic analysis of random replacement caches. He derives a number of logical conclusions, considering first random replacement and then random placement versus deterministic alternatives. Conclusion 1: LRU is preferable to random replacement in that it *always* has better guaranteed performance compared to the simple re-use distance formula for SPTA given by Davis et al. [39]. (We note that this is no longer the case with more effective SPTA techniques [7, 6]). Regarding MBPTA, Reineke [108] notes that the measurement protocol introduced by Cucu-Grosjean et al. [34] flushes the cache on program start and also prevents all input-dependent memory accesses from accessing the cache. Thus for a given path the sequence of accesses going to the cache is the same on every run, independent of the input data. Under that assumption, the behaviour of an LRU cache is fixed for a given path and so the program has only one execution time for each path. Conclusion 2 follows: MBPTA can be more efficiently employed on top of an LRU cache. With random placement, Reineke [108] shows that no independent non-zero probabilities of a cache hit can be assigned to accesses with a stack distance greater than zero. Conclusion 3 follows: random placement would require complex conditional static probabilistic analysis and is not amenable to current analysis techniques. Regarding MBPTA, Reineke [108] shows that with random placement, MBPTA cannot necessarily detect that there may be rare layouts (e.g. with a probability of 10^{-6}) that result in a large number of cache misses. Such layouts may not be observed in the runs used in the analysis phase of MBPTA. Conclusion 4 follows: random placement is not suitable for use with MBPTA. (We note that the same conclusion has also been reached by Maxim et al. [92] by considering the reproducibility property of a measurement protocol required to ensure the convergence of any set of measurements towards a representative, and thus sufficient, number of execution time observations for EVT-based estimation of the pWCET distribution).

The technical critique given by Reineke [108] was discussed by Mezzetti et al. [96] later in 2015. Regarding the use of SPTA techniques with random replacement caches (Conclusion 1), they point to the more effective SPTA methods developed by Altmeyer and Davis [7], published after Reineke's paper, which show that the guaranteed performance of random replacement caches is incomparable with that for LRU caches. With respect to SPTA and random placement (Conclusion 3), they agree that while it is true that current SPTA approaches cannot be used with random placement, future advances may be possible along this line. To the best of our knowledge, as yet no such advances have been made. Regarding the use of MBPTA with random replacement caches (Conclusion 2), the authors note that random replacement has better performance than LRU when the stack distance of accesses exceeds the number of cache ways, since in that case LRU treats all accesses as misses. With respect to MBPTA and random placement (Conclusion 4) they note the work on identifying the potential for cache risk patterns [4] as a possible means of addressing this issue.

In 2015, Stephenson et al. [123] outlined a certification argument structure aimed at providing an appropriate argument for the use of MBPTA methods [34] in an industrial context. This makes use of Goal Structuring Notation (GSN) to provide a modular structure showing the relationships between different parts of the required argument that have different concerns. The required argument is broken down into a number of elements relating to execution time measurement, soundness of the MBPTA method, soundness of the timing data extraction, and uncertainty mitigation. Further details of what is required for the "sound method" argument are then discussed. These include the need for testing to adequately sample the path or paths that represent the worst-case, and a number of factors related to the statistical methods used. Such factors include the use of appropriate parameter values in hypothesis testing, e.g. in the i.i.d. test, and testing the hypothesis that the tail of the distribution matches a Gumbel distribution. The authors point out that a lack of confidence in the values of the parameters used could lead to a lack of confidence in the overall method. Similarly, confidence is needed that the size of the sample of observations used is sufficient. They also note that *"The application of the MBPTA method requires providing evidence that the hypothesis on which it stands hold in the context of use."* and that currently the parameters used are based on general practice for statistical approaches.

Subsequently in 2017, Gil et al. [49] discussed the open challenges in MBPTA. They identified three main areas:

1. How to ensure that a representative set of observations is obtained? This involves determining the requirements for representativity, generating appropriate test vectors that will result in representative observations, checking that coverage of the program and hardware states are sufficient for representativity, and determining how many observations are needed.
2. How to ensure a trustable application of EVT? This involves demonstrating that the methods used to obtain EVT configuration parameters are reliable, and that the application of those methods is reproducible.
3. How to interpret the results of EVT? This involves understanding the uncertainties in the overall measurement and analysis process, and determining an appropriate exceedance probability to use.

7.2 Case Studies and Evaluation

In 2011, Santos et al. [113] investigated the composition of statistical models of execution time for components, and how this is affected by different architectural features. They considered pairs of components $c1$ and $c2$, and examined whether the simple convolution of the execution time distributions obtained for the components in isolation (which is valid assuming independence) gives a good approximation of the joint distribution obtained when $c2$ is executed immediately following $c1$. They used the Kolmogorov-Smirnov goodness-of-fit test to test the hypothesis that the two distributions are the same. The evaluation used code from the MiBench suite, with the SimpleScalar tool chain used to simulate modern processors with features such as out-of-order execution. Out of 100 compositions (pairs and triples) only 3 resulted in rejection of the null hypothesis. (The null hypothesis is that the distribution formed from convolution of the distributions for single components obtained independently is the same as the distribution obtained by running the components consecutively). They also investigated which of 37 different hardware configuration parameters had the strongest influence on the difference between the distribution obtained via convolution and that directly measured. The re-order buffer had the most influence overall; however, for the cases where the null hypothesis was rejected the branch predictor had the most influence.

Time composability was also investigated by Kosmidis et al. [75] in 2013 in relation to software running on a system with fully-associative random replacement caches. They showed that the cache interference due to some other software component B running between invocations of a

particular component A, can be characterised in terms of the maximum number of cache evictions that B can cause. This is limited to at most the number of distinct addresses u that the code in B accesses. They propose using a micro-benchmark to characterise the maximum impact that any such code B with u distinct accesses can have on the pWCET distribution of A. Equation (2) in the paper aims to compute the number e of evictions required to evict *at least* u distinct entries from the cache. (We note that with random replacement no such guarantee can be made; instead, the formula approximates the number of evictions required such that the *expected* number of distinct entries evicted is u). This formula is used to determine how many evictions a micro-benchmark should make to upper bound the impact that the interfering code in B can have on the subsequent execution time of the code in A by evicting at most u useful lines from the cache. The improvements found in the pWCET estimates at an appropriate probability of exceedance compared to flushing the cache were 5-10% for a 4Kbyte cache and 10-25% for a 32Kbyte cache for programs from the Mälardalen benchmark suite.

Later in 2013, Kosmidis et al. [77] applied the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) to systems that include *buffer resources*. They showed that resources with deterministic behaviour such as FIFO buffers do not create different probabilities of execution time outcomes for any given sequence of events, but rather propagate execution time variability or jitter. (We note that this result is somewhat unsurprising, since such buffers have deterministic behaviour any single fixed pattern of inputs yields a single fixed pattern of outputs).

In 2014, Abella et al. [2] compared deterministic and probabilistic methods of estimating the WCET / pWCET distribution using code from the Maladarlen benchmark suite [56]. This work compares classical static deterministic timing analysis using the Heptane tool for an LRU cache to SPTA and MBPTA for a random replacement cache. The comparisons investigate the sensitivity of the different approaches to cache line size and associativity. The SPTA method used is the initial approach derived by Davis et al. [39] (see Section 3.3) that uses only reuse distances, and provides a simple multi-path analysis. The MBPTA method used is the one developed by Cucu-Grosjean et al. [34] (see Section 4.1). The authors only consider single-path benchmarks. They found that in this case the results for MBPTA were less pessimistic compared to simulation than those from SPTA. We note that it would be interesting to see these comparisons repeated using the more sophisticated SPTA methods subsequently developed by Altmeyer et al. [7, 6], as well as for multi-path programs using the SPTA derived by Lesage et al. [83, 82] (see Section 3.3).

In 2013, Wartel et al. [129] applied the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) to an Integrated Modular Avionics case study. The software for the case study comprises five functions from an application that performs data concentration and maintenance of the flight control computers. This software was run on a simulator composed of a PowerPC MPC755 instruction set and pipeline emulator combined with a time accurate cache simulation. The memory hierarchy comprised separate 32KB, 8-way set-associative write-through L1 caches, and a 64KB, 8-way set-associative unified L2 copy-back cache. The caches used both random replacement and random placement. MBPTA was successfully applied, with only a few hours needed to extract the measurements and analyse the resulting observations to produce estimated pWCET distributions. The pWCET estimates at appropriate exceedance probabilities resulted in values between 0.1% and 3.8% greater than the highest observed execution time for an equivalent configuration with caches using LRU replacement and modulo placement. We note that it is not clear if these figures include the overheads of random placement, which were previously shown to be considerably higher by Kosmidis et al. [68] (see Section 6.1).

In a further avionics case study in 2015, Wartel et al. [128] applied the MBPTA method of Cucu-Grosjean et al. [34] to two different programs. The first performs data concentration and maintenance of the flight control computers, while the second computes an estimation of the centre

of gravity of the aircraft. The programs were run on a cycle accurate timing simulator which models the MPC755 architecture. The case study investigates two different hardware configurations. The first is a single core that uses software random placement of functions and stack frames with LRU caches. The second is a multi-core that uses hardware randomisation (random replacement caches, random placement, and a random permutation bus arbiter). In both cases the observations of execution times pass the i.i.d. tests enabling the MBPTA method to be applied. The results show that software randomisation increases the pWCET estimate at an appropriate exceedance probability by 12% for the first application compared to the maximum observed value for an equivalent system with modulo placement. In the multi-core case, the increases were 15% and 28% for the two applications. The authors note that the observed execution times in the multi-core case were relatively independent of the load running on the other processors, and that this was not the case with an equivalent system using a Round-Robin bus arbiter where the execution times varied by more than a factor of 3. We note that this is a consequence of the fact that Round-Robin is work-conserving whereas random permutation is effectively a variation on TDMA, which re-orders the slots allocated to each core on each cycle. A more interesting comparison would have been with traditional TDMA.

In 2015, Lesage et al. [85] introduced a framework that can be used to evaluate the precision of MBPTA. This is difficult to do with complex programs and hardware due to the difficulty in obtaining a ground truth in terms of the precise pWCET distribution. Instead of providing real measurements as input to the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1), the proposed framework instead provides realistic data from synthetic tasks. Restrictions on the abstract model used enable the precise pWCET distribution to be computed and used as a reference. The technique operates by first creating an Abstract Syntax Tree (AST) representing the program. Execution Time Profiles (ETPs) are then obtained for basic blocks, via measurements of the real program. These are attached to the AST, which is used to represent a synthetic task. To evaluate the MBPTA method, the synthetic task is "executed", i.e. a random path is chosen through the task and values from the ETPs for the basic blocks visited by that path are chosen at random. The overall synthetic execution time is then passed to MBPTA as an observation. With this model, the precise pWCET distribution can be computed from the AST via a tree-based analysis. For the simple synthetic tasks considered, the path randomisation used is sufficient to ensure path coverage. With full path coverage, the MBPTA method provides a tight and sound bound on the execution time obtained from the computed (precise) pWCET distribution at an exceedance probability of 10^{-9}. Further evaluation was performed removing nodes (and hence complete paths) from the AST, these experiments show that a lack of path coverage quickly degrades the soundness of the results with optimistic execution time predictions appearing and becoming more prevalent as the number of omitted nodes is increased. This illustrates the critical importance of achieving path coverage when analysing real systems. Some aspects of the approach reported are favourable to current MBPTA techniques, for example the random choice of values from the ETPs, and the use of completely independent ETPs for basic blocks avoids path and state dependences which may be present with real programs running on a real system. The authors suggest that the framework could be adapted to model different forms of dependences in future, for example dependences between successive runs, and dependences between the ETPs for successive blocks.

In 2017, Mezzetti et al. [94] applied the EPC method [136] (see Section 5.1) to a railway application; a simplified European Train Control System. They describe how EPC was integrated into an industrial tool chain (Rapita RVS[16]). Recall, that EPC collects observations at the basic

[16] See https://www.rapitasystems.com/

block level. It then uses probabilistic padding computed using SPTA techniques derived by Altmeyer and Davis [7] to increase the executions times observed to account for any advantage that may have accrued due to the path taken to that block. Observations can then be synthesised for all paths. The authors note that although EPC does not require that all paths are exercised when obtaining the execution times for basic blocks, as shown by Lesage et al. [85], all paths *do* need to be covered by the synthesised observations. In practice the number of paths may be very large, the authors therefore make use of semantic information in the form of flow facts to reduce the number of paths considered as feasible, accounting for maximum loop iterations and correlations between conditionals. This is highly effective in the case of the railway application studied, reducing the number of paths considered from 12996 to just 26. The hardware platform used is an FPGA prototype implementing a modified LEON3/4 architecture, with random modulo placement [61] and random replacement caches. We note that there is no mention of whether the issue of cache risk patterns previously identified with random placement was addressed (see Section 6.3). The results of a simple control experiment indicate that the EPC method leads to approx. 10-20% over-approximation of the pWCET at an appropriate exceedance probability, compared to exhaustively exercising all paths and using MBPTA. With the railway case study, unfortunately the worst-case path could not be exercised during testing; however, from analysis of a similar path that was executed, the authors estimate the over-approximation at around 30%. They note that only 23% of basic blocks needed padding, and that padding increases the original observations by on average 10%, and up to 400% in the worst-case.

In 2017, Fernandez et al. [48] reported on a case study using software random placement [72] (see Section 6.2) in support of MBPTA. These techniques were applied to an application that controls the active optics of a space telescope. This application runs on a LEON3 processor with separate L1 data and instruction caches and a unified write-back L2 cache. The evaluation shows that the maximum observed execution time is changed little by the overheads of dynamic software randomisation. Further, the pWCET estimate at a probability of exceedance of 10^{-15} is only approx. 2% larger than the maximum observed execution time for the original version, and hence substantially less than the 20% engineering margin that the authors claim is typically applied in this system. We note that in this work there is no mention of any mitigation of the issues of cache risk patterns (see Section 6.3) that can occur with software random placement.

In a 2-page paper in 2017, Cros et al. [33] reported on a case study applying MBPTA to a space application (Thrust Vector Control). They used a LEON3 processor with random modulo placement [61] and an FPU modified to have fixed latency operations in analysis mode. The results show that the pWCET at a probability of exceedance of 10^{-6} is 1.5 times larger than the maximum observed execution time for the equivalent deterministic system. This equates to the 50% engineering margin that the authors claim is usually applied in this system. The values for an exceedance probability of 10^{-9}, 10^{-12}, and 10^{-15} were approx. 1.75, 2.0, and 2.2 times larger respectively.

The use of the MBPTA method developed by Cucu-Grosjean et al. [34] (see Section 4.1) was extended to multi-core hardware (a 4-core LEON 3 platform implemented on an FPGA) by Diaz et al. [43] in 2017. The key challenge here is to address the additional contention delay on memory requests due to tasks running on the other cores. With the proposed method, the task under analysis is run on a single core in isolation, i.e. with no contenders, and execution time observations recorded. To account for possible contention, Performance Monitoring Counters (PMCs) are used to record the number of requests of different types. Two approaches are then used to add on appropriate upper bounds on the additional delays that could be caused by contention. A fully time composable (fTC) approach includes the maximum possible delay that any contending tasks on other cores could cause. A partially time composable (pTC) approach matches up

each memory request of the task under analysis with the worst-case additional delays that could be caused by specific contending tasks, taking into account the maximum number of delays of different types that they may cause. pTC provides a tighter bound than fTC, but requires specific information about the contenders. The experimental evaluation shows that the results from MBPTA for tasks running in isolation are a few percent above the maximum observed value, thus the remaining evaluation which factors in contention reveals mainly the effectiveness of the fTC and pTC approaches. The pTC approach provides good results with a bound of less than 1.5 times the observed multi-core value for the benchmarks considered. The fTC approach is much more pessimistic with bounds 12 times larger.

Also in 2017, Silva et al. [116] evaluated the reliability and tightness of the estimated pWCET distributions derived by MBPTA via fitting (i) GEV and (ii) Gumbel distributions using the Block Maxima approach. They used the L-moments method to fit a GEV distribution, and the Maximum Likelihood Estimator to fit a Gumbel distribution. The evaluation assesses *reliability* in terms of whether the pWCET estimate at a probability of exceedance of 10^{-15} and its confidence intervals are above the High Water Mark (HWM), i.e. the maximum, obtained from 10^8 observations used for validation. The *tightness* is assessed by comparing the pWCET estimate at a probability of exceedance of 10^{-7} with the maximum values observed from 10^6 and 10^8 samples. Experiments were performed on the *bsort*, *insertsort*, and *bs* sorting algorithms from the Mälardalen benchmark suite using input data that selected the worst-case path (effectively single path examples). Variability in run times was therefore only due to hardware randomisation in the execution platform. This was an FPGA implementation of a dual-core processor with a randomised bus and a random replacement cache. The experiments show that using a GEV distribution, there is significant variability in the pWCET estimates at a probability of exceedance of 10^{-15} for different analysis sample sizes (from 3 to 100 blocks of size 50). Further, since the HWM was often within the confidence interval, and so could be above the pWCET estimate, the results were not reliable. Fitting to a Gumbel distribution, however, provided reliable results which were also tight, i.e. only a few percent higher than the HWM. Further experiments were also performed using synthetic input data from GEV distributions with specific shape parameters in the range $-1/2$ to $+1/2$. In all cases, fitting to a GEV proved unreliable. For shape parameters that were negative or zero, using a Gumbel distribution was reliable and provided reasonably tight results. For positive shape parameters, fitting to a Gumbel distribution is not appropriate and indeed it produced results that were optimistic. The evaluation also showed that the Continuous Ranked Probability Score (CRPS) metric used in the method derived by Cucu-Grosjean et al. [34] to determine when sufficient samples have been obtained for analysis resulted in reliable results, but could be improved upon. Improvements could be achieved either by using a smaller threshold, or via assessing convergence using plots of the pWCET estimate and confidence intervals versus sample size, as shown in the paper. We note that fitting a distribution minimises the absolute error rather than attempting to minimise the over-approximation while avoiding any under-approximation. Fitting a Gumbel distribution to data from a distribution with a finite maximum (which would be better represented by a reversed Weibull distribution) typically results in an over-approximation. The authors consider this over-approximation to be indicative of the reliability of the method when using a Gumbel distribution; however, we would caution against drawing such a conclusion. In these cases, the distribution is an over-approximation because there is a mismatch with the shape parameter. This results in a fitted distribution which is pessimistic at small probabilities of exceedance. This pessimism is not necessarily an indicator that the method is reliable per se (i.e. always produces sound results). Indeed, for cases where the shape parameter is positive, using a Gumbell distribution produces results which are optimistic.

In 2018, Reghenzani et al. [107] described chronovise, an open source software framework for MBPTA. chronovise is a static C++ library that supports the Block Maxima, Peak-over-Threshold and MBPTA-CV [3] approaches.

7.3 Summary and Perspectives

Evaluation of the MBPTA method introduced by Cucu-Grosjean et al. [34] on avionics case studies at Airbus [129, 128] has shown both the applicability of the method to real systems, and also that the use of random replacement caches comes at a relatively small performance penalty compared to using deterministic replacement policies such as LRU. Recent work (discussed in section 4.2) shows that MBPTA methods can also be successfully applied to systems running on time-predictable architectures. For example Berezovskyi et al. [16, 15] successfully apply MBPTA methods to programs running on an NVIDIA Kepler GK104 GPU, and Guet et al. [53, 54] to benchmarks running on an Intel Xeon with 4 cores and 3 levels of cache. Initial work by Diaz et al. [43] in 2017, shows how the MBPTA approach can be extended to a multi-core system.

The important issue of proving the validity and correctness of the results from MBPTA has been investigated by Lesage et al. [85]. They showed that a lack of path coverage quickly degrades the soundness of the results, with optimistic execution time predictions appearing and becoming more prevalent as the number of omitted paths increases. Mezzetti et al. [94] addressed the path coverage problem using the EPC method [136] and demonstrated its effectiveness on a railway application.

Silva et al. [116] evaluated the reliability and tightness of the pWCET estimates derived using MBPTA by fitting GEV and Gumbel distributions, concluding that using GEV is unreliable (with large confidence intervals), while Gumbel can provide results that are reliable and tight in those cases where it is applicable. To show that it is applicable; however, requires checking against the maxima of a very large number of observations obtained in a validation phase. It is important to note here that fitting a distribution minimises the absolute error rather than attempting to minimise the over-approximation while avoiding any under-approximation. Fitting a Gumbel distribution to data from a distribution with a finite maximum, which would be better represented by a reversed Weibull distribution, typically results in a fitted distribution which is pessimistic at small probabilities of exceedance. This pessimism is not necessarily an indicator that the method is reliable per se (i.e. always produces sound results). Indeed, for cases where the shape parameter is positive, using a Gumbell distribution is likely to produce results which are optimistic.

8 Conclusions

In this survey, we reviewed research into probabilistic timing analysis techniques for hard real-time systems. We covered the main subject areas: static probabilistic timing analysis (SPTA), measurement-based probabilistic timing analysis (MBPTA), and hybrid methods (HyPTA), as well as reviewing supporting mechanisms and techniques, case studies, and evaluations.

8.1 Open Issues

We conclude by identifying open issues, key challenges and possible directions for future research. We present these open issues and challenges as a series of questions. While there has been progress in some of these areas, as highlighted in this survey, comprehensive solutions are currently tantalizingly out-of-reach. Further research is needed to secure a sound and comprehensive basis for the potential subsequent development and deployment of industry strength tools.

1. How much hardware time-randomisation is needed to ease the use of MBPTA methods in practice? Is custom time-randomised hardware with random replacement caches, random permutation buses etc. really necessary?
2. What are the hazards involved in applying MBPTA methods to systems comprising time-predictable COTS hardware that use entirely deterministic policies, and how can these hazards be overcome?
3. How can we solve the *path coverage problem* such that the user of MBPTA methods is not required to provide a measurement protocol that exercises all possible paths through the code? What level of coverage is needed for the MBPTA methods to provide sound results?
4. How can we apply MBPTA to multi-path programs? Do we have to apply MBPTA to each path individually and then combine the results to obtain a valid upper bound? Or is it possible to use measurements from different paths as input into EVT and still obtain sound results? (There is an argument that doing so makes it much harder to ensure representativity).
5. How can we solve the *representativity problem* such that the sample of observations used as input to MBPTA result in a pWCET distribution that correctly characterises the behaviour of the system during any future scenario of operation?
6. Given that testing can continue to produce execution time observations almost indefinitely, how do we know when sufficient observations have been obtained for an accurate estimate of the pWCET distribution to be derived?
7. How can we validate that a MBPTA implementation actually produces correct results?
8. How can we provide MBPTA for systems where execution time observations exhibit dependences?
9. How to apply MBPTA methods to systems using multi-core processors where there is substantial contention for shared hardware resources (e.g. interconnect, memory hierarchy, caches, DRAM etc.) between programs running on different cores?
10. How to apply MBPTA methods to systems that make use of multi-threading on a single core, and thus interleave the execution of a number of programs resulting in interference on shared hardware resources?
11. How can we mechanise MBPTA so that, for example block sizes and thresholds can be selected automatically?
12. How can we convince certification authorities that estimated pWCET distributions derived via MBPTA methods are safe?

8.2 Directions for Future Research

We end this survey with a discussion of an important direction for future real-time systems research which probabilistic analysis techniques may be able contribute to.

There is a continuing trend in industry sectors including avionics, automotive electronics, consumer electronics, and robotics away from development and deployment on single-core processors towards using significantly more powerful and complex Common-Off-The-Shelf (COTS) multi-core and many-core hardware platforms. This trend is driven by requirements on size, weight and power consumption, increasing cost pressures and the demand for more complex and capable functionality delivered through software. The use of COTS multi-core hardware poses significant challenges in terms of verifying timing behaviour and ensuring that real-time constraints are met. These challenges stem from the complexity of the architecture and the way in which hardware resources such as the interconnect and the memory hierarchy are shared between different processing cores. Some researchers are seeking to address these problems through approaches based on partitioning and separation (e.g. single-core equivalence [91]), while others aim for solutions based on considering the explicit interference on each hardware resource from co-running programs and

how this demand can be served by the available resource supply [8, 37]. There is the potential for probabilistic timing analysis and probabilistic schedulability analysis techniques (reviewed in a companion survey [38]) to play a role in the timing verification of such complex real-time systems.

There are a number of ways in which the interference effects of cross-core contention can be considered within a framework of probabilistic timing verification:

- By running synthetic "worst-case" contenders on other cores during the analysis phase. The idea being to account for the worst-case cross-core interference that could possibly occur irrespective of the actual co-runners, within the estimated pWCET distributions. The difficulty with this context-independent approach lies in determining which contenders, or combination of contenders, actually produce the worst-case interference for a given program. Further, the cross-core interference assumed may be very pessimistic, i.e. much larger than can actually occur in the operational system with the real co-runners.
- By running the real co-runners during the analysis phase. This approach has the advantage that it can potentially avoid some of the pessimism in the above context-independent approach. The estimated pWCET distributions obtained would only apply for the specific set of co-runners. Thus they would be context-dependent. This approach has the disadvantage that it makes the problem of representativity even more acute, since different combinations and timings of co-runners need to be considered.
- By using isolation or partial isolation techniques (implemented in either software or hardware) to limit or bound the interference that can occur due to contention for shared resources in a way that is independent of the actual behaviour of the co-runners. Synthetic contenders can then be used to generate this maximum interference at analysis time enabling sound upper bounding of the pWCET distributions that can occur during normal operation of the system.
- By running the program under analysis in isolation with all the other cores idle. In this case, the estimated pWCET distributions obtained would not include the impact of any co-runners. Instead, these effects would need to be integrated at a later stage, for example within some form of probabilistic schedulability analysis. The advantage of this approach is that it reduces the issues of representativity to the single core case; however, difficulties remain in soundly including the effects of cross-core contention without the results becoming either unsound or highly pessimistic.
- By running the complete system and applying statistical analysis (e.g. based on EVT) to response times, rather than execution times (see Section 7 of the companion survey [38] for a review of the initial work in this area). This approach again raises difficulties with representativity; however, it has the advantage that it treats the entire system as a grey box requiring less detailed information about the hardware and software behaviour.

Since the initial work of Burns and Edgar [22] in 2000 on probabilistic timing analysis, significant progress has been made in the development of both measurement-based and static probabilistic timing analysis techniques. However, there are still many important unanswered questions and open issues that need to be addressed. In particular, work on probabilistic timing analysis and probabilistic schedulability analysis for multi-core and many-core systems is in its infancy with opportunities for significant advances addressing an important research challenge.

Acknowledgements. The research that went into writing this survey was funded, in part, by the Inria International Chair program and the ESPRC grant MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.

The authors would like to thank David Griffin and Alan Burns for their comments on an earlier draft of this survey.

References

1 J. Abella, F. J. Cazorla, E. Quinones, and T. Vardanega. Measurement-based probabilistic timing analysis and i.i.d property. White Paper Version 2. Technical report http://www.proartis-project.eu/publications/MBPTA-white-paper, BSC, July 2014.

2 J. Abella, D. Hardy, I. Puaut, E. Quiñones, and F. J. Cazorla. On the Comparison of Deterministic and Probabilistic WCET Estimation Techniques. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 266–275, July 2014. doi:10.1109/ECRTS.2014.16.

3 J. Abella, M. Padilla, J. Del Castillo, and F. J. Cazorla. Measurement-Based Worst-Case Execution Time Estimation Using the Coefficient of Variation. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4):72:1–72:29, June 2017. doi:10.1145/3065924.

4 J. Abella, E. Quiñones, F. Wartel, T. Vardanega, and F. J. Cazorla. Heart of Gold: Making the Improbable Happen to Increase Confidence in MBPTA. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 255–265, 2014. doi:10.1109/ECRTS.2014.33.

5 I. Agirre, M. Azkarate-askasua, C. Hernandez, J. Abella, J. Perez, T. Vardanega, and F. J. Cazorla. IEC-61508 SIL 3 Compliant Pseudo-Random Number Generators for Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 677–684, August 2015. doi:10.1109/DSD.2015.26.

6 S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Springer Real-Time Systems*, 51(1):77–123, 2015. doi:10.1007/s11241-014-9218-4.

7 S. Altmeyer and R. I. Davis. On the Correctness, Optimality and Precision of Static Probabilistic Timing Analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 26:1–26:6, 2014. URL: http://dl.acm.org/citation.cfm?id=2616606.2616638.

8 S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. A Generic and Compositional Framework for Multicore Response Time Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 129–138, 2015. doi:10.1145/2834848.2834862.

9 H. Anwar, C. Chen, and G. Beltrame. A probabilistically analysable cache implementation on FPGA. In *IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 1–4, June 2015. doi:10.1109/NEWCAS.2015.7181984.

10 I. Bate and U. Khan. WCET Analysis of Modern Processors Using Multi-criteria Optimisation. *Empirical Softw. Engg.*, 16(1):5–28, February 2011.

11 P. Benedicte, C. Hernandez, J. Abella, and F. J. Cazorla. Design and integration of hierarchical-placement multi-level caches for real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 455–460, March 2018. doi:10.23919/DATE.2018.8342052.

12 P. Benedicte, C. Hernandez, J. Abella, and F. J. Cazorla. RPR: A Random Replacement Policy with Limited Pathological Replacements. In *Proceedings of ACM Symposium on Applied Computing (SAC)*, pages 593–600, 2018. doi:10.1145/3167132.3167197.

13 P. Benedicte, L. Kosmidis, E. Quinones, J. Abella, and F. J. Cazorla. Modelling the confidence of timing analysis for time randomised caches. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, May 2016. doi:10.1109/SIES.2016.7509421.

14 P. Benedicte, L. Kosmidis, E. Quiñones, J. Abella, and F. J. Cazorla. A confidence assessment of WCET estimates for software time randomized caches. In *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, pages 90–97, July 2016. doi:10.1109/INDIN.2016.7819140.

15 K. Berezovskyi, F. Guet, L. Santinelli, K. Bletsas, and E. Tovar. Measurement-Based Probabilistic Timing Analysis for Graphics Processor Units. In *Proceedings of the International Conference on the Architecture of Computing Systems (ARCS)*, pages 223–236, April 2016. doi:10.1007/978-3-319-30695-7_17.

16 K. Berezovskyi, L. Santinelli, K. Bletsas, and E. Tovar. WCET Measurement-based and Extreme Value Theory Characterisation of CUDA Kernels. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 279–288, 2014. doi:10.1145/2659787.2659827.

17 G. Bernat, A. Burns, and M. Newby. Probabilistic Timing Analysis: An Approach Using Copulas. *J. Embedded Comput.*, 1(2):179–194, April 2005. URL: http://dl.acm.org/citation.cfm?id=1233760.1233763.

18 G. Bernat, A. Colin, and S. Petters. pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems. Technical report, Department of Computer Science, University of York, 2003.

19 G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 279–288, 2002. doi:10.1109/REAL.2002.1181582.

20 B. Braams, S. Altmeyer, and A. D. Pimentel. EDiFy: An execution time distribution finder. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2017. doi:10.1145/3061639.3062233.

21 S. Bunte, M. Zolda, M. Tautschnig, and R. Kirner. Improving the Confidence in Measurement-Based Timing Analysis. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 144–151, March 2011. doi:10.1109/ISORC.2011.27.

22 A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 89–96, 2000. doi:10.1109/EMRTS.2000.853996.

23 A. Burns and D. Griffin. Predictability as an emergent behaviour. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pages 27–29, 2011.

24 S. Bünte, M. Zolda, and R. Kirner. Let's get less optimistic in measurement-based timing analysis. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 204–212, June 2011. doi:10.1109/SIES.2011.5953663.

25 F. J. Cazorla, J. Abella, J. Andersson, T. Vardanega, F. Vatrinet, I. Bate, I. Broster, M. Azkarate-Askasua, F. Wartel, L. Cucu, F. Cros, G. Farrall, A. Gogonel, A. Gianarro, B. Triquet, C. Hernandez, C. Lo, C. Maxim, D. Morales, E. Quinones, E. Mezzetti, L. Kosmidis, I. Aguirre, M. Fernandez, M. Slijepcevic, P. Conmy, and W. Talaboulma. PROXIMA: Improving Measurement-Based Timing Analysis through Randomisation and Probabilistic Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 276–285, August 2016. doi:10.1109/DSD.2016.22.

26 F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 12(2s):94:1–94:26, May 2013. doi:10.1145/2465787.2465796.

27 F. J. Cazorla, T. Vardanega, E. Quiñones, and J. Abella. Upper-bounding Program Execution Time with Extreme Value Theory. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 64–76, 2013. doi:10.4230/OASIcs.WCET.2013.64.

28 C. Chen and G. Beltrame. An Adaptive Markov Model for the Timing Analysis of Probabilistic Caches. *ACM Trans. Des. Autom. Electron. Syst.*, 23(1):12:1–12:24, August 2017. doi:10.1145/3123877.

29 C. Chen, J. Panerati, and G. Beltrame. Effects of online fault detection mechanisms on Probabilistic Timing Analysis. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 41–46, September 2016. doi:10.1109/DFT.2016.7684067.

30 C. Chen, J. Panerati, I. Hafnaoui, and G. Beltrame. Static probabilistic timing analysis with a permanent fault detection mechanism. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, June 2017. doi:10.1109/SIES.2017.7993373.

31 C. Chen, L. Santinelli, J. Hugues, and G. Beltrame. Static probabilistic timing analysis in presence of faults. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016. doi:10.1109/SIES.2016.7509422.

32 S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001. doi:10.1007/978-1-4471-3675-0.

33 F. Cros, L. Kosmidis, F. Wartel, D. Morales, J. Abella, I. Broster, and F. J. Cazorla. Dynamic software randomisation: Lessons learned from an aerospace case study. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 103–108, March 2017. doi:10.23919/DATE.2017.7926966.

34 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, July 2012. doi:10.1109/ECRTS.2012.31.

35 L. David and I. Puaut. Static determination of probabilistic execution times. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 223–230, June 2004. doi:10.1109/EMRTS.2004.1311024.

36 R. A. Davis and T. Mikosch. The extremogram: A correlogram for extreme events. *Bernoulli*, 15(4):977–1009, November 2009. doi:10.3150/09-BEJ213.

37 R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. An extensible framework for multicore response time analysis. *Springer Real-Time Systems*, 54(3):607–661, July 2018. doi:10.1007/s11241-017-9285-4.

38 R. I. Davis and L. Cucu-Grosjean. A Survey of Probabilistic Schedulability Analysis Techniques for Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems (LITES)*, 6(1):04:1–04:53, May 2019. doi:10.4230/LITES-v006-i001-a004.

39 R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of Probabilistic Cache Related Pre-emption Delays. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–179, July 2013. doi:10.1109/ECRTS.2013.27.

40 R. I. Davis, J. Whitham, and D. Maxim. Static Probabilistic Timing Analysis for Multicore Processors with Shared Cache. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, pages 3–5, 2013.

41 R.I. Davis. Improvements to Static Probabilistic Timing Analysis for Systems with Random Cache Replacement Policies. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, pages 22–24, July 2013.

42 J-F. Deverge and I. Puaut. Safe measurement-based WCET estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, 2005.

43 E. Díaz, M. Fernández, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and F. J. Cazorla. *MC2: Multicore and Cache Analysis via Deterministic and Probabilistic Jitter Bounding*, pages 102–118.

Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-60588-3_7.

44 J. L. Diaz, J. M. Lopez, M. Garcia, A. M. Campos, Kanghee Kim, and L. L. Bello. Pessimism in the stochastic analysis of real-time systems: concept and applications. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 197–207, December 2004. doi:10.1109/REAL.2004.41.

45 S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 215–224, December 2001. doi:10.1109/REAL.2001.990614.

46 P. Embrechts, C. Kluppelberg, and T. Mikosch. *Modelling extremal events for insurance and Finance*. Springer, 1997. doi:10.1007/978-3-642-33483-2.

47 I. Fedotova, B. Krause, and E. Siemens. Applicability of Extreme Value Theory to the Execution Time Prediction of Programs on SoCs. In *Proceedings of the International Conference on Applied Innovations in IT (ICAIIT)*, March 2017.

48 M. Fernandez, D. Morales, L. Kosmidis, A. Bardizbanyan, I. Broster, C. Hernandez, E. Quinones, J. Abella, F. Cazorla, P. Machado, and L. Fossati. Probabilistic timing analysis on time-randomized platforms for the space domain. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 738–739, March 2017. doi:10.23919/DATE.2017.7927087.

49 S. Jimenez Gil, I. Bate, G. Lima, L. Santinelli, A. Gogonel, and L. Cucu-Grosjean. Open Challenges for Probabilistic Measurement-Based Worst-Case Execution Time. *IEEE Embedded Systems Letters*, PP(99):1–1, 2017. doi:10.1109/LES.2017.2712858.

50 D. Griffin, I. Bate, B. Lesage, and F. Soboczenski. Evaluating Mixed Criticality Scheduling Algorithms with Realistic Workloads. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2015.

51 D. Griffin and A. Burns. Realism in Statistical Analysis of Worst Case Execution Times. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 44–53, 2010. doi:10.4230/OASIcs.WCET.2010.44.

52 D. Griffin, B. Lesage, A. Burns, and R. I. Davis. Static Probabilistic Timing Analysis of Random Replacement Caches Using Lossy Compression. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 289–298, 2014. doi:10.1145/2659787.2659809.

53 F. Guet, L. Santinelli, and J. Morio. On the Reliability of the Probabilistic Worst-Case Execution Time Estimates. In *Proceedings of the European Congress on Embedded Real Time Software and Systems (ERTS)*, January 2016. URL: https://hal.archives-ouvertes.fr/hal-01289477.

54 F. Guet, L. Santinelli, and J. Morio. Probabilistic analysis of cache memories and cache memories impacts on multi-core embedded systems. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, May 2016. doi:10.1109/SIES.2016.7509420.

55 F. Guet, L. Santinelli, and J. Morio. On the Representativity of Execution Time Measurements: Studying Dependence and Multi-Mode Tasks. In Jan Reineke, editor, *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 57 of *OASICS*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/OASIcs.WCET.2017.3.

56 J. Gustafsson, A. Betts, A. Ermedahl, and B. Lisper. The Mälardalen WCET Benchmarks – Past, Present and Future. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 137–147, July 2010.

57 J. Hansen, S. A. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation . In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 252, 2009.

58 D. Hardy and I. Puaut. Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 35–44, 2013. doi:10.1145/2516821.2516842.

59 D. Hardy and I. Puaut. Static Probabilistic Worst Case Execution Time Estimation for Architectures with Faulty Instruction Caches. *Springer Real-Time Systems*, 51(2):128–152, March 2015. doi:10.1007/s11241-014-9212-x.

60 C. Hernandez, J. Abella, F. J. Cazorla, J. Andersson, and A. Gianarro. Towards making a LEON3 multicore compatible with probabilistic timing analysis. In *Proceedings of the Data Systems In Aerospace Conference (DASIA)*, May 2015.

61 C. Hernandez, J. Abella, A. Gianarro, J. Andersson, and F. J. Cazorla. Random Modulo: A New Processor Cache Design for Real-time Critical Systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 29:1–29:6, 2016. doi:10.1145/2897937.2898076.

62 K. Höfig. *Failure-Dependent Timing Analysis - A New Methodology for Probabilistic Worst-Case Execution Time Analysis*, pages 61–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-28540-0_5.

63 T. Hsing. On Tail Index Estimation Using Dependent Data. *The Annals of Statistics*, 19(3):1547–1569, 1991. URL: http://www.jstor.org/stable/2241962.

64 M. Ivers and R. Ernst. *Probabilistic Network Loads with Dependencies and the Effect on Queue Sojourn Times*, pages 280–296. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-10625-5_18.

65 J. Jalle, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Bus Designs for Time-probabilistic Multicore Processors. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 50:1–50:6, 2014. URL: http://dl.acm.org/citation.cfm?id=2616606.2616668.

66 T. Kelter, H. Falk, P. Marwedel, S. Chattopadhyay, and A. Roychoudhury. Static Analysis of Multi-core TDMA Resource Arbitration

Delays. *Springer Real-Time Systems*, 50(2):185–229, March 2014. doi:10.1007/s11241-013-9189-x.

67 L. Kosmidis, J. Abella, E. Quinones, and F. J. Cazorla. Efficient Cache Designs for Probabilistically Analysable Real-Time Systems. *IEEE Transactions on Computers*, 63(12):2998–3011, December 2014. doi:10.1109/TC.2013.182.

68 L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. A cache design for probabilistically analysable real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 513–518, March 2013. doi:10.7873/DATE.2013.116.

69 L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Multi-level Unified Caches for Probabilistically Time Analysable Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 360–371, December 2013. doi:10.1109/RTSS.2013.43.

70 L. Kosmidis, J. Abella, F. Wartel, E. Quiñones, A. Colin, and F. J. Cazorla. PUB: Path Upper-Bounding for Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 276–287, July 2014. doi:10.1109/ECRTS.2014.34.

71 L. Kosmidis, D. Compagnin, D. Morales, E. Mezzetti, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Measurement-Based Timing Analysis of the AURIX Caches. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, 2016.

72 L. Kosmidis, C. Curtsinger, E. Quiñones, J. Abella, E. Berger, and F. J. Cazorla. Probabilistic timing analysis on conventional cache designs. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 603–606, March 2013. doi:10.7873/DATE.2013.132.

73 L. Kosmidis, E. Quiñones, J. Abella, G. Farrall, F. Wartel, and F. J. Cazorla. Containing Timing-Related Certification Cost in Automotive Systems Deploying Complex Hardware. In *Proceedings of the Design Automation Conference (DAC)*, pages 22:1–22:6, 2014. doi:10.1145/2593069.2593112.

74 L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, I. Broster, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis and Its Impact on Processor Architecture. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 401–410, August 2014. doi:10.1109/DSD.2014.50.

75 L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, and F. J. Cazorla. Achieving timing composability with measurement-based probabilistic timing analysis. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 1–8, June 2013. doi:10.1109/ISORC.2013.6913193.

76 L. Kosmidis, E. Quiñones, J. Abella, T. Vardanega, C. Hernandez, A. Gianarro, I. Broster, and F. J. Cazorla. Fitting processor architectures for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 2016. doi:10.1016/j.micpro.2016.07.014.

77 L. Kosmidis, T. Vardanega, J. Abella, E. Quiñones, and F. J. Cazorla. Applying Measurement-Based Probabilistic Timing Analysis to Buffer Resources. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 97–108, 2013. doi:10.4230/OASIcs.WCET.2013.97.

78 L. Kosmidis, R. Vargas, D. Morales, E. Quiñones, J. Abella, and F. J. Cazorla. TASA: Toolchain-Agnostic Static Software randomisation for critical real-time systems. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, November 2016. doi:10.1145/2966986.2967078.

79 K. Lahiri, A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: a new high-performance communication architecture for system-on-chip designs. In *DAC*, pages 15–20, 2001. doi:10.1109/DAC.2001.156100.

80 S. Law and I. Bate. Achieving Appropriate Test Coverage for Reliable Measurement-Based Timing Analysis. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 189–199, July 2016. doi:10.1109/ECRTS.2016.21.

81 M. R. Leadbetter, G. Lindgren, and H. Rootzen. Conditions for the convergence in distribution of maxima of stationary normal processes. *Stochastic Processes and their Applications*, 8(2), 1978.

82 B. Lesage, D. Griffin, S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. On the analysis of random replacement caches using static probabilistic timing methods for multi-path programs. *Real-Time Systems*, December 2017. doi:10.1007/s11241-017-9295-2.

83 B. Lesage, D. Griffin, S. Altmeyer, and R. I. Davis. Static Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 361–372, December 2015. doi:10.1109/RTSS.2015.41.

84 B. Lesage, D. Griffin, R. I. Davis, and S. Altmeyer. On the application of Static Probabilistic Timing Analysis to Memory Hierarchies. In *Proceedings of the Real-Time Scheduling Open Problems Seminar (RTSOPS)*, 2014.

85 B. Lesage, D. Griffin, F. Soboczenski, I. Bate, and R. I. Davis. A Framework for the Evaluation of Measurement-based Timing Analyses. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 35–44, 2015. doi:10.1145/2834848.2834858.

86 Y. Liang and T. Mitra. Cache modeling in probabilistic execution time analysis. In *Proceedings of the Design Automation Conference (DAC)*, pages 319–324, June 2008.

87 G. Lima and I. Bate. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

88 G. Lima, D. Dias, and E. Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016.

89 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A New Way About Using Statistical Analysis of Worst-case Execution Times. *SIGBED Rev.*, 8(3):11–14, September 2011. doi:10.1145/2038617.2038619.

90 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A trace-based statistical worst-case execution time analysis of component-based real-time embedded systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4, September 2011. doi:10.1109/ETFA.2011.6059190.

91 R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. WCET(m) Estimation in Multi-core Systems Using Single Core Equivalence. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 174–183, July 2015. doi:10.1109/ECRTS.2015.23.

92 C. Maxim, A. Gogonel, I. Asavoae, M. Asavoae, and L. Cucu-Grosjean. Reproducibility and representativity: mandatory properties for the compositionality of measurement-based WCET estimation approaches. *SIGBED Review*, 14(3):24–31, 2017. doi:10.1145/3166227.3166230.

93 A. Melani, E. Noulard, and L. Santinelli. Learning from probabilities: Dependences within real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2013. doi:10.1109/ETFA.2013.6648013.

94 E. Mezzetti, M. Fernandez, A. Bardizbanyan, I Agirre, , J. Abella, T. Vardanega, and F. J. Cazorla. EPC Enacted: Integration in an Industrial Toolbox and Use Against a Rialway Application. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

95 E. Mezzetti, N. Holsti, A. Colin, G. Bernat, and T. Vardanega. Attacking the sources of unpredictability in the instruction cache behavior. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, 2008.

96 E. Mezzetti, M. Ziccardi, T. Vardanega, J. Abella, E. Quiñones, and F. J. Cazorla. Randomized Caches Can Be Pretty Useful to Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 2(1):01–1–01:10, 2015. doi:10.4230/LITES-v002-i001-a001.

97 a. Milutinovic, j. Abella, i. Agirre, M. Azkarate-Askasua, E. Mezzetti, T. Vardanega, and F. J. Cazorla. *Software Time Reliability in the Presence of Cache Memories*, pages 233–249. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-60588-3_15.

98 S. Milutinovic, J. Abella, and F. J. Cazorla. Modelling Probabilistic Cache Representativeness in the Presence of Arbitrary Access Patterns. In *Proceedings of the IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC)*, pages 142–149, May 2016. doi:10.1109/ISORC.2016.28.

99 S. Milutinovic, J. Abella, and F. J. Cazorla. On the assessment of probabilistic WCET estimates reliability for arbitrary programs. *EURASIP Journal on Embedded Systems*, 2017(1):28, April 2017. doi:10.1186/s13639-017-0076-8.

100 S. Milutinovic, J. Abella, E. Mezzetti, and F. J. Cazorla. Measurement-based Cache Representativeness on Multipath Programs. In *Proceedings of the Design Automation Conference (DAC)*, pages 123:1–123:6, 2018. doi:10.1145/3195970.3196075.

101 S. Milutinovic, E. Mezzetti, J. Abella, T. Vardanega, and F. J. Cazorla. On uses of extreme value theory fit for industrial-quality WCET analysis. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–6, June 2017. doi:10.1109/SIES.2017.7993402.

102 M. Panic, J. Abella, C. Hernandez, E. Quiñones, T. Ungerer, and F. J. Cazorla. Enabling TDMA Arbitration in the Context of MBPTA. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 462–469, August 2015. doi:10.1109/DSD.2015.68.

103 M. Panić, J. Abella, E. Quiñones, C. Hernandez, T. Ungerer, and F. J. Cazorla. Adapting TDMA arbitration for measurement-based probabilistic timing analysis. *Microprocessors and Microsystems*, 52:188–201, 2017. doi:10.1016/j.micpro.2017.06.006.

104 B. Pasdeloup. Static probabilistic timing analysis of worst-case execution time for random replacement caches. Technical report, Inria, 2014.

105 J. Pickands. Statistical Inference Using Extreme Order Statistics. *Ann. Statist.*, 3(1):119–131, January 1975. doi:10.1214/aos/1176343003.

106 E. Quinones, E. D. Berger, G. Bernat, and F. J. Cazorla. Using Randomized Caches in Probabilistic Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 129–138, July 2009. doi:10.1109/ECRTS.2009.30.

107 F. Reghenzani, G. Massari, and Fornaciari W. chronovise: Measurement-Based Probabilistic Timing Analysis framework. *Journal of Open Source Software*, 3(28), June 2018. doi:10.21105/joss.00711.

108 J. Reineke. Randomized Caches Considered Harmful in Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems*, 1(1):03–1–03:13, 2014. doi:10.4230/LITES-v001-i001-a003.

109 A. Rukhin, J. Soto, J. Nechvatal, E. Barker, S. Leigh, M. Levenson, D. Banks, A. Heckert, J. Dray, S. Vo, A. Rukhin, J. Soto, M. Smid, S. Leigh, M. Vangel, A. Heckert, J. Dray, and L. E. Bassham. Statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST special publication, 2010.

110 L. Santinelli, F. Guet, and J. Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the IEEE Real-Time*

and Embedded Technology and Applications Symposium (RTAS), April 2017.

111 L. Santinelli and Z. Guo. *On the Criticality of Probabilistic Worst-Case Execution Time Models*, pages 59–74. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-69483-2_4.

112 L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 21–30, 2014. doi:10.4230/OASIcs.WCET.2014.21.

113 M. Santos, B. Lisper, G. Lima, and V. Lima. Sequential Composition of Execution Time Distributions by Convolution. In *Proceedings of the Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (CRTS)*, pages 30–37, November 2011. URL: http://www.es.mdh.se/publications/2215-.

114 C. Scarrott and A. MacDonald. A review of extreme value threshold estimation and uncertainty quantification. *REVSTAT–Statistical Journal*, 10(1):33–60, 2012.

115 M. Schlansker, R. Shaw, and S. Sivaramakrishnan. *Randomization and associativity in the design of placement-insensitive caches*. Hewlett Packard Laboratories, 1993.

116 K. P. Silva and R. Silva de Oliveira L. F. Arcaro. On Using GEV or Gumbel Models when Applying EVT for Probabilistic WCET Estimation. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, 2017.

117 M. Slijepcevic, M. Fernandez, C. Hernandez, J. Abella, E. Quinones, and F. J. Cazorla. pT-NoC: Probabilistic Time-Analyzable Tree-Based NoC for Mixed Criticality Systems. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, 2016.

118 M. Slijepcevic, C. Hernandez, J. Abella, and F. J. Cazorla. Boosting Guaranteed Performance in Wormhole NoCs with Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, pages 440–444, August 2017. doi:10.1109/DSD.2017.71.

119 M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. DTM: Degraded Test Mode for Fault-Aware Probabilistic Timing Analysis. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 237–248, July 2013. doi:10.1109/ECRTS.2013.33.

120 M. Slijepcevic, L. Kosmidis, J. Abella, E. Quiñones, and F. J. Cazorla. Time-analysable non-partitioned shared caches for real-time multicore systems. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6, June 2014. doi:10.1145/2593069.2593235.

121 J. E. Smith and J. R. Goodman. A Study of Instruction Cache Organizations and Replacement Policies. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*, ISCA '83, pages 132–137, New York, NY, USA, 1983. ACM. doi:10.1145/800046.801648.

122 J. E. Smith and J. R. Goodman. Instruction Cache Replacement Policies and Organizations. *IEEE Transactions on Computers*, C-34(3):234–241, March 1985. doi:10.1109/TC.1985.1676566.

123 Z. Stephenson, J. Abella, and T. Vardanega. Supporting industrial use of probabilistic timing analysis with explicit argumentation. In *Proceedings of the IEEE International Conference on Industrial Informatics (INDIN)*, pages 734–740, July 2013. doi:10.1109/INDIN.2013.6622975.

124 N. Topham and A. Gonzalez. Randomized cache placement for eliminating conflicts. *IEEE Transactions on Computers*, 48(2):185–192, February 1999. doi:10.1109/12.752660.

125 N. Tracey, J. Clark, K. Mander, and J. McDermid. An automated framework for structural test-data generation. In *Proceedings 13th IEEE International Conference on Automated Software Engineering*, pages 285–288, October 1998. doi:10.1109/ASE.1998.732680.

126 N. Tracey, J. A. Clark, and K. Mander. The way forward for unifying dynamic test-case generation: The optimisation-based approach. *Proceedings of the IFIP International Workshop on Dependable Computing and Its Applications (DCIA)*., 1998.

127 D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla. Resilient random modulo cache memories for probabilistically-analyzable real-time systems. In *IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 27–32, July 2016. doi:10.1109/IOLTS.2016.7604666.

128 F. Wartel, L. Kosmidis, A. Gogonel, A. Baldovino, Z. Stephenson, B. Triquet, E. Quiñones, C. Lo, E. Mezzetta, I. Broster, J. Abella, L. Cucu-Grosjean, T. Vardanega, and F. J. Cazorla. Timing analysis of an avionics case study on complex hardware/software platforms. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 397–402, March 2015.

129 F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248, June 2013. doi:10.1109/SIES.2013.6601497.

130 J. Wegener and F. Mueller. A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints. *Real-Time Systems*, 21(3):241–268, November 2001.

131 J. Wegener, H. Sthamer, B. F. Jones, and D. E. Eyres. Testing real-time systems using genetic algorithms. *Software Quality Journal*, 6(2):127–135, June 1997. doi:10.1023/A:1018551716639.

132 I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-Based Timing Analysis. In *Leveraging Applications of Formal Methods, Verification and Validation*, pages 430–444, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

133 R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdin-

and, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The Worst-case Execution-time Problem Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3):36:1–36:53, May 2008. doi:10.1145/1347375.1347389.

134 N. Williams. WCET measurement using modified path testing. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 1 of *OpenAccess Series in Informatics (OASIcs)*, 2005. doi:10.4230/OASIcs.WCET.2005.809.

135 N. Williams and M. Roger. Test generation strategies to measure worst-case execution time. In *ICSE Workshop on Automation of Software Test*, pages 88–96, May 2009. doi:10.1109/IWAST.2009.5069045.

136 M. Ziccardi, E. Mezzetti, T. Vardanega, J. Abella, and F. J. Cazorla. EPC: Extended Path Coverage for Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 338–349, December 2015. doi:10.1109/RTSS.2015.39.

A Appendix: Measurement Protocols

In this appendix, we briefly discuss *measurement protocols* and test vector generation, since they underpin approaches to measurement-based timing analysis. We use the term *scenario* of operation to mean a potentially repeating sequence of runs of a program, starting from a feasible initial hardware state and progressing via a valid evolution of the program's input values. To run a program over a particular scenario of operation, a measurement protocol needs to provide information about the initial hardware configuration, which is used to set up the initial hardware state, and a sequence of input values, referred to as a *test vector*, which are input to the program as it iterates over a number of runs. The aim of a *measurement protocol* is to exercise the program in ways that are *relevant* to the parameter being measured. For example, if the parameter being measured is a code coverage metric, then the measurement protocol would aim to use a set of scenarios and test vectors designed to exercise paths that cover all of the statements (or all of the conditions) in the code. In the case of the WCET, the set of scenarios and test vectors need to be designed to exercise the longest paths through the code, assuming that those paths can be identified in some way. The major difficulty in designing an appropriate measurement protocol is in choosing which scenarios and hence which test vectors to use.

Search-based techniques were successfully applied to the problem of automatic test data generation for structural code coverage by Tracey et al. [125] in 1998. While measurement protocols designed for code coverage can potentially provide a useful starting point for the WCET problem, in general even MC-DC coverage is insufficient. Further, full path-coverage is typically unattainable due to issues of tractability, although some programs for high integrity systems may be simple enough that all paths can be covered. Structural coverage offers a more attractive starting point for hybrid measurement-based analysis which records execution times at the level of simple functions or sub-programs as discussed by Deverge and Puaut [42] in 2005; however, there are still issues with how these execution times are combined due to dependences on the previous history of execution.

Search techniques developed by Wegener et al. [131] in 1997, Tracey et al. [126] in 1998, Wegener and Mueller [130] in 2001, and multi-criteria optimisation developed by Bate and Khan [10] in 2011 have also been investigated in the context of test data generation with the aim of finding input values that result in large execution times. The basic idea is to use an evolutionary algorithm to mutate or evolve a population of test data, with the fitness function determined by the execution time of the program with that data as input. Multi-criteria optimisation works in a similar way, but takes into account additional criteria such as the number of cache misses as well as the execution time.

In 2005, Williams [134] proposed a static analysis tool that seeks to determine a set of test vectors that exercise every path. This was later extended in 2009 by Williams and Roger [135] with the aim of avoiding the need for full path coverage, for example by maximising loop counts.

In 2008, Wenzel et al. [132] described a tool for measurement-based timing analysis that uses a combination of test data reuse, random search, heuristics and model checking. It partitions the program into user defined segments (to ease path complexity), with instructions inserted at segment boundaries to ensure a consistent hardware state. In 2011, Bunte et al. [21] showed that commonly used metrics for functional testing including statement coverage, decision coverage, and MC/DC coverage are *insufficient* to obtain safe WCET estimates. Instead, they propose a balanced path metric which ensures that all feasible pairs of basic blocks are exercised in combination. This metric is shown to be much more effective than the common code coverage metrics, but is still not completely safe. Later in 2011, Bunte et al. [24] explored the combination of model checking, used to produce a set of test vectors that provide basic block coverage, and a genetic algorithm, which aims to modify these test vectors to maximise execution times. They evaluated this combined approach using the relative safety metric developed previously [21].

Recent work by Law and Bate [80] in 2016, aimed at maximising loop bounds and achieving path coverage at the level of individual functions. The techniques proposed make use of simulated annealing in combination with fitness functions that target branch coverage and loop counts as well as execution time. They show that this approach is more effective in providing WCET approximations than fitness functions based solely on maximising execution time.

While these methods are effective in finding large execution times, there are no guarantees that test data which results in the worst-case execution time will be found.

Further related work in 2017 by Braams et al. [20] presented EDiFy, a measurement-based framework that aims to derive the execution time distribution of a program via exhaustive evaluation of the program inputs. Since the execution time distribution depends on the distribution of input values, the input value distribution is assumed to be provided for each independent input variable and also the conditional distribution for any dependent variables. EDiFy addresses issues of tractability via a combination of static analysis and an anytime algorithm. Static analysis is used to reduce the state-space by pruning irrelevant input variables, and clustering variable ranges where the execution time is guaranteed to be the same. The anytime algorithm makes use of a logarithmic traversal function over the variable ranges. This ensures rapid convergence and an early tight approximation of the execution time distribution. Execution times are obtained by running the program with the selected input values.

Although we have touched upon issues of test data generation and measurement protocols in the above discussion a comprehensive review of research in this area is outside of the scope of this survey. As far as we are aware, to date there has only been very limited work done on the problem of defining appropriate measurement protocols to support MBPTA. The majority of works on MBPTA (see Section 4) aim at analysing single paths and focus on obtaining sufficient observations for the path under analysis. They then rely on additional knowledge to identify the worst-case paths or existing functional testing to provide sufficient path coverage. A pWCET distribution for the program is then constructed using an envelope over the pWCET distributions for individual paths, i.e. the per-path method (see Section 2.3). Cucu-Grosjean et al. [34] note that full path coverage is a pre-requisite to obtaining sound results from MBPTA. This is backed up by the empirical work of Lesage et al. [85] which shows that omitting some paths can quickly degrade the estimated pWCET distribution output by MBPTA leading to optimistic (i.e. unsound) results. Hybrid methods (see Section 5) go some way to addressing the path coverage problem; however, they are limited in their applicability. Further research is clearly needed to define appropriate measurement protocols that can fully support MBPTA methods, while also addressing issues of representativity.

A Survey of Probabilistic Schedulability Analysis Techniques for Real-Time Systems

Robert I. Davis
University of York, UK and Inria, France
rob.davis@york.ac.uk

Liliana Cucu-Grosjean
Inria, France
liliana.cucu@inria.fr

Abstract

This survey covers schedulability analysis techniques for probabilistic real-time systems. It reviews the key results in the field from its origins in the late 1980s to the latest research published up to the end of August 2018. The survey outlines fundamental concepts and highlights key issues. It provides a taxonomy of the different methods used, and a classification of existing research. A detailed review is provided covering the main subject areas as well as research on supporting techniques. The survey concludes by identifying open issues, key challenges and possible directions for future research.

2012 ACM Subject Classification Software and its engineering → Software organization and properties, Software and its engineering → Software functional properties, Software and its engineering → Real-time schedulability, Computer systems organization → Real-time systems
Keywords and Phrases Probabilistic, real-time, schedulability analysis, scheduling
Digital Object Identifier 10.4230/LITES-v006-i001-a004
Received 2018-01-04 **Accepted** 2019-02-26 **Published** 2019-05-14

1 Introduction

Systems are characterised as *real-time* if, as well as meeting functional requirements, they are required to meet timing requirements. Real-time systems may be further classified as *hard* real-time, where failure to meet their timing requirements constitutes a failure of the system, or *soft* real-time, where failure to meet timing requirements leads only to a degraded quality of service. Today, both hard and soft real-time systems are found in many diverse application areas including: automotive, aerospace, medical systems, robotics, and consumer electronics.

Real-time systems are typically implemented via a set of programs, also referred to as tasks, which are executed on a recurring basis. Verifying the timing correctness of a real-time system is typically framed as a two step process:

- *Timing Analysis* seeks to characterise the amount of time which each task can take to execute on the hardware platform. Typically, this is done by estimating, as a single value, an upper bound on the *Worst-Case Execution Time* (WCET) of the task.
- *Schedulability Analysis* seeks to characterise the end-to-end response time of functionality involving one or more tasks, taking into account the way in which the tasks are scheduled and any interference between them. An upper bound on the Worst-Case Response Time (WCRT) can then be compared to the deadline for that function to determine if end-to-end timing requirements can be guaranteed.

The concept of *probabilistic real-time systems* departs in two main ways from the classical model described above. Firstly, it recognises that the execution times of tasks may exhibit significant variability due to hardware effects (e.g. caches, branch prediction, pipelines, and other hardware acceleration features) as well as due to different input values and paths taken through the code.

Thus the WCET may be substantially larger than typical execution times and may rarely occur. Much of the work on the analysis of probabilistic real-time systems therefore models the execution time of each task using a probability distribution with distinct probabilities associated with each possible value of execution time. Secondly, the timing requirements are such that deadlines are no longer considered to require absolute guarantees, but rather the probability of the deadline being exceeded must be below some specified threshold. (In practice, there may also be constraints on the number of consecutive deadlines that can be missed and hence on the probability that such a black-out period exceeds a certain length).

The concept of probabilistic real-time systems spans both the traditional classifications of hard and soft real time systems. In the case of a hard real-time system, failure to meet a deadline may constitute a failure of the system; however, provided that the probability of such a failure occurring is sufficiently small, for example leading to a *failure rate* of no more than 10^{-7}, 10^{-8}, or 10^{-9} per hour of operation, then it may still be acceptable to the system designers. This stems from the observation that the mechanical and electrical components of systems are typically designed with similar failure rates in mind (measured in terms of the number of failures per hour or billion hours). Thus engineering the timing behaviour of a system function to ensure a much smaller failure rate would have little or no impact on overall reliability and availability, while it could potentially require the provision of much more costly hardware. Acceptable failure rates depend on the criticality level assigned to the system function, as an example in the automotive standard ISO-26262 each Automotive Safety Integrity Level (ASIL) is associated with an observable failure rate. These are 10^{-9} per hour for ASIL D, 10^{-8} for ASIL C and B, and 10^{-7} for ASIL A. (Note, the relationship between failure rates per hour of operation and appropriate thresholds on probabilities of failure for individual tasks depend on various factors considered in fault tree analysis, including any mitigations and recovery mechanisms that may be applied in the event of a timing failure [62]).

We note that traditional analysis techniques can still be applied in cases where a very low level of deadline misses are permissible; however, since the reasoning used can only argue that either all deadlines will be met or not, then the results produced can in some cases be very conservative, thus requiring substantial hardware over-provision compared to the results of *probabilistic schedulability analysis*[1] expressed in terms of probabilities or probability distributions.

In the case of soft real-time systems, failure to meet a deadline and the consequent late response represents a degradation in the quality of service provided or the utility of the results produced. Here, probabilistic schedulability analysis can be used to characterise the expected quality of service that the system will provide.

This survey classifies and reviews probabilistic schedulability analysis techniques for real-time systems, where one or more task parameters are modelled as random variables, i.e. via probability distributions. Probabilistic schedulability analysis aims to determine for a set of tasks with parameters described by probability distributions, scheduled according to a given policy (for example fixed priority preemptive scheduling or Earliest Deadline First (EDF)) if those tasks can be guaranteed to meet their timing requirements, described in terms of *probabilistic deadlines* (i.e. deadlines with associated thresholds on the maximum acceptable probability of a deadline miss), or to simply compute the probability of deadline failure.

Much of the literature on probabilistic schedulability analysis models task execution times via probability distributions, while other works consider probabilistic Worst-Case Execution Time (pWCET) distributions. The latter can be derived via probabilistic timing analysis, us-

[1] In this survey, we adopt the widely used term "probabilistic schedulability analysis" noting that it can easily be misinterpreted. To clarify, while the results produced are expressed in terms of probabilities or probability distributions, the analysis methods themselves are deterministic in the sense of always producing the same results from the same inputs, unlike, for example, randomised search techniques.

ing *analytical methods* referred to as *Static Probabilistic Timing Analysis (SPTA)* [35, 53, 11, 10, 85, 84] or *statistical methods* referred to as *Measurement-Based Probabilistic Timing Analysis (MBPTA)* [32, 58, 66, 44, 150, 135, 133, 88, 87]. Probabilistic timing analysis techniques are reviewed in detail in a companion survey [52].

Research into probabilistic schedulability analysis can be classified into eight main categories. This classification forms the basis for the main sections of this survey. Note, for ease of reference we have numbered the categories to match the sections, starting at 3.

3. *Probabilistic Response Time Analysis*: these methods compute the probability distribution of the response time of each task, or the jobs of each task. These response time distributions can then be compared to the deadlines to determine if the timing requirements are met.
4. *Probabilistic Analysis assuming Servers*: these methods assume that each task is allocated a proportion of the processor bandwidth via a server. This has the advantage of isolating the tasks from interfering with one another, which simplifies the schedulability analysis.
5. *Real-Time Queuing Theory*: these methods estimate the fraction of deadlines that will be met from queue length dependent *lead-time profiles*. These profiles describe the time to go to the deadline and the distribution of queue lengths obtained via queueing theory.
6. *Probabilities from Faults*: works in this category assume additional execution time or load on the system from fault recovery operations as a consequence of faults that occur according to some probability distribution.
7. *Statistical Analysis of Response Times*: works in this category differ from those above in that they use statistical techniques based on observations of response times to predict the probability of deadline failure.
8. *Probabilistic Analysis of Mixed Criticality Systems*: these methods analyse mixed criticality systems using a richer representation (e.g. execution times described by probability distributions) rather than discrete WCET estimates for different levels of criticality.
9. *Miscellaneous*: research in this category explores different aspects of scheduling probabilistic real-time systems, including tasks with precedence constraints, the imprecise computational model, randomised job dropping, priority assignment policies, component based scheduling, and multiprocessor scheduling.
10. *Addressing Issues of Intractability*: the methods reviewed in this section aim to significantly reduce the runtime of probabilistic schedulability analysis techniques.

The research in these categories is summarised by authors and citations in Table 1. Note the sub-categories correspond to the subsections of this survey.

It is interesting to note how research in the different categories has progressed over time. Figure 1 illustrates the number of papers reviewed in each of the main categories covered by this survey that were published in 3-year time intervals from 1988 to 2018. (This figure is best viewed online in colour). A number of observations can be drawn from Figure 1. Firstly, the volume of research into probabilistic schedulability analysis has greatly increased since its origins in the late 1980s / early 1990s. The number of publications on the main theme of probabilistic response time analysis (Section 3) has steadily increased during this time. Work on server-based analysis and real-time queueing theory (Sections 4 and 5) have produced a small number of papers over most of the time period. By contrast, work on probabilities derived from faults (Section 6) began in 1999, with a peak in 2003 - 2006, and fewer publications in recent years, Categories of more recent interest (i.e. publications since 2007) include statistical analysis (Section 7) and addressing issues of tractability (Section 10). However, the hot topic in recent years, albeit with mainly preliminary publications, is work on probabilistic approaches to mixed criticality systems (Section 8). This area has shown a rapid expansion in the number of publications since 2015.

Before moving to the sections of this survey which review the literature, we first discuss (in Section 2) fundamental concepts and issues pertaining to probabilistic schedulability analysis.

Table 1 Summary of publications from different authors in the categories described in the main sections and subsections of this survey.

3 Probabilistic Response Time Analysis
3.1 Analysis for Periodic Tasks with No Backlog Woodbury and Shin [152], Tia et al. [143], Atlas and Bestavros [13], Gardner et al. [60], Tanasa et al. [141]
3.2 Analysis for Periodic Tasks with Backlog Diaz et al. [54, 55], Lopez et al. [93], Kim et al. [76], Ivers and Ernst [70], Tanasa et al. [140]
3.3 Analysis for More Complex Task Models Cucu-Grosjean and Tovar [41], Cucu-Grosjean [40], Maxim et al. [105], Maxim and Cucu-Grosjean [106], Maxim and Bertout [104], Santinelli and Cucu-Grosjean [130, 131], Santinelli et al. [136], Khan et al. [74], Santinelli [129], Carnevali et al. [34], Ben-Amor et al. [23], Markovic et al. [103]
4 Probabilistic Analysis assuming Servers
4.1 Analysis for Server-based Systems Abeni and Buttazzo [2, 3, 4] , Kaczynski et al. [72], Manica et al. [99], Abeni et al. [6, 5], Palopoli et al. [122, 120, 121], Frias et al. [59, 146]
5 Real-Time Queueing Theory
5.1 Analysis based on Real-Time Queuing Theory Barrer [20], Panwar et al. [123], Lehoczky [83], Doytchinov et al. [56], Hansen et al. [67], Zhu et al. [154], Gromoll and Kruk [63], Kruk at al. [79]
6 Probabilities from Faults
6.1 Analysis of Fault Recovery on Processors Burns et al. [33, 30], Broster and Burns [27, 26], Kim and Kim [75], Aysan et al. [18], Short and Proenza [138], Santinelli et al. [134]
6.2 Analysis of Fault Recovery on CAN Navet et al. [117], Broster et al. [28, 29], Aysan et al. [19] Axer et al. [17] , Davis and Burns [48], Nolte et al. [119], Zeng et al. [153]
7 Statistical Analysis of Response Times
7.1 Statistical Estimation Navet et al.[116], Lu et al. [96, 97, 94, 95], Liu et al. [91], Maxim et al. [111]
8 Probabilistic Analysis of Mixed Criticality Systems
8.1 Analysis for Mixed Criticality Systems Santinelli and George [132], Guo et al. [64], Maxim et al. [107, 108], Alahmad and Gopalakrishnan [9, 8], Draskovic et al. [57], Abdeddaim and Maxim [1], Kuttler et al. [80]
9 Miscellaneous
9.1 Task Graphs and Precedence Constraints Manolache et al. [100, 101, 102], Hua et al. [69]
9.2 Multiprocessor Analysis Nissanke et al. [118], Leulseged and Nissanke [86], Mills and Anderson [113, 112], Liu et al.[92], Wang et al. [149, 148] , Ren et al. [128]
9.3 Miscellaneous Models and Techniques Hu et al. [68], Hamann et al. [65], Kim et al. [77], Gopalakrishnan [61]
9.4 Position Papers Quinton et al. [125], Cucu-Grosjean [42, 43]
10 Addressing Issues of Intractability
10.1 Re-sampling Refaat et al. [127], Maxim et al. [110, 109], Milutinovic et al. [114]
10.2 Analytical Methods and Other Techniques Chen and Chen [36] von der Bruggen et al. [147], Chen et al. [37]

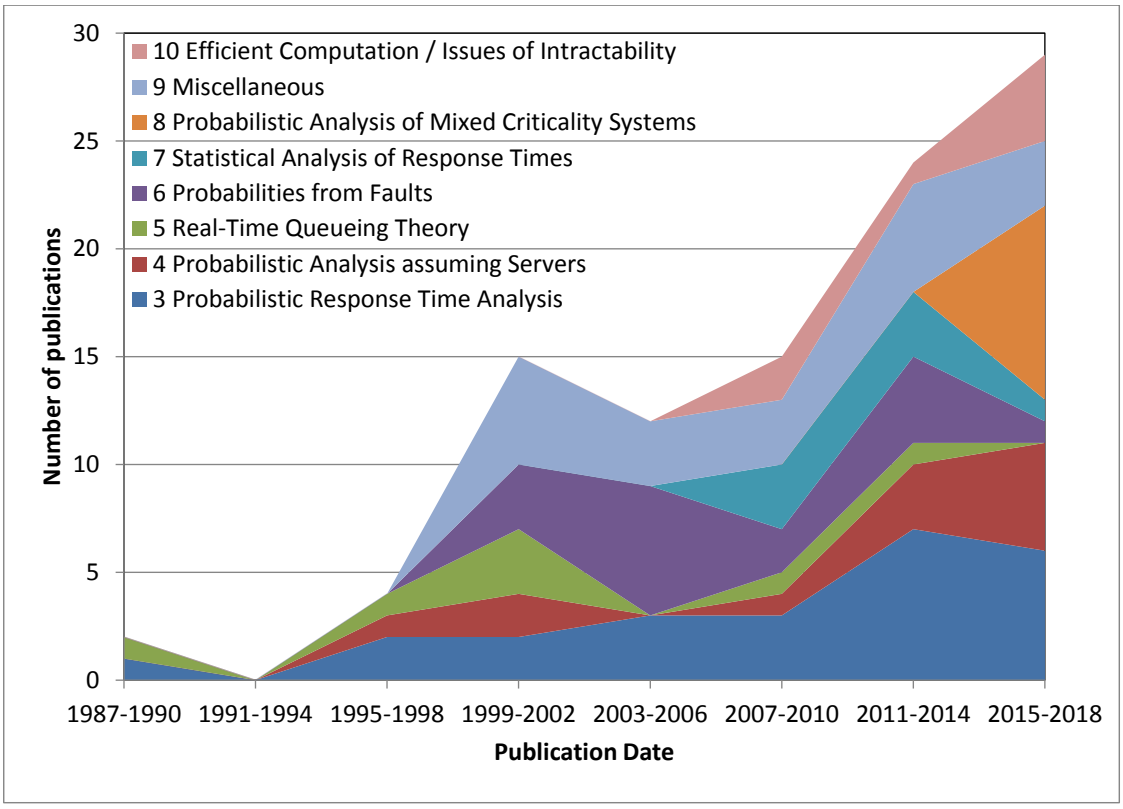


Figure 1 Intensity of research in the different categories corresponding to Sections 3 to 10 of this survey.

Note that conventional schedulability analysis techniques aimed at systems where task parameters are specified by single values rather than probability distributions are outside of the scope of this survey, they are reviewed in detail in a number of prior works [137, 49, 45].

2 Fundamentals and Key Issues

The term *probabilistic real-time systems* is used to refer to real-time systems where one or more of the task parameters (e.g. execution time, period, etc.) are described by random variables. Although a parameter, such as the execution time of a task, is described i.e. *modelled* by a random variable, this does not necessarily mean that the actual parameter itself exhibits random behaviour or that there is necessarily any underlying random element to the system that determines its behaviour. The actual behaviour of the parameter may depend on complex and unknown or uncertain behaviours of the overall system. As an example, the outcome of a coin toss can be modelled as a random variable with heads and tails each having a probability of 0.5 of occurring, assuming that the coin is fair. However, the actual process of tossing a coin does not actually have a random element to it. The outcome could in theory be predicted to a high degree of accuracy if there were sufficiently precise information available about the initial state and the complex behaviour and evolution of the overall physical processes involved. There are, however, many useful results that can be obtained by modelling the outcome of a coin toss as a random variable. The same is true in the analysis of probabilistic real-time systems.

In this survey we use calligraphic characters to denote random variables. As an example, the discrete probability distribution of the execution time $\mathcal{X}$ of a task may be described by a *Probability Mass Function* (PMF)

$$\mathcal{X} = \begin{pmatrix} 1 & 2 & 4 \\ 0.85 & 0.1 & 0.05 \end{pmatrix} \tag{1}$$

indicating that there is a probability of 0.85 that the execution time will be 1, a probability of 0.1 that it will be 2, and a probability of 0.05 that it will be 4. Thus the *Cumulative Distribution Function* (CDF) is given by:

$$F_{\mathcal{X}}(x) = P(\mathcal{X} \leq x) = \begin{cases} 0 & \text{if } x = 0 \\ 0.85 & \text{if } x = 1 \\ 0.95 & \text{if } x = 2 \\ 0.95 & \text{if } x = 3 \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

Further, the *Complementary Cumulative Distribution Function* (1-CDF) or *Exceedance Function* is given by:

$$\bar{F}_{\mathcal{X}}(x) = P(\mathcal{X} > x) \begin{cases} 1 & \text{if } x = 0 \\ 0.15 & \text{if } x = 1 \\ 0.05 & \text{if } x = 2 \\ 0.05 & \text{if } x = 3 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Timing requirements in probabilistic real-time systems are typically specified in terms of *probabilistic deadlines*. In contrast to conventional timing requirements where only a relative deadline is specified, probabilistic timing requirements also prescribe a limit or threshold on the maximum acceptable probability that the deadline will be missed. Extending the simple example above, let us assume that the task has a period of 10 and a deadline of 3, and is the only task in the system. In that case it would have a *Deadline Miss Probability* of 0.05 (which can be obtained directly from the 1 - CDF) and would therefore be viewed as schedulable if the threshold on the probability of deadline failure were specified as 0.1.

In the above example, with a single task, schedulability analysis deriving the probabilistic response time distribution is trivial, since the response time distribution is the same as the execution time distribution. However, once multiple tasks are considered, then execution time distributions need to be combined in some way to form a probabilistic response time distribution, and this gives rise to issues relating to *independence*.

2.1 Independence

▶ **Definition 1.** Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are *probabilistically independent* if they describe two events such that knowledge of whether one event did or did not occur does not change the probability that the other event occurs. Stated otherwise, the joint probability is equal to the product of their probabilities $P(\{\mathcal{X} = x\} \cap \{\mathcal{Y} = y\}) = P(\mathcal{X} = x) \cdot P(\mathcal{Y} = y)$.

In our context this means that the execution times of two jobs of the same or different tasks are *independent* if the *event* that the execution time of the first job takes a particular value x has no effect on the probability that the execution time of the second job will take some value y. If this is not the case, then the execution times of the two jobs are said to be *dependent*.

Much of the literature on probabilistic schedulability analysis assumes that the random variables describing the execution times of jobs of the same or different tasks are independent. When the assumption of independence holds, then the *basic convolution* operator $\otimes$ can be used to determine the sum $\mathcal{Z} = \mathcal{X} \otimes \mathcal{Y}$ of the independent random variables $\mathcal{X}$ and $\mathcal{Y}$ where:

$$P\{\mathcal{Z} = z\} = \sum_{k=-\infty}^{k=+\infty} P\{\mathcal{X} = k\}P\{\mathcal{Y} = z - k\} \tag{4}$$

For example, if both $\mathcal{X}$ and $\mathcal{Y}$ are given by $\begin{pmatrix} 2 & 10 \\ 0.6 & 0.4 \end{pmatrix}$, then we have:

$$\begin{pmatrix} 2 & 10 \\ 0.6 & 0.4 \end{pmatrix} \otimes \begin{pmatrix} 2 & 10 \\ 0.6 & 0.4 \end{pmatrix} = \begin{pmatrix} 4 & 12 & 20 \\ 0.36 & 0.48 & 0.16 \end{pmatrix} \tag{5}$$

Issues of dependence are of great importance in probabilistic schedulability analysis, since an assumption of independence when it does not in fact exist can easily result in the computation of unsound[2] i.e. optimistic probabilities of a deadline miss. As an example, adapted from the work of Ivers and Ernst [70], consider how the execution time distributions $\mathcal{X}$ and $\mathcal{Y}$ for the jobs of two tasks may be combined to obtain a response time distribution. Here, we assume that task τ_X has the higher priority and runs first and we are interested in the response time of task τ_Y which is released at the same time as task τ_X, but runs once task τ_X has completed. If the execution times of the two jobs are independent, then the response time distribution may be obtained via convolution as given in (5) above. Thus assuming a deadline of 10 the probability of a deadline miss is 0.64, whereas with a deadline of 15 the probability of a deadline miss would be 0.16. If the execution times are instead perfectly positively correlated, then whenever task τ_X executes for 2 time units then so does task τ_Y, and similarly if τ_X executes for 10 time units, then so does task τ_Y. In this case, the response time can only take two values: 4 and 20, with probabilities of 0.6 and 0.4 respectively. Thus, compared to the independent case, the probability of missing a deadline of 15 is increased to 0.4, whereas the probability of missing a deadline of 10 is decreased to 0.6. If instead the execution times are perfectly negatively correlated, then whenever task τ_X executes for 2 time units then task τ_Y executes for 10 time units, and vice-versa. In this case the response time is always 12. Interestingly, while the probability of missing a deadline of 15 is reduced to zero, missing a deadline of 10 is now certain; it has a probability of 1, which is higher than in the case of either independence or positive correlation. This simple example serves to illustrate that the correctness of probabilistic schedulability analysis, and the response time distributions produced, crucially hinges upon the dependences between task execution times. The difficulty is that in practice the actual execution times of jobs of the same or different tasks are unlikely to be independent, rather they may exhibit correlation emanating from history dependent software states (e.g. static local variables) or history dependent hardware states (e.g. shared cache lines within the memory hierarchy), and also from dependences due to common or correlated input values that change only slowly over time.

2.2 pWCET Distributions

Some recent works on probabilistic schedulability analysis assume that the execution time behaviour of tasks is characterised by a probabilistic Worst-Case Execution Time (pWCET) distribution.

[2] In this survey, we use the adjective *sound* to indicate a description, an analysis, or a probability distribution that provides information about the system that is not optimistic with respect to its timing behaviour. Thus the information provided may be precise, or it may be pessimistic.

This term has been used widely in the literature, with a number of different definitions given. Below, we provide an overarching definition of the pWCET distribution.

▶ **Definition 2.** The *probabilistic Worst-Case Execution Time (pWCET)* distribution for a task is the least upper bound, in the sense of the greater than or equal to operator $\succeq$ (defined below), on the execution time distribution of the jobs of the task for every valid scenario of operation, where a *scenario* of operation is defined as an infinitely repeating sequence of input states (including both input values and software state variables) and initial hardware states that characterise a feasible way in which recurrent execution of the task may occur.

▶ **Definition 3.** (From Diaz et al. [55]) The probability distribution of a random variable $\mathcal{X}$ is *greater than or equal to* (i.e. upper bounds) that of another random variable $\mathcal{Y}$ (denoted by $\mathcal{X} \succeq \mathcal{Y}$) if the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is never above that of $\mathcal{Y}$, or alternatively, the 1-CDF of $\mathcal{X}$ is never below that of $\mathcal{Y}$. Similarly, the probability distribution of a random variable $\mathcal{X}$ is *less than or equal to* (i.e. lower bounds) that of another random variable $\mathcal{Y}$ (denoted by $\mathcal{X} \preceq \mathcal{Y}$) if the Cumulative Distribution Function (CDF) of $\mathcal{X}$ is never below that of $\mathcal{Y}$, or alternatively, the 1-CDF of $\mathcal{X}$ is never above that of $\mathcal{Y}$.

Graphically, Definition 2 means that the 1 - CDF of the pWCET distribution is never below that of the execution time distribution for any specific scenario of operation. Hence the 1 - CDF or *exceedance function* of the pWCET distribution may be used to determine an upper bound on the probability p that the execution time of a randomly selected job of the task will exceed an execution time budget x, for any chosen value of x. This upper bound is valid for any feasible scenario of operation.

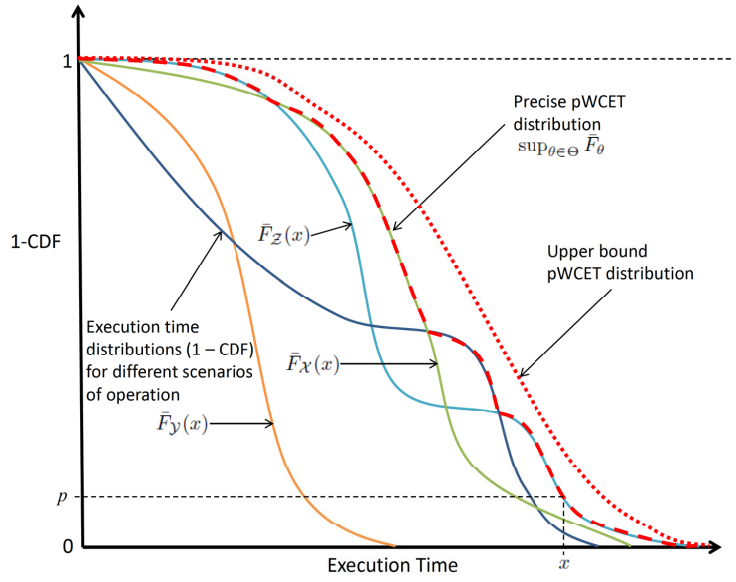


Figure 2 Exceedance function or 1-CDF for the pWCET distribution of a task, and also execution time distributions for specific scenarios of operation.

Figure 2 illustrates the execution time distributions for a number of different scenarios of operation (solid lines), the precise pWCET distribution (red dashed line) which is the least upper bound (i.e. the point-wise maxima of the 1 - CDF) for all of these distributions, and also some

arbitrary upper bound pWCET distribution (red dotted line) which is a pessimistic estimate of the precise pWCET. Also shown (on the y-axis) is an upper bound p on the probability that any randomly selected job of the task will have an execution time that exceeds x (on the x-axis). The value x is referred to as the pWCET estimate at a probability of exceedance of p. (More formally, the precise upper bound pWCET distribution is given by $\sup_{\theta \in \Theta} \bar{F}_\theta$ where $\bar{F}_\theta$ is the 1 - CDF for scenario of operation θ, and Θ is the space of all valid scenarios of operation).

Note that the greater than or equal to relation $\succeq$ between two random variables does not provide a total order, i.e. for two random variables $\mathcal{X}$ and $\mathcal{Z}$ it is possible that $\mathcal{X} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq X$. Hence the precise pWCET distribution may not correspond to the execution time distribution for any specific scenario. This can be seen in Figure 2, considering the execution time distributions $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$. It is the case that $\mathcal{X} \succeq \mathcal{Y}$, but $\mathcal{X} \not\succeq \mathcal{Z}$ and $\mathcal{Z} \not\succeq X$.

We note that the term pWCET is open to misinterpretation and is often misunderstood. To clarify, it does *not* refer to the probability distribution of the worst-case execution time, since the WCET is a single value. Rather informally, following Definition 2, the pWCET may be thought of as the "worst-case" (in the sense of upper bound) probability distribution of the execution time for any scenario of operation.

It is interesting to consider the use and interpretation of the pWCET distribution for a task. Let us assume that the task will be run repeatedly a potentially unbounded number of times, and that a fixed execution time budget of x applies to each run. Further, we assume that this budget is enforced by the operating system, and therefore that any job of the task that has not completed within an execution time of x is terminated and assumed to have failed. The pWCET distribution provides the following information, by reading off the probability of exceedance p associated with the execution time budget x (see Figure 2):

(i) An upper bound p on the probability (with a long-run frequency interpretation) equating to the number of jobs expected to exceed the execution time budget divided by the total number of jobs in a long (tending to infinite) time interval.

(ii) An upper bound p on the probability that the execution time budget will be exceeded by a randomly selected job. (This is broadly equivalent to the above long-run frequency interpretation).

Contrast this with the binary information provided by the WCET. If x is greater than or equal to the WCET, then we can expect the budget to never be exceeded. If x is less than the WCET, then we expect the budget to be exceeded on some runs, but we have no information on how frequently this may occur.

We note that some researchers have interpreted the pWCET distribution as giving the probability or confidence $(1-p)$ that the WCET does not exceed some value x. This interpretation can be confusing, since the meaning of the WCET is normally taken to be the largest possible execution time that could be realised on any single job of the task. Instead, in line with Definition 2 and (i) above, we view the 1 - CDF of the pWCET distribution as providing, for any chosen value x for the execution time budget, an associated upper bound probability p (with a long-run frequency interpretation) equating to the number of jobs of the task expected to exceed the execution time budget x, divided by the total number of jobs of the task executing in a long time interval.

2.3 pWCET distributions and dependences

There are two main ways of obtaining pWCET distributions: via *Static Probabilistic Timing Analysis (SPTA)* or via *Measurement-Based Probabilistic Timing Analysis (MBPTA)*. (A detailed discussion and review of these methods is given in the companion survey [52] on probabilistic timing analysis).

The aim of SPTA methods is to *construct* an upper bound on the pWCET distribution of a task by applying static analysis techniques to the code, supplemented by information about input values, along with an abstract model of the hardware behaviour. For static analysis to produce a non-degenerate[3] pWCET distribution there typically has to be some part of the system or its environment that contributes random or probabilistic timing behaviour. SPTA methods for tasks running on time-randomised hardware (e.g. with a random replacement cache) effectively consider each path through the code. For each path, these methods construct a pWCET distribution that upper bounds the probability distribution of the execution time for that path, considering all possible initial hardware states and all possible input states that cause execution of the path. The upper bound pWCET distributions for every path are then combined using an envelope function (taking the point-wise maximum over the 1-CDFs) to determine an upper bound on the pWCET distribution for the task that is valid *independent* of the path taken. (More sophisticated SPTA methods analyse sub-paths and use appropriate join operations at path convergence to compute tighter upper bounds on the pWCET distribution of the task).

The upper bound pWCET distribution for a task derived by SPTA (as described above) upper bounds the pWCET distributions for every path through the code. Similarly, the pWCET distribution for each path upper bounds the execution time distribution for every input state that drives that path, and every initial hardware state that could occur at the start of that path. Hence, by *construction* the pWCET distribution derived by SPTA is *probabilistically independent* of the input state chosen or the path taken[4]. This has implications for the use that can be made of the pWCET distribution. Firstly, it can be used to bound the behaviour of any randomly selected job of the task. Secondly, since the pWCET distribution is independent of the input state, which may have strong dependences and correlations with the input states for previous jobs, it can be composed using basic convolution to upper bound the execution time behaviour of multiple jobs in a sequence. It can therefore be used to derive probabilistic worst-case response time (pWCRT) distributions, which can then be compared to deadlines to determine the probability of a deadline miss.

We note that the actual execution times for a sequence of jobs of a task, which exercise the same or different paths, may well show strong correlations and dependences. It is the *modelling* of the execution times via an appropriate pWCET distribution which enables probabilistic independence to be assumed. (This is similar to the conventional case of a single WCET which can similarly be used in this way, even though the actual execution times of different jobs have strong dependences).

The aim of Measurement-Based Probabilistic Timing Analysis (MBPTA) methods is to make a *statistical estimate* of the pWCET distribution of a task. This estimate is derived from a sample of execution time observations obtained by executing the task on the actual hardware or on a cycle-accurate simulator[5] according to an appropriate *measurement protocol*. The measurement protocol executes the task multiple times according to some sequence(s) of feasible input states and initial hardware states, thus sampling one or more possible scenarios of operation.

Provided that the sample of execution time observations passes appropriate statistical tests, then *Extreme Value Theory* (EVT) [39] can be used to derive a statistically valid estimate of the probability distribution of the *extreme values* of the execution time distribution, i.e. to estimate the pWCET distribution. By modelling the shape of the distribution of the extreme execution times, EVT is able to predict the probability of occurrence of execution time values that exceed

[3] A degenerate distribution has only a single possible value.

[4] This holds provided that the random values generated, for example by the random number generator within a random replacement cache, are also independent.

[5] A cycle-accurate simulator provides the same timing behaviour as the actual hardware, accurate to a single processor clock cycle.

any that have been observed. (For a basic introduction to the use of EVT in this context, see the companion survey [52] on probabilistic timing analysis. More detailed information about EVT can be found in Stuart Coles' textbook on the subject [39]).

▶ **Definition 4.** A sample of input states and initial hardware states used for analysis is *representative* of the population of states that may occur during a future scenario of operation if the property of interest (i.e. the pWCET distribution) derived from the sample of states used for analysis matches or upper bounds the property that would be obtained from the population of states that occur during the entire scenario of operation.

▶ **Definition 5.** As determined by MBPTA, the estimated pWCET distribution for a task (or path through a task) is a statistical estimate of the probability distribution of the *extreme values* of the execution time of that task (or path), valid for any future scenarios of operation that are properly represented by the sample of input states and initial hardware states used in the analysis.

Ideally, MBPTA would provide a pWCET distribution that is valid for any of the many possible future scenarios of operation; however, an important issue here is that there may not be one single distribution of input states and hardware states that is representative of all possible future scenarios of operation. This issue of *representativity* is a key open problem in research on the practical use of MBPTA, see Section 2.3 of the companion survey [52] on probabilistic timing analysis for a further discussion of this issue.

There are two main ways of applying MBPTA, referred to as *per-path* and *per-program*:

1. *Per-path:* MBPTA is applied at the level of paths. A measurement protocol is used to exercise all feasible paths through the program, then the execution time observations are divided into separate samples according to the path that was executed. EVT is then used to estimate the pWCET distribution for each path. The pWCET distribution for the program as a whole is then estimated by taking an upper bound (an envelope or point wise maxima on the 1 - CDFs) over the set of pWCET estimates for all paths.
2. *Per-program (or task):* MBPTA is applied at the level of the program i.e. the task. A measurement protocol is again used to exercise all paths. In this case, all of the execution time observations are grouped together into a single sample. EVT is then applied to that sample, thus estimating directly the pWCET distribution for the entire program.

With MBPTA, if the per-path approach is used, and it is known that the execution times for each path vary *only* due to random elements in the hardware, and do not otherwise vary across different input states that are in the same equivalence class (i.e. that drive the same path), then probabilistic independence of the pWCET distribution is assured. (Recall that with the per-path approach, the pWCET estimate for the task is an upper bound over the pWCET distributions for all of its paths). Probabilistic independence of the pWCET distribution means that it can be used to characterise the behaviour of any randomly selected job of the task, and also composed using basic convolution to upper bound the interference from multiple jobs in probabilistic schedulability analysis.

In other cases, the execution times obtained for a path may be dependent on the particular input states used to drive it. These input states may themselves exhibit dependences and correlations. Further, with the per-program approach, there may be dependences and correlations between the paths taken on consecutive jobs of the task, as happens with a software state machine. For systems with such dependences, then, assuming that the set of input states used during analysis is representative of those occurring during operation, the estimated pWCET distribution will still be valid in terms of characterising the extreme execution time behaviour of a *randomly* chosen single job of the task. However, it will typically *not* be valid to compose the pWCET distributions

using basic convolution. The reason is that the pWCET distribution is not necessarily a valid estimate for the extreme execution time values of a job *conditional* on specific execution times having occurred for previous jobs.

As an example, consider a program E that implements a software state machine with four states and hence four paths that runs on time-randomised hardware. Here the main factor which affects the execution time of the program is the path taken, which is determined by the value of a software state variable. For this program, all valid scenarios of operation involve the software state variable cycling through its four possible values in order, and hence the four possible paths executing in order on any four consecutive runs of the program. Further, assume that there is some variability in the execution time of each path due to an independent random element in the hardware (e.g. a random number generator), which contributes either 5 or 10 time units to the execution time for the path, with a probability of 0.5 in each case. The execution times of the four paths are (i) 10 or 15, (ii) 20 or 25, (iii) 30 or 35, and (iv) 40 or 45. Note, each path has a probability of 0.25 of being taken in a randomly chosen execution of the program.

The precise pWCET distribution valid for any scenario of operation is:

$$pWCET_{per-program} = \begin{pmatrix} 0 & 10 & 15 & 20 & 25 & 30 & 35 & 40 & 45 \\ 1 & 0.875 & 0.75 & 0.625 & 0.5 & 0.375 & 0.25 & 0.125 & 0 \end{pmatrix}$$

Using the per-program approach, MBPTA may tightly upper bound this distribution. While using the per-path approach, the pWCET distribution obtained would tightly upper bound the following distribution for the longest path.

$$pWCET_{per-path} = \begin{pmatrix} 0 & 40 & 45 \\ 1 & 0.5 & 0 \end{pmatrix}$$

We note that while the $pWCET_{per-program}$ distribution is valid to describe the execution time behaviour of a randomly chosen job of the task, convolution of this distribution is not valid to describe the overall execution time of two or more jobs. This is because the distribution is not valid conditional on the execution times observed for previous jobs. For example, if an execution time of 30 or 35 is observed for a specific job, then the execution time of the next job will necessarily be either 40 or 45 (each with a probability of 0.5) due to the behaviour of the software state machine. The probability that the total execution time of two consecutive jobs exceeds 70 is therefore 0.1875, whereas the value computed by applying convolution to the $pWCET_{per-program}$ distribution is 0.15625, which is optimistic due to an incorrect assumption of independence. In this simple example optimism can be avoided by using the $pWCET_{per-path}$ distribution, which would also be obtained by SPTA; however, this discards information and so gives a pessimistic result, indicating that the probability of the total execution time of two consecutive jobs exceeding 70 is 1.

2.4 Probabilistic Inter-arrival Times

As well as probabilistic execution times and pWCETs, a few works on probabilistic schedulability analysis have also considered tasks with inter-arrival times characterised by random variables (see Section 4.1) or probabilistic Minimum Inter-arrival Times (pMIT) (see Section 3.3).

▶ **Definition 6.** The *probabilistic Minimum Inter-arrival Time (pMIT) distribution* for a task is the greatest lower bound, in the sense of the less than or equal to operator $\preceq$ (see Definition 3), on the inter-arrival time distribution of the jobs of the task for every valid scenario of operation, where a *scenario* of operation is defined as an infinitely repeating sequence of job arrivals that characterise a feasible way in which recurrent execution of the task may occur.

For example, the pMIT of a task τ_i may be described by $\mathcal{T}_i = \begin{pmatrix} 5 & 10 \\ 0.2 & 0.8 \end{pmatrix}$ meaning that the probabilistic minimum inter-arrival time is lower bounded such that a job of τ_i has a probability of 0.8 of arriving 10 or more time units after the previous job, and a probability of 1.0 of arriving 5 or more time units after the previous job.

The pMIT distribution provides the following information, by reading off the probability of exceedance p (from the 1 - CDF) associated with a limit t on the inter-arrival time:

(i) An upper bound p on the probability (with a long-run frequency interpretation) equating to the number of jobs that are expected to have an inter-arrival time of less than t, divided by the total number of jobs in a long (tending to infinite) time interval.

(ii) An upper bound p on the probability that the inter-arrival time of a randomly selected job will be less than t. (This is broadly equivalent to the above long-run frequency interpretation).

This contrasts with the sporadic task model which simply assumes that the minimum inter-arrival time is bounded by some value T_i (which would be 5 in the above example), but gives no information about how often longer inter-arrival times may occur.

The majority of works surveyed that model inter-arrival times as random variables assume that the inter-arrival times between jobs of the same task and between consecutive jobs of one task and consecutive jobs of another task are *independent*. For example, one practical application described by Maxim and Cucu-Grosjean [106] is vehicle reverse parking systems. Here, the inter-arrival time of the sensing task is randomised to avoid the possibility of systematic and repeated interference between the parking sensors of two vehicles that are reversing towards each other.

We note, however, that dependences are easily possible as a consequence of particular implementations. For example, the inter-arrival times of a task could follow a fixed pattern due to the operation of a software state machine which sets the next arrival time. This would give a fixed but repeating sequence, such as $(5, 10, 5, 10, \ldots)$. Further, two tasks could have inter-arrival times that are generated by a simple algorithm from the output values of a random number generator. They would have pMITs that are easily found from the properties of the random number generator and the algorithm used. In this case, the inter-arrival times of jobs of the same task would be independent, but the inter-arrival times between consecutive jobs of the two tasks would be in lock step i.e. correlated and dependent.

As with execution times and pWCETs, work on probabilistic schedulability analysis that considers tasks with inter-arrival times or pMITs characterised by random variables must take great care to either ensure that arrival times are independent or to handle dependences correctly.

2.5 Probabilistic Real-Time Constraints

In classical real-time systems, timing constraints are typically specified in terms of task deadlines. The relative deadline D_i of a task τ_i is inherited by all of it's jobs. The deadline D_i specifies the maximum elapsed time that is permitted from the release of a job of the task until that job completes its execution. The time from the release to the completion of a job is referred to as its *response time*. The worst-case response time R_i of a task τ_i is the longest possible response time of any of its jobs. If it can be proven, via schedulability analysis, that all jobs of a task will always complete by their deadlines, i.e. that $R_i \leq D_i$, then the task is said to be *schedulable*. If all the tasks in a system are schedulable, then the system itself is said to be schedulable.

Building upon these concepts, probabilistic real-time constraints are typically expressed in the form of *probabilistic deadlines* defined as follows:

▶ **Definition 7.** The *probabilistic deadline* of a task τ_i is given by the combination of a relative

deadline D_i with a single (deterministic) value and a *threshold* ρ_i specifying the maximum acceptable probability that the deadline may be exceeded.

The concept of a probabilistic deadline has, in a few works, been extended to relative deadlines that may take a number of different values each with an associated probability, expressed as a discrete random variable $\mathcal{D}_i$. Again, a *threshold* ρ_i specifies the maximum acceptable probability that the deadline may be exceeded. In this survey, we use the term *probabilistic deadline* to refer to the simpler form defined above with a single value for D_i, and make a clear distinction when describing work that considers deadlines as random variables.

In the classical view of hard real-time systems any deadline miss is sufficient to make a task, and hence the system unschedulable, thus the pattern and probability of deadline misses are not considered relevant. By contrast, in probabilistic real-time systems there are a number of different ways in which the probability of a deadline miss can be considered and interpreted:

1. As a probability with a long-run frequency interpretation equating to the expected number of missed deadlines divided by the total number of deadlines in a long (tending to infinite) time interval.
2. As the probability that a randomly selected job will miss its deadline, which is broadly equivalent to the long-run frequency interpretation.
3. As a bound on the probability that any specific job will miss its deadline.

These interpretations can be made considering the jobs belonging to (i) a specific task, (ii) a group of tasks that comprise a given application, or (iii) the system as a whole (i.e. all tasks). In the latter cases, there would be a threshold on the probability of deadline misses for each application, or a single threshold for the system as a whole; rather than thresholds for each task.

In the literature, the term *Deadline Miss Probability* (DMP) is often used to refer to the long-run frequency interpretation. This quantity is typically computed for task sets that are strictly periodic and thus have a behaviour which repeats after the hyperperiod or Least Common Multiple (LCM) of the task periods. In contrast, the term *Worst-Case Deadline Failure Probability* (WCDFP) is used to mean a bound on the probability that any specific job of a task will miss its deadline. This quantity is typically computed by reference to the probabilistic Worst-Case Response Time (pWCRT) of a task.

▶ **Definition 8.** The *probabilistic Response Time (pRT)* of a job $\tau_{i,j}$ of task τ_i, denoted by $\mathcal{R}_{i,j}$, describes the probability distribution of the response time of the j-th job of that task, indexed from the start of the hyperperiod.

▶ **Definition 9.** The *probabilistic Worst-Case Response Time (pWCRT)* of a task τ_i, denoted by $\mathcal{R}_i$, is an upper bound on the worst-case response time distribution for *any* job of the task.

Note the pWCRT of a task can be computed for both periodic and sporadic tasks by taking into account the worst-case arrival pattern.

If the tasks are strictly periodic (with a single value for their periods), then the deadline miss probability for a task can be computed by taking the average of the deadline miss probabilities of all of its jobs activated during a hyperperiod as follows:

▶ **Definition 10.** The *Deadline Miss Probability* DMP_i for a task τ_i is given by :

$$DMP_i = \frac{1}{n_{LCM}} \sum_{j=1}^{n_{LCM}} P(\mathcal{R}_{i,j} > D_i) \tag{6}$$

where n_{LCM} is the number of jobs of task τ_i activated during the hyperperiod.

▶ **Definition 11.** The *Worst-Case Deadline Failure Probability* $WCDFP_i$ for task τ_i is an upper bound on the probability that any single job of the task misses its deadline, computed directly from the pWCRT distribution $\mathcal{R}_i$ and the deadline D_i of the task as follows:

$$WCDFP_i = P(\mathcal{R}_i > D_i) \tag{7}$$

There are advantages and disadvantages to using the DMP and the WCDFP formulations. For strictly periodic task sets, it is possible to compute the DMP over the hyperperiod. (Note, this calculation becomes more complex if the task model permits a *backlog* of outstanding execution at the end of the hyperperiod, see Section 3.2). For sporadic task sets the DMP formulation is not viable. This is because the deadline miss probability for the periodic case does not provide an upper bound on that for the sporadic case. As an example, consider a system with two tasks with minimum inter-arrival times $T_1 = 10$ and $T_2 = 15$. Sporadic behaviour of task τ_1 which aligns every release of that task with a job of τ_2 may result in a larger deadline miss probability than strictly periodic behaviour with a period of 10. This can be seen by considering the degenerate case where both tasks have an execution time and a deadline of 1 with task τ_1 having the higher priority. With all releases synchronised, i.e. both periods set to 15, task τ_2 never meets its deadline, while if task τ_1 is released more frequently with a period of 10, and task τ_2 has a period of 15, then the deadline miss probability for task τ_2 becomes 0.5. Attempting to account for all possible phasings of jobs of sporadic tasks leads to problems of intractability, hence a different approach is needed.

The WCDFP formulation potentially introduces some pessimism, since the relationship between task periods means that not all jobs of a task may be subject to the maximum interference from other tasks; however, it provides a valid upper bound on the probability of deadline misses for systems where not all tasks have strictly periodic releases (i.e. for the sporadic task model).

Hard real-time systems in many application domains can in practice tolerate a small number of consecutive deadline misses, but cannot tolerate long black-out periods when no deadlines are met. In the literature on deterministic schedulability analysis these issues have been investigated in work on *weakly-hard real-time systems* [24, 25], *m-k firm deadlines* [126], *skip-over* [78] techniques, and *typical worst-case analysis* [7]. The problem of reconciling requirements on the length of potential black-out periods (i.e. the number of consecutive missed deadlines) and a probabilistic treatment of deadline misses has, as far as we are aware, received little attention in the literature.

There are a number of issues here, which reflect long-run versus short-run viewpoints and the underlying issue of *independence*. We illustrate these problems via a simple example. Assume that we have two periodic task systems A and B, both with a hyperperiod that equates to four jobs of the task that we are interested in analysing. Further, let the probability that each of these jobs misses its deadline be as follows: system A $(0.75, 0.75, 0.25, 0.25)$ and system B $(0.5, 0.5, 0.5, 0.5)$. For both systems, the DMP for the task is 0.5; however, for system A, the WCDFP is 0.75, whereas for system B it is 0.5. Further, in order to reason about the probability of a black-out period equating to two consecutive deadline misses, we need to know if the response time distributions for the jobs, and hence the probabilities of deadline misses are independent or if they are correlated in some way. The probability that the first two jobs in the hyperperiod of system A both miss their deadlines would be 0.5625 if independence is assumed; however, it could be as high as 0.75 if the response times of the two jobs are positively correlated. We note that the response times of consecutive jobs of the same task may fail to be independent as a direct result of their execution times being dependent (e.g. due to dependences on shared software or hardware states, shared or correlated input variables etc.) or as an indirect result of interference from jobs of another task that exhibit execution time dependences.

2.6 Summary

In this section we described the fundamental building blocks used in probabilistic schedulability analysis, namely execution time distributions, response time distributions, and probabilistic deadlines. We also discussed the key underlying assumption in much of the literature on this topic: the independence of task execution times or pWCET distributions. We return to this issue in the conclusions. The next eight sections review the existing literature on probabilistic schedulability analysis.

3 Probabilistic Response Time Analysis

In this section we review work on probabilistic response time analysis. Comparing the probability distribution of the response time of a job of a task to its deadline enables the probability of a deadline miss for the job to be determined. For a task set with periodic release times, considering all these values for each job of the task over the hyperperiod (Least Common Multiple of task periods) enables the deadline miss probability for the task to be computed. Alternatively, some works derive the probabilistic worst-case response time distribution with respect to any job of the task, thus directly providing an upper bound on the worst-case deadline failure probability valid for any job. This latter approach works for both periodic and sporadic task sets.

In the following subsections, we classify and review papers according to the underlying task models that they support.

3.1 Analysis for Periodic Tasks with No Backlog

Initial work on probabilistic response time analysis focused on a simple task model where: (i) all tasks are periodic, (ii) the execution times of the jobs of each task are independent of those of other jobs of the same or different tasks, (iii) there is no *backlog* at the end of the hyperperiod, i.e. the worst-case processor utilisation is ≤ 1 and so the processor is guaranteed to be idle at the end of the hyperperiod.

In 1988, Woodbury and Shin [152] introduced analysis that computes the deadline miss probability for periodic tasks. The analysis assumes that tasks can be described in terms of multiple paths, each of which has a single deterministic execution time and a probability of occurrence. Thus each job of a task effectively has a probability distribution for its execution time, composed from the information about the paths. The analysis iterates over the hyperperiod in steps equating to the Greatest Common Denominator of the task periods. It computes the execution time remaining from higher priority jobs at the start of each step, and the joint distribution of the execution time of the jobs released at the start of the step. This information is used to derive the response time distribution for each job in the hyperperiod, and hence the probability of deadline failure for each task over the entire hyperperiod (i.e. considering all of its jobs). The technique is suitable for scheduling policies where the priority of a job does not change between releases (i.e. fixed priority or EDF). We note that it is arguable whether meaningful analysis can be obtained by assuming that the probability of taking a given path is known, and that the paths taken by different jobs are independent. In practice, the path taken is typically determined by the inputs, which are unlikely to be independent, for example sensor values may change only slowly over time. Further, formulating the analysis on the basis of paths can lead to tractability issues, since the number of paths could in practice be very large.

Probabilistic Time-Demand Analysis (PTDA) for fixed priority preemptive scheduling was proposed by Tia et al. [143] in 1995, based on the time-demand analysis technique given for the simpler case of WCETs by Lehoczky et al. [81]. With PTDA, at each scheduling point the cumulative probability distribution is computed (via convolution) for all job releases up to that point. This enables a bound to be computed on the probability that the task can meet its deadline.

The authors also propose a *transform-task* method of scheduling where part of the execution time distribution of each task (up to some specified value) is treated as a simple periodic task and guaranteed by time-demand analysis [81]. The remaining part of the distribution is considered as an additional sporadic task and runs under a sporadic server [139]. An alternative approach using EDF and slack stealing is also considered.

Statistical Rate-Monotonic Scheduling (SRMS) was introduced in 1998 by Atlas and Bestavros [13]. This method assumes that when a job is released then its execution time becomes known. SRMS consists of two parts, (i) admission control and (ii) a fixed priority (rate-monotonic) scheduling algorithm. The basic idea is to aggregate the capacity available to execute jobs of a task over multiple periods, equating to the so called *superperiod* given by the period of the next lower priority task. Jobs of a task are admitted if they can be executed within the remaining budget for the superperiod, and there is time available to do so before the deadline of the job, taking into account higher priority interference. Admitted jobs are then scheduled according to fixed priorities with rate-monotonic priority assignment. The authors show how the probability of admission can be computed for each period of a task within its superperiod. This allows quality of service guarantees to be computed in terms of the probability that a randomly selected job of a task will meet its deadline over an arbitrarily long time interval. Aside from the independence assumptions, the main drawback of SRMS is the requirement that job execution times become known upon release. In practice this is unlikely to be the case, however, in some cases execution times could be estimated from the input values.

Stochastic Time-Demand Analysis (STDA) for fixed priority scheduling, was developed in 1999 by Gardner et al. [60]. They note an issue with the prior work by Tia et al. on PTDA [143] in that it is only valid if there is no backlog at the deadline of a task. The problem is similar to the classical case of fixed priority scheduling with arbitrary deadlines [82]. All jobs in a priority level-i busy period need to be considered, since the first job is not necessarily the one that exhibits the worst-case behaviour. The authors provide an analysis based on the priority level-i busy period and the backlog present at subsequent releases of each job. They remark that the worst-case pattern of releases might not necessarily be following synchronous release of all tasks. They explore this aspect by considering different phasing in simulation, but did not find any cases where the probability of deadline failure was higher for random phasing than in the synchronous case. The convolution operations required as part of STDA are also discussed. Here, the authors indicate that the Fourier transform may be used to reduce complexity from $O(N^2)$ to $O(NlogN)$ where N is the number of points in each distribution. To use a fast Fourier transform, the distributions must have the same number of points and the same sampling rate.

In 2015, Tanasa et al. [141] studied the problem of determining probabilistic worst-case response time (pWCRT) distributions and hence deadline miss probabilities for a set of periodic tasks with execution times described by random variables. This work differs from earlier publications in that it describes the distributions via continuous variables and tightly approximates them with polynomial functions. This enables an analytical approach to be taken to their integration. The authors provide methods to compute the response time distribution for each job in the hyperperiod. They also introduce a method to compute the joint distribution between two jobs i.e. the probability that the response time of job $k_1 \leq r_1$ AND the response time of job $k_2 \leq r_2$. The methods used have exponential complexity and are thus limited in their application. In the evaluation, the maximum utilisation is limited to 85% for the experiments examining the univariate distributions and to no more than 40% for the joint distributions, using small hyperperiods of 25 or so jobs. The authors highlight an interesting observation, that even when the execution time distributions are continuous, the response time distributions may be disjoint, containing gaps where there are values that cannot be obtained. This occurs as a result of an additional preemption from a higher priority task that has a minimum as well as a maximum execution time.

3.2 Analysis for Periodic Tasks with Backlog

In contrast to classical task models, task sets containing a number of tasks with execution times described by random variables can usefully have a total worst-case processor utilisation that exceeds 1. This means that there is a *backlog*, meaning outstanding task execution with a finite probability of occurrence, at the end of each hyperperiod. This backlog makes the analysis of probabilistic response times for each job in the hyperperiod much more complex. Diaz et al. addressed this problem in two seminal papers published in 2002 [54] and 2004 [55]. This work has since been built upon by a number of other authors. Note the papers surveyed in this section also assume independence between the execution times of jobs of the same and different tasks, with the exception of the work of Ivers and Ernst [70] which addresses the important issue of dependences.

In 2002, Diaz et al. [54] noted that prior work on PTDA [143] and SDTA [60] assumes that the worst-case occurs for a job in the first busy period following synchronous release; however, this is not necessarily correct when the worst-case processor utilisation exceeds 1, and may lead to an under estimation of the response time distributions. They show that the *backlog* at the start of each hyperperiod is stationary provided that the average utilisation is less than 1. As the backlog at the end of one hyperperiod depends only on the backlog at the start of the previous hyperperiod, it can be modelled as a Markov chain. The authors show how to compute the stationary backlog, and use this backlog via a method of convolution (adding the execution of preempting jobs) and *shrinking* (shifting the distribution left and then accumulating the probabilities for all negative values at zero) to compute the response time distributions for each job of a given task over the hyperperiod. (A useful illustrative example is given in later work by the same authors [55]). By taking the average of these distributions, the response time distribution for the task can be computed and hence a bound on its deadline miss probability determined. We note that the method requires that the release times of jobs are fixed (i.e. periodic rather than sporadic), and when the hyperperiod is large then finding the stationary backlog can be computationally very expensive.

Subsequent research into the same problem by Diaz et al. [55] in 2004 discussed the concept of pessimism in probabilistic analysis, and introduced the concept of *greater than or equal to* between random variables (See Section 2, Definition 3). The authors note that any approximations in the analysis, for example of response time distributions, must result in distributions that are *greater than or equal to* the precise distribution in order to ensure soundness. They prove various properties with respect to their analysis, including that approximating the backlog and execution time distributions with distributions that are greater than or equal to ($\succeq$) the precise distributions produces sound results, i.e. there is no under-approximation of response time distributions. We note that this concept is similar to the one of *sustainability* introduced later for deterministic schedulability analysis by Baruah and Burns [21].

Diaz et al. [55] highlighted and addressed three issues with their previous work [54]:

- **(i)** When the maximum utilisation exceeds 1, then the steady state backlog has an infinitely long tail.
- **(ii)** Starting with zero backlog and iterating over a number of hyperperiods, the backlog becomes closer and closer to the steady state backlog; however, the estimation remains optimistic.
- **(iii)** Probability distributions with a large number of points require a very large amount of memory and processing resource.

The problem of the infinite tail is solved via truncation and accumulation of the probabilities for larger values at ∞. A solution to the problem of a large number of points is suggested, accumulating values to the right, effectively a sound form of re-sampling (see Section 10.1 for later work in this area). The authors also show how blocking due to shared resource accesses

can be accommodated as an extension to the task model by taking a *supremum* over all of the distributions for the execution time of relevant resource accesses. They also provide a sketch proof that the priority assignment algorithm of Audsley [15], which is optimal in the deterministic case, remains optimal when execution times are described by probability distributions and schedulability is defined as meeting the probabilistic deadline for each task. This was later confirmed by the work of Maxim et al. [105]. In 2008, Lopez et al. [93] extended the work of Diaz et al. [54, 55] providing: (i) a set of transformations that can be made to the parameters of a system which are guaranteed to result in a response time distribution greater than or equal to ($\succeq$) that for the original system; (ii) addressing issues related to the use of finite precision arithmetic; (iii) handling release jitter, by assuming the deterministic worst-case and its effect on the release times of jobs.

In 2005, Kim et al. [76] built upon the analysis framework set out by Diaz et al. [54, 55]. They discuss three solutions to obtaining the stationary backlog, an exact solution for the Markov matrix which has a high computational cost, and two approximate solutions. The first approximate solution truncates the matrix; however, the way in which truncation affects accuracy is unresolved. The second approximation involves iteratively computing the backlog over a number of hyperperiods, and examining its convergence. The number of iterations needed to provide a given level of accuracy is however unknown in advance. The evaluation shows that the deadline miss probability computed over the hyperperiod may be considerably smaller than that computed for jobs in the busy period following a critical instant, e.g. via SDTA [60]. The complexity of backlog computation is shown to be $O(j^2m^2)$ *per* job, where j is the number of the job and m is the number of points in the execution time distribution. In their conclusions the authors claim that the method could be indirectly applied to sporadic task systems by modeling them as a periodic system with periods equal to minimum inter-arrival times, and that this would give a safe approximation. We note that this assertion is not correct, as shown by the example given in Section 2.5.

The important problem of *dependences* between the execution times of jobs of the same task and jobs of different tasks was first addressed by Ivers and Ernst [70] in 2009. They presented an analysis that accounts for the effect of unknown *dependences* between the execution times of jobs of tasks in a system using fixed priority preemptive scheduling. The authors give a motivating example which illustrates how dependences can have a marked effect on the response time distribution. Their simple example, which we presented as an exemplar in Section 2, shows that considering a worst-case convolution (i.e. matching the largest values from each distribution) is in general not sufficient to determine the worst-case deadline miss probability. The authors introduce a method based on probability boxes which soundly bounds the response time distribution in the presence of unknown dependences. The approach uses the concept of *copulas* to model the relationship between marginal distributions and the joint distribution of two random variables, and Frechet bounds that give upper and lower bounds on the relationship between the marginal distributions and the joint distribution. Probabilistic bounds are then given on the *sum* of the two random variables. These bounds have been shown to be sound and point-wise tight [151]. The method of Diaz et al. [54, 55] is then lifted to probability boxes, enabling sound and tight bounds on the response time distributions to be computed for systems with unknown dependences between job execution times. Worked examples show that assuming independence when it does not exist can easily result in substantial optimism in the results.

In 2013, Tanasa et al. [140] provided a probabilistic analysis of the response times of messages transmitted via the dynamic segment of FlexRay. The system model assumes that messages are queued for transmission periodically, but are subject to variable queuing jitter, stemming from the task that queues them, and that this jitter may be modelled via some probability distribution. The method determines the deadline miss ratio for each message over the hyperperiod of message

periods and the FlexRay cycle. To achieve this it first computes the possible backlog vectors at the end of the time interval by constructing a transition graph linking the backlog vector at the start of the interval to those backlog vectors that can possible stem from it at the end of the interval. We note that this transition graph may be very large depending on the length of the hyperperiod, the number of messages, and the number of possibilities in terms of different values of jitter. The authors use an MILP formulation to compute the transition graph and the vectors. As a second step, the method computes the probability that message instances miss their deadlines on each transition. In the third step, the deadline miss probability is computed by iterating over a number of hyperperiods. This provides an under approximation as the backlog converges towards a stationary value (see the earlier work of Diaz et al. [54]). The method is computationally expensive and the authors implement part of it on a TeslaM2050GPU with 448 cores to help speed up processing.

3.3 Analysis for More Complex Task Models

Following initial work on periodic tasks, researchers have explored more complex task models. The majority of the research published in this area emanates from Cucu-Grosjean and co-authors. This includes work [41] on sporadic task models where inter-arrival times are described by random variables, and subsequently the development by Maxim and Cucu-Grosjean of analysis [106] for tasks with execution times, inter-arrival times and deadlines that are described by random variables. Further work by Maxim et al. [105] explored issues of priority assignment. Note all of the research described in this section assumes that task parameters such as execution times and inter-arrival time are independent.

The first work in this area, by Cucu-Grosjean and Tovar [41] in 2006, considered a model where tasks have constant execution times, but their inter-arrival times are modelled by random variables and have known probability distributions. The authors introduce a method of computing the probabilistic worst-case response time distribution for tasks under fixed priority preemptive scheduling. This model is useful for message streams where the message length is fixed, or varies very little, but the inter-arrival times may vary widely. In 2008, Cucu-Grosjean [40] applied the method to the problem of deriving response time distributions for CAN messages where message arrivals are assumed to be independent and described by probability distributions in a range between some minimum and maximum values. The analysis assumes that the maximum utilisation of the system is less than 1, thus avoiding issues relating to backlog.

In 2011, Maxim et al. [105] investigated three related problems of priority assignment in fixed priority preemptive systems where task execution times are described by random variables. They define the deadline miss probability for a job of a task, and the deadline miss probability[6] for a task, which is effectively the expected deadline miss probability over all of the jobs in some long interval of time, for example the hyperperiod. The three priority assignment problems involve:

(i) Finding a priority assignment that results in the deadline miss probability of each task being below the threshold specified for that task.

(ii) Finding a priority assignment that minimises the maximum deadline miss probability over all tasks.

(iii) Finding a priority assignment that minimises the average deadline miss probability over all tasks.

The authors give an example showing that rate-monotonic priority assignment is not optimal for problem (i) in the case of implicit deadline tasks. Optimal solutions to problems (i) and (ii) are

[6] Referred to as the deadline miss ratio in [105].

given using a greedy approach similar to the optimal priority assignment algorithm of Audsley [15]. A greedy approach is shown to be non-optimal for problem (iii), and a tree-based search is proposed as a potential solution.

In 2013, Maxim and Cucu-Grosjean [106] introduced *exact* probabilistic response time analysis for tasks which may have their worst-case execution times, inter-arrival times and deadlines described by independent random variables. The scheduler is assumed to be fixed priority preemptive. Deadlines are assumed to be constrained, i.e. not greater than the inter-arrival time to the next job of the same task. The authors show that for the task model considered, where incomplete tasks are aborted at their deadline, the worst-case response time distribution of any job of a task occurs for the first job in the synchronous busy period. The method presented is exponential in the number of tasks and the size of the random variables. They note that as the problem is a superset of the non-cyclic Generalised Multi-Frame (GMF) task model [115] there cannot be a pseudo-polynomial time exact test. Re-sampling techniques [109] are shown to be effective in reducing the complexity of the method in practice (see Section 10). The method iteratively computes the response time distribution by considering each preemption following synchronous release. For each preempting job, it convolves the pWCET distribution of that job with the tail of the current response time distribution from the preemption point onwards. The result is scaled by the probability of the job preempting at that time. This is repeated for all possible preemption times for the job, and the resulting distributions coalesced. Iteration then moves on to the next preempting job. Iteration ends when the next possible preemption is later than the final point in the current response time distribution, or the deadline of the task under analysis is exceeded. The probability of a deadline miss can then be found by subtracting the deadline distribution from the final response time distribution, with the probability mass strictly greater than zero giving the probability of a deadline miss. The analysis has been implemented in a publicly available simulator and analysis tool called PanSim [104]. Note, motivation for variable inter-arrival times comes from an automotive application where ultra-sound parking sensors randomise their sampling frequency to avoid the reduced effectiveness that would occur if the same sampling frequency were used by two vehicles reversing back-to-back.

In 2016 Ben-Amor et al. [23] derived probabilistic schedulability analysis for tasks with precedence constraints with execution times described by random variables, scheduled under EDF. They built upon the work by Chetto et al. [38] for tasks with deterministic parameters, deriving an equivalent schedulability analysis for the probabilistic case. This includes proposing a new comparison operator between distributions for probabilistic response times (or execution times) and deadlines that are also described by independent random variables. This comparison is equivalent to the method of subtracting the deadline distribution used by Maxim et al. [106]. The authors note that the greater than or equal to operator ($\succeq$) of Diaz et al. [55] cannot be used for such comparisons, since it would give optimistic results. For example, consider the two distributions $\mathcal{R} = \begin{pmatrix} 1 & 3 \\ 0.9 & 0.1 \end{pmatrix}$ and $\mathcal{D} = \begin{pmatrix} 2 & 4 \\ 0.8 & 0.2 \end{pmatrix}$. Even though $\mathcal{D} \succeq \mathcal{R}$, there is still a non-zero probability that the deadline is missed. This occurs when the response time has a value of 3 and the deadline has a value of 2. Assuming independence, then such a combination has a probability of 0.08 of occurring.

In 2018, Markovic et al. [103] presented a probabilistic schedulability analysis for the limited preemption model where each task is scheduled using fixed priorities and preemption is only permitted at fixed preemption points. A key aspect of this problem is the selection of preemption points for each task from a larger set of possible preemption points. The system model used assumes that each task is divided into sub-tasks, separated by possible preemption points. Further, the execution time of each sub-task and the preemption overhead of allowing preemption at a

particular point are represented by pWCET distributions. The authors utilise the approach of Diaz et al. [54], adapted along the lines of existing deterministic analysis for limited preemptive systems, to form a probabilistic schedulability analysis for limited preemption scheduling with fixed preemption points. Further, they propose an algorithm for preemption point selection, with the aim of minimising the deadline failure probability. This algorithm iterates over a number of confidence levels, starting at 1.0 and decrementing by a small step. The confidence level is used to reduce the pWCET distributions for sub-tasks and preemption point overheads to scalar values. This is done by taking the minimum value that does not have a probability of being exceeded that is more than the confidence level. This reduction of the random variables to scalar values enables an existing deterministic method to be used for preemption point selection. Once a set of preemption points have been selected then probabilistic schedulability analysis is used to determine the corresponding probability of deadline failure. The final preemption point selection is the one with the smallest probability of deadline failure found in any of the iterations. We note that by starting with a confidence level of 1.0, the algorithm is guaranteed to always find feasible solutions when the equivalent deterministic approach would do so.

Probabilistic Real-Time Calculus (an extension of Real-Time Calculus [142]) was proposed in 2011 by Santinelli and Cucu-Grosjean [130] with an extended journal version published in 2015 [131]. The authors study a task model where both execution times and inter-arrival times are described by independent random variables. A component model is used, with composability according to an *assume-guarantee* abstraction providing the basis for schedulability analysis. The model describes the resource demand of a task via a probabilistic upper bounding function (or curve). This curve upper bounds the cumulative demand as a function of the interval length t such that the probability that the actual demand exceeds the bound is less than a given probability threshold. Similarly, the resource supply (or service) is described by a lower bounding function (or curve) such that the probability that the actual supply is less than the bound is less than a given probability threshold. Different request and service curves may be obtained for different probability thresholds. The authors provide an algebra using Real-Time Calculus that facilitates the composition of systems from components and associated probabilistic schedulability analysis. The early work from 2011 [130] provides analysis for fixed priority scheduling, with extensions to EDF and hierarchical scheduling given later in 2011 by Santinelli et al. [136]. The journal extension [131] published in 2015 provides detailed proofs for the previous schedulability results. The same component-based formulation was also considered by Khan et al. [74] in 2012 in the context of multiprocessor scheduling, with identical cores, an interconnect (e.g. a TDMA bus), and a fixed partitioning of tasks between cores. They propose a mapping between the level of assurance (ASIL) needed for the safety of a function and its components and the probability threshold used. In 2016, Santinelli [129] proposed a component-based formulation for networks, similar to that previously proposed for tasks and processors [130, 131]. Here, the author considers the probability distribution for message payloads and inter-arrival times. We note that possible dependences between these distributions are not considered. The performance of the network is calculated for the case of AFDX switches implementing FIFO queues.

In 2014, Carnevali et al. [34] proposed computing the probability of deadline misses within a time interval t, based on modelling tasks using Stochastic Timed Petri Nets. They consider fixed-priority non-preemptive scheduling of periodic tasks. Task execution times are modelled via Erlang distributions, the parameters of which are selected to tightly upper bound the pWCET distribution for the task obtained via EVT. This enables regenerative transient analysis based on stochastic state classes to be used to compute the probability that each task misses a deadline within a time t.

3.4 Summary and Perspectives

Probabilistic response time analysis aims to compute either the response time distribution for each job over a relevant interval (such as the hyperperiod for a set of periodic tasks) or the worst-case response time distribution obtained for the worst-case scenario in terms of job releases (for sporadic tasks). Deadline miss probabilities and hence schedulability can then be determined via comparison between response time distributions and deadlines. We highlight three important threads of research in this area:

1. Diaz et al. [54] showed how to compute the probabilistic response time distributions for a set of jobs of periodic tasks taking into account the potential backlog accruing at the end of the hyperperiod. They also proved key properties required to ensure that the results obtained from probabilistic response time analysis are sustainable over-approximations [55].
2. Ivers and Ernst [70] showed that considering a worst-case convolution (i.e. matching the largest values from each distribution) is *not sufficient* to obtain a sound response time distribution when there are dependences between task execution times. They extended the approach of Diaz et al. [54, 55] to tasks with execution times that are related via unknown dependences.
3. Maxim and Cucu-Grosjean [106] proved that for systems with constrained deadlines where incomplete tasks are aborted at their deadline then the worst-case response time distribution is assumed by the first job of each task following synchronous release. Using this result, they derived probabilistic response time analysis for systems where execution times, inter-arrival times, and deadlines are all described by independent random variables.

In a further thread of research, Santinelli and various co-authors [130, 136, 131, 129] explored a probabilistic extension to Real-Time Calculus. We note that this approach relies heavily on the assumption that both the execution times and inter-arrival times of consecutive jobs are independent.

Two key problems that remain with probabilistic response time analysis are the tractability of the analysis for task sets of practical sizes (see Section 10 for initial work in this area), and issues relating to dependences between execution times.

4 Probabilistic Analysis assuming Servers

In this section, we review probabilistic schedulability analysis for tasks that are run within servers. Servers provide a partitioning of the available processor time and thus isolate the execution of each task from interference due to other tasks running on the same processor. This is useful in simplifying the analysis and in removing concerns regarding dependences between the execution times of jobs of different tasks.

4.1 Analysis for Server-based Systems

The majority of the research in this area has been published by Abeni and co-authors, including the seminal initial work [2, 3] from the late 1990s that also introduced the Constant Bandwidth Server, subsequent research that considers execution times [4, 121] and also inter-arrival times [6, 99, 122] described by independent random variables, and more recent work [59, 5] that considers dependences between the execution times of jobs of the same task.

In 1998 and 1999, Abeni and Buttazzo [2, 3] provided analysis for soft real-time tasks that have (i) variable execution times according to some probability distribution and fixed inter-arrival times, or (ii) fixed execution times and variable inter-arrival times according to some probability distribution. Jobs of a task are executed under a Constant Bandwidth Server (also introduced in the paper). This ensures independence between tasks, since each task can only utilise the capacity

of its own server. The authors provide statistical guarantees determining the probability that jobs of a task will meet their deadlines. This is achieved by modelling the Constant Bandwidth Server as a queue and determining the stationary solution of the Markov chain for those cases where the queue is stable. Conditions for stability of the queue comprise ensuring that the average utilisation does not exceed the server bandwidth. The output is the probability distribution of the response time by which the jobs will finish. This can then be used to determine the probability of meeting deadlines and hence the quality of service of the soft real-time tasks. Subsequently in 2001, Abeni and Buttazzo [4] extended their earlier work [2] on variable execution times, by relaxing the assumption that the server period must exactly divide the task period. They show how to handle inter-arrival times that are expressed as a probability distribution by approximating this distribution in accordance with the value of the server period. With this approximation, larger server periods generally introduce more pessimism, but have the advantage of resulting in a smaller number of context switches.

The more complex model where tasks have both execution times and arrival times modelled via random variables with known probability distributions was addressed by Kaczynski et al. [72] in 2007. They assume that there is no minimum time between arrivals for an aperiodic task and thus motivate the use of servers to prevent unbounded interference on other tasks. The authors extend the method introduced by Diaz et al. [54] (see Section 3.2) to this model, using a server for each task.

In 2012, Abeni et al. [6] considered tasks with both execution times and inter-arrival times described by probability distributions. Jobs of these tasks are scheduled via resource reservations where each task has its own server. The authors present an efficient algorithm to compute a bound on the probability of deadline failure. As part of the method, the inter-arrival time distributions are soundly approximated by simpler distributions limited to multiples of the server periods. Further, incomplete execution time distributions can be handled, provided that the remaining probability for values over some limit is known. The method provides a valid bound provided that the expected utilisation of each task does not exceed the capacity of its server. Evaluation shows that the bound produced is sound and that the over-approximation is small compared to exact analysis, which takes over 1000 times longer in some cases. In further works in 2012, Manica et al. [99] and Palopoli et al. [122] also considered tasks with both execution times and inter-arrival times described by probability distributions. Manica et al. [99] derived a model for tasks scheduled by a Constant Bandwidth Server. They show that the model takes the form of a Quasi-Birth-Death Process that can be solved using a numerical algorithm to determine a bound on the probability of a deadline miss. Palopoli et al. [122] used a conservative model for the evolution of a periodic task scheduled via a periodic server to construct a closed-form bound on the probability of a deadline miss. The model again takes the form of a Quasi-Birth-Death Process which enables efficient numerical computation of the probabilities required. For tasks with implicit deadlines, the model is further simplified allowing analytical calculation of a lower bound on the probability of meeting the deadline. This bound enables server bandwidth to be appropriately sized in order to meet requirements on the maximum permitted probability of deadline failure. Evaluation shows that the approximation error in the closed-form solution is high when the server bandwidth is close to the expected utilisation of the task, but reduces to acceptable levels as the bandwidth increases. In the former case, the probability of a deadline miss is in any case high ($> 60\%$), and so the cases where the approximation error is large are unlikely to be those that are of practical interest. An important limitation of the methods proposed by Manica et al. [99] and Palopoli et al. [122] relates to their pessimism due to the fact that the model used neglects the server budget that may be shared between consecutive jobs of a task.

In 2016, Palopoli et al. [121] considered periodic tasks with execution times described by probability distributions. Jobs of these tasks are scheduled via resource reservations where each task has its own Constant Bandwidth Server. They model the evolution of a task scheduled via a resource reservation as a Discrete-Time Markov Chain that takes the form of a Quasi-Birth Death Process. The outcome of the analysis is an expression for the steady state probability of meeting the deadline. This is used to construct a numerical algorithm and also an analytical bound that can be used to determine the probability of meeting the deadline. The evaluation compares the performance of the analytical bound, the numerical algorithm, and the authors' previous method [6]. The results show that the analytical bound has a runtime that can be orders of magnitude faster, and provides a high degree of accuracy in the cases where the bandwidth provided by the server is sufficient such that the probability of meeting the deadline is high ($> 90\%$). Further, the performance of the numerical algorithm can be tuned using a scaling factor, trading-off between runtime and accuracy. The effectiveness of the approach is demonstrated using a case study involving the playback of two video streams.

In 2017, Frias et al. [59] noted that many real-time applications exhibit a wide variation in execution times and are resilient to occasional deadline misses. Such systems have previously been afforded probabilistic schedulability guarantees based on the use of a resource reservation scheduler and the assumption that execution times are independent and identically distributed (i.i.d.). They consider robotic applications where the execution time of the vision algorithms depends on the complexity of the scene while also exhibiting a random behaviour due to a random element of the search. Thus the i.i.d. assumption does not hold. Instead execution times may be described via a Hidden Markov Model (HMM). The authors show how to identify the different modes of execution, and for each mode the distribution of execution times, which are independent within the mode. The Markov Computation Time Model (MCTM) used enables an accurate estimation of the probability of missing deadlines. The effectiveness of the approach is evaluated on a robot vision case study (a lane detection algorithm for a robotic car) where it is shown to provide accurate results. By comparison analysis based on an i.i.d. assumption leads to significant and optimistic under-estimation of the probability of missing deadlines. In a further short paper in 2017, Abeni et al. [5] show how the MCTM can be derived from a set of execution time measurements using the theory of HMM. Here, a task is modelled as having a number of states (modes) with probabilities for transitions between them. In each state the execution time is described by a different probability distribution, while within a state, the execution times are i.i.d. The method is able to estimate the number of modes from the raw execution time data. Assuming the number of states is known, then existing techniques (Baum-Welch algorithm) can be used to estimate the transition probability matrix and the probability distributions for the different modes. The authors therefore propose a method of finding the correct number of states via a gradient-like approach based on cross validation of the likelihood. The approach is validated on the robot vision case study (discussed above). It is also shown to recover a single state, as expected, for standard i.i.d. processes.

A software tool called PROSIT that supports the design and analysis of real-time systems based on the idea of probabilistic deadlines was initially described by Palopoli et al. [120] in 2015, with a more extensive description of a later version of the tool given by Frias et al. in 2018 [146]. PROSIT provides analysis for fixed priority preemptive scheduling of periodic tasks with execution times described by random variables, based on the analysis of Diaz et al. [54, 55]. It also provides an analysis of server-based scheduling of periodic and aperiodic tasks with execution times and potentially also inter-arrival times described by random variables. Here, the execution times can either be i.i.d. or a Markov process. The tool implements the various analyses proposed by Palopoli et al. [122] and Frias et al. [59] . In the case of server-based scheduling, PROSIT can be used to optimise the reservations (server budgets) for periodic tasks so as to optimise the global quality of service.

4.2 Summary and Perspectives

In this section, we reviewed work on probabilistic schedulability analysis for tasks that are run within servers that provide a partitioning of the available processor time. The use of servers has the advantage that the execution of each task is isolated from interference by other tasks; however, it has the disadvantage that jobs requiring a long execution time cannot directly make use of spare capacity freed up by jobs of other tasks that have a shorter than expected execution time. Arguably, this removes one of the main advantages of probabilistic schedulability analysis, which can otherwise take advantage of that fact that it is unlikely that jobs of multiple tasks will simultaneously require their worst-case or near to worst-case execution times.

We highlight the important thread of research initiated by Abeni et al. [2, 3] on probabilistic schedulability analysis for tasks that are run within servers, and the subsequent extensions of the analysis by Abeni et al. [6], Manica et al. [99], and Palopoli et al. [122] to tasks with both execution times and inter-arrival times described by independent random variables. While the use of servers removes issues of dependences between tasks, the issue of dependences between jobs of the same task remain. In practice, it is often the case that jobs of the same task exhibit dependences between their execution times, due to input variables that change only slowly over time, as well as execution starting from similar software and hardware states. Only in the recent work of Frias et al. [59] has this issue of dependences begun to be investigated. The analysis derived by Palopoli et al. [122] and Frias et al. [59] has recently been implemented in a software tool called PROSIT [146].

5 Real-Time Queueing Theory

In this section, we review research based on Queuing Theory, which is the mathematical study of queues with the aim of deriving information about queue lengths and waiting times. Problems in queuing theory are typically described in Kendall's notation [73] $A/B/C$ where A is the distribution of arrival times, B is the distribution of service (execution) times, and C is the number of servers. For example M/G/1 implies a Poisson or Markovian (M) arrival process, a General (G) execution time distribution, and a single (1) server.

5.1 Analysis based on Real-Time Queuing Theory

The majority of the research in this area has been published by Lehoczky and co-authors, including the seminal paper [83] on Real-Time Queuing Theory from 1996, and its later mathematical formalisation [56]. Subsequent work has investigated the impact of quantisation on EDF queues [67, 154], and extended the analysis to systems where jobs can be reneged, i.e. the remaining work of a job is discarded if its deadline is reached before it has completed execution [79].

The origins of work in this area can be traced back to the 1950s. In 1957, Barrer [20] considered the problem of queues with jobs that arrive according to a Poisson process, have execution times that follow an exponential distribution, i.e. an M/M/1 queue in Kendall's notation [73], and have a single fixed deadline. This work computes the ratio of the rate at which jobs miss their deadlines and are discarded to the arrival rate. In 1988, Panwar et al. [123] considered the problem of single-server queues with deadlines on jobs which become known on their arrival. The aim here is to schedule the jobs so that the fraction of jobs served within their deadline is maximised. The authors show that the Shortest Time to Extinction (STE) policy is optimal for this problem, for a class of non-preemptive M/G/1 queues that are work-conserving, i.e. do not permit inserted idle time, and where execution times are i.i.d random variables. The STE policy schedules the ready job with the earliest deadline that is still in the future. Jobs with expired deadlines are discarded.

They also showed that when inserted idle time is permitted, then policies from the STEI class are the best possible. (An STEI policy either schedules the job with the earliest unexpired deadline, or inserts idle time).

Real-Time Queueing Theory was introduced in 1996 by Lehoczky [83] as a means of analysing soft real-time systems where tasks have execution times that exhibit a large amount of variation. In such systems, using deterministic upper bounds on response times would lead to the system being significantly under-utilised on average. The method extends heavy traffic queueing theory utilizing the fact that the number of tasks in the queue behaves like reflected Brownian motion under heavy traffic. The theory can estimate the *lead-time* profile (meaning the time to go until the deadline) for the tasks in the queue, based on the distributions of arrival time, execution time, and deadline, and the queueing policy (EDF and processor sharing policies were considered). The fraction of missed deadlines can be derived from the queue-length dependent lead-time profiles and the distribution of queue lengths obtained via queueing theory. We note that the heavy traffic phenomenon occurs only for processor utilisations close to 1, which implies long queues and which may imply large latencies. Real-Time Queueing Theory was subsequently placed on firm mathematical foundations by Doytchinov et al. [56] in 2001. The heavy traffic constraint was subsequently relaxed by Zhu et al. [154]. Thus the main limitation in applying real-time queueing theory is that the probability distributions for all tasks must belong to the same family.

In 2002, Hansen et al. [67] examined the effect that quantisation has on EDF queues with a stochastic traffic model (arrival times, execution times and deadlines). Instead of using precise relative deadlines, this method quantises them, assigning tasks the next smaller deadline from a small set. The authors found via simulation that using just 3 bits (8 quantisation bins) was sufficient to obtain performance for Q-EDF close to that of EDF. They found that a log bin quantisation was more effective than a uniform one, since the log bins provide better granularity for small deadlines. The small number of bits required is important for network scheduling where this information takes up bandwidth. Also in 2002, following on from the work of Hansen et al. [67], Zhu et al. [154] aimed to optimise the set of quantisation bins used by Q-EDF. Using Real-Time Queueing Theory [83] they proved properties of a deadline distribution that would result in the worst-case behaviour for Q-EDF. It has tasks with deadlines that are at either end of the quantisation bins, leading to the maximum priority inversion. Hence they found that to minimise the maximum number of deadline misses for an arbitrary distribution of deadlines with the same minimum, maximum, and mean, one should create the bins by dividing the deadline range in a uniform way. Further they showed that with uniform bins 3-bits (8 bins) were sufficient for the performance of Q-EDF to converge to that of EDF in practical cases.

An alternative scheduling policy to EDF was examined by Gromoll and Kruk [63] in 2007. They considered processor sharing where all active jobs in the queue receive an equal share of the available processor time, and used heavy traffic queueing theory to obtain approximations for the lead-time profile and the profile of times in the queue.

The analysis given by Lehoczky [83] and Doytchinov et al. [56] assumes that late jobs continue to execute. Complementary work by Kruk et al. [79] in 2011 presented a heavy-traffic analysis of the behaviour of a single queue under an EDF scheduling policy adapted so that when the deadline of a job is reached, if the job has not yet completed, then its remaining work is discarded (referred to as *reneged*). The performance metric used is the fraction of work that is lost (reneged) due to missed deadlines. The authors show that this metric is minimised by using EDF scheduling. The evolution of the lead-time distribution of jobs in the queue is described by a measure-valued process. The heavy traffic limit of this process is shown to be a deterministic function of the limit of the scaled workload process which in turn is identified to be a doubly reflected Brownian motion. The fraction of reneged work in a heavily loaded system and the fraction of late work

in the corresponding system without reneging are compared using formulas based on the heavy traffic approximations. These formulas closely match simulation results, which show that the amount of work lost (i.e. reneged or late) due to deadline misses is reduced by a factor of one to two orders of magnitude if remaining work is discarded at the deadline.

5.2 Summary and Perspectives

In this section, we reviewed work on real-time queuing theory. The analysis used requires that the traffic intensity / processor utilisation is close to 1, where the queues are long, and the latencies may be high, which might not be acceptable in real systems. We note that although the theory does not characterise system behaviour at lower traffic intensities, the probability of deadline misses decreases as the traffic intensity decreases, all other parameters being equal. Thus the results for heavy traffic can serve as bounds on the behaviour at lower intensities.

We highlight the important thread of research initiated by Lehoczky [83] and Doytchinov et al. [56] on real-time queueing theory and its subsequent extension by Kruk at al. [79] to an EDF scheduler that discards any incomplete jobs when their deadlines expire. The main limitation of real-time queuing theory is that it requires job execution times, arrival times, and deadlines to be independent, which may not be the case in real systems.

6 Probabilities from Faults

In this section we review work on probabilistic schedulability analysis where random variables are used to represent the occurrence of some form of fault. Initial research focused on faults occurring in tasks running on a processor and the impact of fault recovery operations; however, the main thread of research in this area concerns Controller Area Network (CAN), a broadcast bus that is widely used in the automotive industry for in-vehicle networks.

6.1 Analysis of Fault Recovery on Processors

Research into probabilistic schedulability analysis for systems with faults was initiated by Burns and co-authors [33, 30, 27, 26], who considered the impact of fault recovery operations on tasks running on a processor.

In 1999, Burns et al. [33] addressed fault tolerant hard real-time systems. They introduced the concept of a probabilistic guarantee on schedulability for a fixed priority preemptive system. This is achieved by incorporating the cost of fault recovery (e.g. re-running a task, recovery block, executing since the last check point) into the analysis, along with the minimum time between faults. It is assumed that a single fault causes a detectable error in a single task. Sensitivity analysis is then used to determine a threshold on the minimum time between faults that can be tolerated by the system, while still meeting all deadlines. This threshold and the lifetime of the system are then used as parameters in a fault model which determines the probability that no two faults will occur closer together than the threshold during the lifetime of the system. Faults are modelled as a *homogeneous Poisson process* which enables this probability to be analytically computed, or upper and lower bounded using a simpler approach. We note that the approach is potentially somewhat pessimistic, since the schedulability analysis considers only a critical instant corresponding to synchronous release, whereas over much of the lifetime of the system the phasing of tasks may be such that a higher fault rate could be tolerated. Nevertheless, the method provides a valid lower bound on the probability that all jobs will meet their deadlines over the lifetime of the system.

Further work by Burns et al. [30] in 2003 looked at how conventional response time analysis [71, 14] could be extended to provide probabilistic guarantees. They remarked on the limitations of a deterministic approach: (i) in fault tolerant systems, fault arrivals are inherently stochastic, (ii) more complex applications have widely varying execution times, (iii) even for simple applications, advanced processor architectures (with cache, pipelines etc.) lead to wide variability in execution times. The authors present two methods of accounting for fault arrivals in response time analysis. The first method computes the maximum schedulable periodic fault rate from modified response time equations using sensitivity analysis [124]. An upper bound can then be derived on the probability of failure due to faults arriving closer together than the maximum tolerated rate. The second method computes the response time assuming no faults and then determines the maximum number of faults that could occur in that time with a probability higher than a specified threshold on the probability of failure. This number of faults is then added and the extended response time computed. This process iterates until it converges or the deadline is exceeded.

In 2004, Broster and Burns [27, 26] presented a simple method for computing probabilistic worst-case response times when there is just one task that has arrivals described by a probability distribution. This method is based on the approach taken to probabilistic modelling of fault arrivals on CAN [28] (see Section 6.2). The method works by computing the response times R_i^0 to R_i^m for task τ_i assuming 0 to m random arrivals of the higher priority task with random behaviour. It then determines the probability that each response time occurs. For example to obtain response time R^2, there must be exactly two arrivals within that interval, excluding the case where there are no arrivals in R_i^0 or exactly one arrival in R_i^1, since the latter cases result in a smaller response time. Once the probability of each response time has been computed, then the probability of deadline failure can be determined.

Also in 2004, Kim and Kim [75] presented a method of probabilistic schedulability analysis for harmonic task systems under Dual-Modular Temporal Redundancy (DMTR). DMTR utilises two processors to simultaneously run duplicates of each task, with execution split into sub-tasks that execute in defined time slots. At the end of each time slot, comparisons are made between the outputs of the sub-task duplicates. If they are the same, then processing proceeds with the next sub-task, otherwise the duplicates are re-executed and comparisons made between the four outputs (two new and two previous). If two or three matching outputs are found, then processing can proceed, otherwise the duplicate sub-tasks are executed again. A maximum of three time slots can be used for each sub-task, corresponding to three separate temporary data areas available to the processor. The authors derive a probabilistic schedulability analysis for the DMTR model, assuming transient faults that occur according to a Markov model with arrivals according to a Poisson process and an exponentially distributed duration. The approach is formulated using state transition probability matrices. The authors use their formulation to determine the optimal number of sub-slots to use given a fixed checkpointing overhead. They note that there are issues with the complexity of the analysis and restrict their examples and evaluation to three tasks.

In 2011, Aysan et al. [18] provided a probabilistic schedulability analysis for tasks, running under fixed priority preemptive scheduling, that are subject to faults in the form of error bursts. The model of recovery is that errors detected at the end of a task lead to the subsequent execution of an alternate, or the re-running of the original task. The error model analysed consists of bursts of errors occurring with a minimum inter-arrival time (T_E), with each burst having a duration described by a probability distribution and a number of errors within it separated by an intra-burst minimum inter-arrival time. Sensitivity analysis is used to compute the minimum value for T_E for each burst length in the distribution such that the task set remains schedulable if error bursts of that duration are separated by at least T_E. For each burst length and associated minimum value of T_E, the probability of the system remaining schedulable over its lifetime is computed, based

on a Poisson distribution of events (burst arrivals). This information is then used to compose the overall probability of the system remaining schedulable for its lifetime. The analysis takes account of the fact that errors may impact both the task of interest (whose schedulability is being checked) and higher priority tasks that may execute within its response time.

Error occurrence described by a two-state discrete Markov model was considered by Short and Proenza in 2013 [138]. This model specifies two states G and B (Good and Bad) and the probabilities for the transitions between them. Further, associated with each state is the probability of error occurrence while in that state. This model is a general one that can describe bursts of errors with probabilistic inter-arrival times between the start of each consecutive burst, and probabilistic inter-arrival times between the errors within a burst. The authors derive an efficient closed-form bound on the maximum number of errors occurring in an arbitrary time interval t according to this model, subject to a required confidence level or probability r. This bound can be used to provide a function giving the number of errors that must be tolerated as a function of t for the system to operate with a probability of failure that is no more than $1 - r$. The authors show how this function can be incorporated into standard schedulability analysis for EDF providing a "one-shot" analysis that can determine if a simple fault tolerant EDF-scheduled system can operate with a probability of deadline failure that is lower than $1 - r$. We note that a similar approach could be applied to fixed priority scheduling by integrating the bound into response time analysis.

In 2016, Santinelli et al. [134] discussed the idea of a C-Space (the space of task execution times that lead to a feasible, i.e. schedulable system) and how it can be adapted to a probabilistic model of execution times. The authors consider separate pWCET distributions resulting from (i) no-fault (or *LO-safe*) and (ii) fault (or *HI-safe*) behaviour for each task. These pWCET distributions are used to determine discrete execution time budgets with a probability of being exceeded of, for example, 10^{-9} for each task, assuming (i) no-fault and (ii) fault behaviour. The C-Space can then be used to indicate the probability of each combination of execution time budgets being exceeded (assuming i.i.d. execution times), and hence whether a particular set of execution time budgets can be considered feasible for a given combination of behaviours that have to be accommodated (for example task τ_1 considering LO-safe behaviour with no faults, and task τ_2 considering HI-safe behaviour with faults).

6.2 Analysis of Fault Recovery on CAN

Controller Area Network (CAN) is a broadcast bus used for in-vehicle networks. Messages sent on CAN have a bounded length and are transmitted according to a fixed priority non-preemptive scheduling policy, with the message ID also representing the message priority (see Davis et al. [50] for full details of the protocol and its analysis). CAN has strong error checking mechanisms that can detect faults that result in bit-errors on the bus and so cause message corruption. The protocol ensures that any message that fails to be transmitted correctly will be later re-sent. Hence faults result in an additional load on the bus due to re-transmissions, which can potentially result in deadline misses. Even though CAN is a deterministic protocol and the messages have bounded lengths, the random occurrence of faults means that analysis techniques are needed that can determine the probability of messages failing to meet their deadlines. A significant thread of research in this area began with the work of Navet et al. [117] in 2000. This was built upon by Broster et al. [28, 29], and Davis and Burns [48], and later adapted to more complex message arrival functions by Axer et al. [17].

In 2000, Navet et al. [117] proposed a fault model for messages on CAN based on random arrivals, where faults are assumed to occur according to a Poisson distribution. They introduce the idea of a *tolerable error threshold*, corresponding to the maximum number of errors that a message can tolerate before it becomes unable to meet its deadline. This threshold is then used in a calculation of the worst-case deadline failure probability (WCDFP).

Subsequently, in 2002, Broster et al. [28, 29] extended the work of Navet et al. [117], correcting and improving upon the WCDFP analysis. (We note that this work is based on early analysis of CAN that contains a number of flaws, but the method could easily use the correct equations that were derived by Davis et al. [51] in 2007). The analysis works by deriving the probability $p(R_m|K)$ that a worst-case response time of $R_m|K$ occurs, where $R_m|K$ corresponds to the worst-case response time for message m when exactly K faults occur between it being queued for transmission and transmission completing, i.e. within the response time of the message. The calculation of $R_m|K$ assumes the worst-case scenario, i.e. the maximum delay due to blocking, maximum interference from higher priority messages, and maximum bit stuffing. The probability that K faults occur in a given time interval is obtained from the Poisson distribution of faults. The probability $p(R_m|K)$ is computed for values of K from zero to the maximum number of faults that the message can tolerate without missing its deadline. The sum of these probabilities lower bounds the probability that the message will be successfully transmitted by its deadline, hence subtracting this sum from 1 gives an upper bound on the worst-case deadline failure probability. We note that due to the worst-case assumptions in the response time calculation, this WCDFP may be significantly larger than the actual probability of deadline failure averaged over a large number of instances of the message. In 2012, Axer et al. [17] extended the approach of Broster et al. [28] to more complex arrival functions for messages, including the case where messages have arbitrary deadlines.

The work of Broster et al. [28, 29] was improved upon by Axer and Ernst [16] in 2013. They considered the probabilistic schedulability analysis of messages on CAN assuming a Poisson distribution of faults. They present a method of probabilistic response time analysis based on the use of probability distributions representing queueing delays, busy period lengths, and response times. The key idea is to represent the worst-case total transmission time for each message including its re-transmission due to faults as a probability distribution, based on the probability of k faults occurring *within* transmission of that specific message. This is possible since the Poisson fault model is memoryless. The schedulability analysis derives from the deterministic response time analysis for CAN [51] adapted to consider probability distributions. The authors show how to compute the longest priority level-i busy period that can occur with a probability that is above some small threshold of interest. The response time distributions for each message of priority i in the busy period are then computed using a process of convolution and splitting. An upper bound on the probability that the response time of any message of priority i in the busy period will exceed an arbitrary time t (e.g. its deadline) can then be obtained from these distributions. Evaluation shows that the results given by the analysis are very close to the empirical distribution obtained via Monte Carlo simulation. Further, the probability of failure using typical fault rates for CAN is approximately one order of magnitude better than can be obtained via the analysis of Broster et al. [28, 29]. The reason being that the latter approach pessimistically assumes that the largest possible re-transmission time (for any higher priority message) occurs on every fault.

In 2009, Davis and Burns [48] introduced algorithms that determine Robust Priority Assignments (RPA) [47] for messages sent over CAN. They also investigated a probabilistic variant of this problem. This work builds on prior analysis of the worst-case deadline failure probability (WCDFP) for CAN messages in the presence of faults by Navet et al. [117] and Broster et al. [28, 29]. The authors derive a Probabilistic RPA algorithm which determines a robust and optimal priority ordering, in the sense that it returns a priority ordering which minimises the maximum WCDFP over all messages, provided that a schedulable priority ordering exists. The algorithm builds on the optimal priority assignment algorithm of Audsley [15]. A case study example shows that using the Probabilistic RPA algorithm can result in a worst-case failure rate that is orders of magnitude better than that obtained using deadline monotonic priority assignment (e.g. a failure rate of 1 in 28,500 compared to values in the range 1 in 500 to 1 in 1000).

With CAN, the physical layer employs *bit stuffing* to ensure that there are enough transitions in polarity to maintain synchronisation between the nodes on the network. Thus within each message transmission, a bit of opposite polarity is inserted after every five consecutive bits of the same polarity. This increases the transmission time of the messages. In 2003, Nolte et al. [119] provided a probabilistic worst-case response time analysis for messages on CAN. They used distributions of the number of stuff bits, as opposed to worst-case values, to calculate probabilistic response times based on the critical instant. The authors assume independence between the distributions of the number of stuff bits for different messages and between those numbers for instances of the same message. We note that it is unlikely that such independence would exist in practice. For example consider the situation at start up, when the vehicle is not moving. Many of the signals in the CAN messages may be at their default values e.g. zero, which incur a large number of stuff bits. Further many values (temperatures, pressures etc.) change only slowly over time, thus it is reasonable to expect a strong correlation between the values in one instance of a message and the next, and hence a strong correlation between the numbers of stuff bits.

A probabilistic analysis for CAN messages and end-to-end latencies in an automotive system was presented by Zeng et al. [153] in 2009. They build upon the basic approach of Diaz et al. [54, 55] (see Section 3.2), with a number of approximations and adaptations for distributed systems connected via CAN. Task execution times are assumed to be independent and described by probability distributions. Similarly, the transmission times of CAN messages are also assumed to be independent and described by probability distributions, in this case accounting for the varying levels of bit-stuffing assuming variable message contents. The key approximation introduced for CAN involves handling the lack of synchronisation between messages sent by different ECUs. (The entire system is not simply periodic). This is done by approximating all messages sent by a remote ECU via a single *characteristic message* that has a probability distribution and values for its transmission time equating to the number of messages of priority i or higher that may be released at the same time. For example if there are two messages with periods of 10 and 40 sent by the ECU, then the characteristic message will have a period of 10 (the greatest common denominator) and a probability distribution indicating 1 message with a probability of 0.75 and 2 messages with a probability of 0.25. The authors note that this introduces some inaccuracy into the analysis and it may be quite pessimistic for long intervals of time. Further, there is also the potential for optimistic (i.e. unsound) results. The analysis is, however, intended as an approximation to be used in design space exploration rather than as an upper bound. The characteristic message is given an offset and random jitter to account for the lack of synchronisation between messages transmitted by different ECUs. Messages sent by the same ECU as the message under analysis are treated as individual messages, since their phasing with respect to the message of interest is known. Blocking due to lower priority messages is also accounted for by assuming that the probability of such messages being transmitted is uniform over the hyperperiod. Again, the authors note this is a potential source of inaccuracy. The analysis method follows that of Diaz et al. [54, 55], first the stationary backlog is computed at the start of the hyperperiod (of messages on the ECU that transmits the message under analysis), then the backlog at the release time of each message instance is computed, and finally, the response time distribution of each message instance within the hyperperiod. These are averaged to obtain the response time distribution of the message. The evaluation considers a 69 message case-study based on an experimental vehicle. The results show that the stochastic analysis provides results that are close to those obtained via simulation averaged over 10^8 different relative phasings. The analysis is subsequently extended to end-to-end latency (i.e. tasks communicating across multiple ECUs and two CAN buses). Again, the analysis results are a close fit to those obtained via extensive system level simulation. The key advantage of the stochastic analysis over simulation is its speed e.g. 8 seconds of analysis versus 20 hours of simulation. This means that the analysis is much better suited for use in design space exploration.

Building on their prior work on probabilistic schedulability analysis for tasks [18], in 2012 Aysan et al. [19] derived probabilistic schedulability analysis for CAN under a general fault model. This model considers fault bursts of a duration described by a probability distribution. The analysis proceeds in three steps: (i) For each potential burst duration in the distribution, sensitivity analysis is used to determine the minimum inter-arrival time T_E^{burst} of errors within that burst such that the system remains schedulable. (ii) An upper bound is computed on the probability of a smaller inter-arrival time than T_E^{burst} occurring within the burst for each duration. These bounds are then used to determine an upper bound on the probability that the minimum tolerable inter-arrivals times for errors are violated for all potential fault durations over the mission duration. (iii) Finally, the probability of unschedulability is computed from the probability of two bursts occurring too close together and the probability of the minimum tolerable inter-arrival time of errors within a burst being violated.

6.3 Summary and Perspectives

In this section we reviewed work on probabilistic schedulability analysis where random variables are used to represent the occurrence of some form of fault. Beginning with initial work by Burns et al. [33] in 1999, we can trace an important thread of research providing probabilistic schedulability analysis for fixed priority systems, in particular Controller Area Network (CAN), under an assumed fault model [117, 28, 29, 48, 16, 17]. This work derives effective estimates of the worst-case probability of deadline failure, and provides the tools needed to assign message priorities in such a way as to make the system as robust as possible to the occurrence of faults. The main issue with this line of research is whether the fault models used reflect reality; however, the method is sufficiently flexible to incorporate any reasonable fault model where the probability of some number of faults is monotonically non-decreasing in the length of the time interval considered.

7 Statistical Analysis of Response Times

Previous sections reviewed work on probabilistic schedulability analysis based on analytical models of the system. By contrast, in this section we review work that takes a statistical approach, treating the system as a "black box" and making observations of response times from which an estimation of the response time distribution and hence deadline miss probabilities can be derived.

One of the key methods used in this thread of research is Extreme Value Theory (EVT). An overview of the use of EVT in measurement-based probabilistic timing analysis can be found in the companion survey [52], with more detailed information given in Stuart Coles' textbook on the subject [39]. Here, we provide a brief synopsis, focusing on the Extreme Value Theorem (or Fisher–Tippett–Gnedenko theorem). This theorem states that if the normalised maximum of a sequence of i.i.d. random variables converges, then the limit distribution belongs to either the Gumbel, Frechet, or reversed Weibull family of distributions. In practice, EVT may be applied using the *Block Maxima* method as follows: (i) obtain a representative sample of observations (e.g. response times), (ii) check using appropriate statistical tests that the sample of observations collected is analysable using EVT, (iii) divide the sample into blocks of observations of a fixed size, and take the maxima for each block, (iv) fit a *Generalised Extreme Value* (GEV) distribution (i.e. reversed Weibull, Gumbel, or Frechet distribution, depending on the shape parameter) to the distribution of the maxima, (v) check the goodness of fit between the distribution of the maxima and the fitted GEV distribution. The GEV distribution so obtained then approximates (estimates) the distribution of the *extreme values* of the sampled distribution. We note that if the underlying (measured) distribution is badly behaved, then the normalised maximum may not converge to any of the limit distributions, in which case the method is not applicable. This can be determined by appropriate statistical tests.

7.1 Statistical Estimation

A number of authors have sought to apply statistical methods to estimate response time distributions and deadline miss probabilities. The main thread of research in this area comes from Lu, Nolte, and their co-authors [96, 97, 94, 95] with a recent investigation into the soundness of such approaches by Maxim et al. [111].

The theory of *Large Deviations* [144] was applied by Navet et al. [116] in 2007 to the problem of estimating the mean or the sum of the response times of a series of aperiodic jobs. This method makes use of frequency histograms of response times that are obtained via measurement. It assumes that the response times of jobs of an aperiodic task are i.i.d. The authors note that this is not the case with the response times of periodic tasks, since the interference from other tasks follows a pattern over the hyperperiod due to the release times of higher priority tasks. The method provides an estimation of the probability that the mean response time of a sequence of n jobs of the task will exceed some value x, for all values of x.

In 2010, Lu et al. [96] introduced a method of estimating probabilistic worst-case response times (pWCRT) using Extreme Value Theory (EVT). They record observations of response times, obtained from simulation. EVT is then applied to these observations, using the Block Maxima method, with the distribution of the maxima fitted to a Gumbel distribution using a χ-squared test. The authors present an algorithm which searches for an appropriate block size to use, while enforcing a minimum of 30 blocks. The results are compared to those obtained via a Monte-Carlo search (i.e. keeping the largest response time found from a set of randomised simulations) and also via meta-heuristic search applied on top of Monte-Carlo simulation. The evaluation considers three system models (M1-M3) indicative of those used in robotic control systems. Here, tasks exhibit strong dependences through asynchronous message passing, shared global variables, and runtime changes to task periods and priorities. The system models used in the evaluation vary in complexity. The validation model based on M1 is amenable to conventional response time analysis techniques, and so an exact worst-case response time could be determined. By contrast, M3 has intricate dependences via message passing, global shared variables, and changes in task periods and priorities. The evaluation results show that the EVT-based approach requires far fewer simulation runs (approx. 6% as many) to produce meaningful results compared to the Monte-Carlo and search-based methods. Following on from this work, Lu et al. [97, 94, 95] refined the method, using a form of simple random sampling to break dependences between observations. They also sought to ensure that the pWCRT value returned by their tool (RapidRT) for a given probability of exceedance is an upper bound with an appropriate level of confidence. This is done by repeating the process of obtaining observations and applying EVT n times to produce n pWCRT distributions. The set of values at a probability of exceedance of 10^{-9} from each of these distributions is then checked to see if it complies with a normal distribution. If so, the pWCRT value returned is the one that corresponds to the desired level of confidence (e.g. $3\sigma \approx 99.7\%$). In their final work in this area, Lu et al. [95] evaluated their method using a case study based on an industrial robotic control system with the results compared against a state of the art method based on using meta-heuristic search to guide Monte Carlo simulation to determine parameters that will lead to long response times. Note such simulation requires a detailed model of the system. Four levels of system complexity were explored containing from 40-60 tasks, 7-12 queues, and in the case of the most complex system, run-time priority and period changes, unbounded message passing, and task offsets. The proposed method was shown to bound the estimates obtained via meta-heuristic search and Monte Carlo simulation, with no more than 15% pessimism.

Subsequent work in 2013 by Liu et al. [91] applied EVT to the problem of estimating the worst-case response times of messages on a CAN bus. Due to the scheduling policy used, the distribution of observations and their maxima show multiple peaks. Such distributions are difficult

to analyse, since they cannot be fitted to the known EVT distributions. To address this problem the authors use a filtering method which aims to reduce the distributions to single peaks by discarding observations below a threshold. (This threshold is set such that the mean value of observations above the threshold is greater than or equal to their median value). Evaluation shows that the method provides results that are only a few percent pessimistic compared to response time analysis for CAN [50] using computed worst-case values.

The soundness and precision of applying statistical techniques to determine the probabilistic worst-case response time (pWCRT) distribution of tasks was investigated by Maxim et al. [111] in 2015. They noted that to obtain meaningful results, a ground truth is required. In other words the pWCRT must be known. This is far from simple, and may not be possible for tasks in a real system. Therefore they constructed a simulation of task behaviour based on pWCET distributions, which could potentially be obtained from a real system. The approach obtains the ground truth via probabilistic worst-case response time analysis using the method given by Maxim et al. [106] (see Section 3.3), which determines precise pWCRT distributions from the input pWCETs. The ground truth is compared to a number of statistical approaches. These include fitting to Normal, reversed Weibull, and Gumbel distributions, and an EVT-based approach using the Block Maxima method. The evaluation shows that fitting to a Normal or reversed Weibull distribution is unsound with approximately half of the pWCRTs under-estimating the probability of a deadline failure. Fitting to a Gumbel distribution produced better results in this respect with about 10% unsound results. Using the EVT-based approach, none of the results were unsound; however, there was an increase in pessimism compared to directly fitting a Gumbel distribution.

7.2 Summary and Perspectives

In this section we reviewed research that takes a statistical approach to estimating response time distributions. Of particular note is the work by Lu et al. [96, 97, 94, 95] and Maxim et al. [111]. The former showing that EVT can provide meaningful predictions of the tail of response time distributions even in the case of systems with intricate dependences, and the latter showing that the results from EVT are sound compared to the ground truth, while those from directly fitting a distribution to the observations are not.

All of the work reviewed in this area has focused on single processor systems. For tasks running on COTS multi-core platforms there are significant difficulties involved in obtaining precise worst-case response times via analytical methods due to issues of cross-core contention. The application of statistical methods to directly predict the extreme values of the response time distributions for tasks in such systems could potentially provide some solutions to this problem. This is an interesting area for future research.

8 Probabilistic Analysis of Mixed Criticality Systems

The term Mixed Criticality System (MCS) is used to describe real-time systems where applications with different *criticality levels* (meaning different levels of assurance required against failure) are integrated onto the same hardware platform. This integration gives rise to research questions in terms of how to reconcile the conflicting requirements of sharing for efficient resource usage and separation for reasons of assurance [31]. In 2007, Vestal [145] described a mixed criticality task model whereby LO-criticality tasks have a single worst-case execution time estimate $C(LO)$, and HI-criticality tasks have two estimates $C(LO)$ and $C(HI)$, with the latter, larger estimate obtained via methods that give a higher level of confidence / assurance that it will not be exceeded. (For example $C(LO)$ might be an upper bound on the longest execution time observed during testing, while $C(HI)$ may be a conservative value obtained via detailed static timing analysis). The timing

constraints placed on the system require that all tasks meet their deadlines provided the $C(LO)$ budgets are not exceeded; however, if a HI-criticality task exceeds its $C(LO)$ budget, then it is *only* required that the HI-criticality tasks meet their deadlines, assuming that they execute for at most their $C(HI)$ budgets. This required behaviour reflects the different failure rates that may be acceptable at different criticality levels (see the discussion in Section 1). For more information on research into scheduling mixed criticality systems, see the survey by Burns and Davis [31].

In this section we review recent research on probabilistic schedulability analysis for MCS. These methods typically use a richer representation based on execution time or pWCET distributions, rather than the discrete execution time budgets $C(LO)$ and $C(HI)$ at different criticality levels assumed by Vestal's model [145]. Here, one may consider the $C(LO)$ and $C(HI)$ budgets from Vestal's model as two points on the x-axis of the 1 - CDF of a pWCET distribution (see Figure 2 in Section 2), each with an associated probability of exceedance (i.e. the corresponding y-axis value).

8.1 Analysis for Mixed Criticality Systems

Research into probabilistic schedulability analysis for mixed criticality systems is in its infancy with a small number of papers published from 2015 onwards. The majority of these works are short papers that have appeared in workshops. A necessarily brief review of them is given below.

In 2015, Santinelli and George [132] presented preliminary work on probabilistic schedulability analysis for MCS scheduled using EDF. They investigated how schedulability varies with task execution times, referred to as the probabilistic C-space. Later the same year, Guo et al. [64] extended the mixed criticality task model with a single exceedance probability value for the low assurance budget of each HI-criticality task, and used probabilistic analysis to improve schedulability.

In 2016, Maxim et al. [107] adapted probabilistic response time analysis from [106] (see Section 3.3) to fixed priority preemptive scheduling of MCS using the Adaptive Mixed Criticality (AMC) and Static Mixed Criticality (SMC) schemes [22]. They compared this analysis to the equivalent deterministic methods, highlighting the performance gains that can be obtained by utilising more detailed information about worst-case execution time estimates described in terms of probability distributions. This work was extended by Maxim et al. [108] to provide a more precise analysis, and also to examine by how much the execution time budgets of LO-criticality tasks can be increased by employing probabilistic rather than deterministic schedulability analysis methods.

In 2016, Alahmad and Gopalakrishnan [9, 8] studied the problem of scheduling mixed criticality job sets with execution times described by random variables. The aim of this work is to compute implementable scheduling policies that meet the probabilistic timing constraints. The problem is modelled as a Constrained Markov Decision Process (CMDP), with feasible policies obtained using a linear program.

In 2016, Draskovic et al. [57] examined fixed priority preemptive scheduling of MCS of periodic tasks with execution times described by random variables. They employed the method of Diaz et al. [54] (see Section 3.2) to compute the probability of a deadline miss for every job in the hyperperiod, and from that the overall deadline failure rate. They also computed the expected time before a change to HI-criticality mode, and showed that this expected time depends on the LO-criticality execution time budget allocated to HI-criticality tasks. A smaller budget results in a lower probability of deadline failure, but a shorter expected time before a transition to HI-criticality mode.

In 2017, Abdeddaim and Maxim [1] derived probabilistic response time analysis for mixed criticality tasks under fixed priority preemptive scheduling, adapting the techniques of Maxim and

Cucu-Grosjean [106] (see Section 3.3) to the MCS model. The analysis computes the probability of deadline misses for each task in each criticality mode. This work does not assume any monitoring, hence lower criticality tasks are assumed to continue executing in higher criticality modes.

In 2017, Kuttler et al. [80] introduced an algorithmic approach to probabilistic schedulability analysis called *symbolic scheduling*. They considered an extension of AMC [22] where the priorities of LO-criticality tasks are reduced (to below that of any HI-criticality task) when the system switches to HI-criticality mode. Assuming this behaviour, they calculate the probability of each job of a LO-criticality task meeting its deadline. The method applies to periodic tasks. Conceptually, it considers every possible combination of execution times, forming a tree where each path from root to leaf represents a possible behaviour of the system. The disadvantage of this naive approach is that the tree quickly becomes very large. Symbolic scheduling is therefore used, whereby paths that may have different execution times but agree on the order of jobs and their success or otherwise in meeting deadlines are combined. The evaluation shows that considering only the probabilistic worst-case response time (i.e. the behaviour at the critical instant) can be pessimistic in its estimate of the probability of LO-criticality jobs missing their deadlines.

8.2 Summary and Perspectives

In this section we reviewed research on probabilistic schedulability analysis for Mixed Criticality Systems (MCS). These methods consider MCS described using execution time or pWCET distributions rather than the conventional $C(LO)$ and $C(HI)$ WCET estimates / execution time budgets of Vestal's model [145]. This additional information provides the potential for improvements in schedulability and in the size of the budgets that can be afforded to different tasks, see for example the work of Maxim et al. [108]. MCS are a hot topic of real-time systems research. A probabilistic view of MCS would appear to provide an excellent match to requirements that are specified in terms of levels of assurance and failure rates. We note, however, that research in this area is currently in its infancy with a small number of works starting in 2015, the majority of which are workshop papers or other short publications. (For a comprehensive review of other research into MCS see the survey by Burns and Davis [31]).

9 Miscellaneous

In this section, we review research that explores miscellaneous aspects of scheduling and schedulability analysis for probabilistic real-time systems, including task graphs and precedence constraints, analysis for multiprocessor systems, miscellaneous models and techniques, and position papers.

9.1 Task Graphs and Precedence Constraints

The majority of the research in this area was published by Manolache et al. in a series of papers [100, 101, 102] from 2001 to 2008.

In 2001, Manolache et al. [100] presented a method of analysing systems with precedence relations between tasks described by task graphs, and task execution times described by probability distributions. They assume that the tasks are periodic with a reasonably small hyperperiod. The method is applicable only to non-preemptive scheduling algorithms such as fixed priority and EDF that do not alter job priorities between scheduling points (i.e. task release times and deadlines). It is assumed that if a job misses its deadline, then it is aborted. The first step in the analysis is to divide the hyperperiod into so called *Priority Monotonic Intervals* (PMI) de-marked by job release times and deadlines. The stochastic process is then constructed and analysed at the same time, thus reducing memory requirements. A stochastic process state consists of the index of the

currently running job, the start time of the job, and the indices of the ready jobs. The number of next states depends on the number of possible execution times of the job. As this number can be very large, states are grouped together, while still preserving the Markovian property. States are processed in order, by PMI first, and then within a PMI by highest priority ready job. The method thus determines the expected deadline miss probability for each task. Evaluation shows that the method is effective for tasks sets of cardinality up to 20 and hyperperiods from 360 to 5040. Subsequently, in 2004, Manolache et al. [101] extended their earlier work [100] to the case where tasks may continue to execute beyond their deadlines. Such overruns are restricted by limiting the maximum number of jobs of the same task that can exist in the system at any given time. On the release of a task, if this limit would be exceeded, then there are two options: discard the oldest job of the task, or reject the new job. Evaluation shows that rejecting the new job leads to much greater complexity, since a bound is removed on the number of successor states. The authors also discuss possible extensions to preemptive scheduling, but note that the complexity of the method would be greatly increased. Later, in 2008, Manolache et al. [102] proposed a solution to the problem of task priority assignment and mapping in a multiprocessor system. The task model is the same as in their previous works [100, 101]. The method is based on a Tabu search, with various approximations used to reduce the complexity of computing estimates of the deadline miss probability and thus the fitness function used in the search.

In 2003, Hua et al. [69] proposed a method of using probabilistic descriptions of task execution times to optimise other parameters of interest in multimedia systems, such as energy consumption. The model considered is a task graph, where the tasks in the graph are executed in a fixed order, and must be completed by a given deadline. The application i.e. the task graph is executed periodically. The system must meet a *completion ratio* condition, effectively a threshold on the expected proportion of a large number of instances of the task graph that will meet their deadlines. A simple formula is given for computing the completion ratio based on the execution time distributions. This formula has exponential complexity, since it effectively considers all combinations of possible task execution times from the distributions. An approximation is therefore used that starts with each task assigned its WCET, and then while the deadline is not met, it removes the largest value from one of the task execution time distributions. This lowers the overall execution time, but decreases the completion ratio. Eventually, either the task graph is deemed schedulable with an acceptable completion ratio, or the completion ratio becomes too small. The value to remove is chosen in a greedy way, by selecting the one that gives the largest reduction in overall execution time, weighted by how much the completion ratio is reduced. The authors also describe an offline/online algorithm for minimising energy consumption via dynamic voltage and frequency scaling (DVFS). The aim here is to either drop jobs or to extend their execution to reduce energy consumption, while meeting the specified threshold on the completion ratio. We note that it is implicitly assumed that the execution time of a job becomes known at the point when it is released, which may not be possible to achieve in practice.

9.2 Multiprocessor Analysis

Below, we cover the few works on probabilistic schedulability analysis for multiprocessor systems. The relative absence of work in this area contrast strongly with the wealth of research into conventional schedulability analysis techniques for multiprocessor systems, (see Davis and Burns [49] for a survey).

In 2002, Nissanke et al. [118] and Leulseged and Nissanke [86] described a probabilistic framework for investigating the schedulability of tasks on a multiprocessor, with execution times and deadlines modelled via probability distributions. Each task has a fixed period, and its behaviour is represented by points on a cartesian graph of remaining execution time versus laxity.

A set of non-zero probabilities characterise the task as arriving with a certain execution time and laxity. Between arrival times, remaining computation times and laxities are also described by probabilities, but evolve according to the scheduling algorithm, reaching either negative laxity indicating a missed deadline or zero remaining computation time indicating completion. By knowing the probability of realising each scenario, and the competition between tasks at the same priority, the probability of a task being executed is computed. This enables calculation of the execution patterns of all tasks over the hyperperiod, enabling the various properties of interest to be derived. The movement of tasks through this scheduling domain depends on the probability of m processors being assigned tasks to execute that are in a particular state (specified by laxity and execution time), and also on the arrival rate. Overall, the framework can be used to determine performance indicators such as expected deadline failure rate, success rate, number of tasks executing, number of tasks at a particular point etc. The authors propose that a probabilistic scheduling policy could be determined by prescribing a probability to executing tasks based on their location in the scheduling domain. These probabilities could be obtained by solving an optimisation problem with the aim of maximising some performance indicator of interest, such as the expected success rate.

In 2010, Mills and Anderson [112] extended prior work on tardiness bounds for global EDF (GEDF) scheduling to tasks with execution times described by i.i.d. random variables. For such systems, they derived a bound on expected mean tardiness for all tasks. Subsequently in 2011, Mills and Anderson [113] generalised their previous work to address tasks with stochastic execution times (specified via mean and variance) scheduled via sporadic servers under GEDF or any other global scheduling algorithm with bounded tardiness. They proved a worst-case tardiness bound when the system has a worst-case utilisation that is bounded by the number of processors, and an expected or average-case tardiness bound when the average-case utilisation is bounded by the number of processors. This latter bound does not require knowledge of the task's WCET, or even for the WCET to be bounded. Hence the average-case tardiness bound can be computed on the basis of the mean and variance obtained from representative execution time observations. An example shows that the computed tardiness bounds are much tighter that those derived previously [112].

In 2014, Liu et al. [92] considered how to deal with dependencies between the execution times of jobs of a task. They build upon the work of Mills and Anderson [113], thus assuming that tasks are scheduled via sporadic servers under GEDF. The key idea is to represent the stochastic execution times of the task via two components: (i) a fixed threshold, and (ii) an excess over that threshold. The idea is that by tuning the threshold to an appropriate level, the non-zero excesses over the threshold become independent. (This notion is similar to the one of extremal independence, where extreme execution time values are sufficiently rare and far apart as to be independent). Using an *independence threshold* for each task enables the system designer to balance the need for a tractable probabilistic analysis based on modeling execution times as independent random variables, and the need to avoid a pessimistic provision based on deterministic worst-case reasoning. The authors integrate the concept of independence thresholds into the prior approach of Mills and Anderson [113]. They present a measurement process based on statistical tests of independence that is able to find the smallest threshold such that dependences are effectively eliminated. Finally, they show via an MPEG video decoding case study that the overall approach is highly effective, on average achieving a two-fold reduction in the required server execution time budgets compared to deterministic worst-case provisioning.

In 2015 and a later journal extension in 2017, Wang et al. [149, 148] proposed a task partitioning algorithm for fixed priority preemptive scheduling of tasks with execution times described by independent random variables on a homogeneous multi-core platform. They explored four different

heuristics that quantify the degree of harmonicity among the tasks assigned to a processor. These are mean-based, variance-based, cumulative distribution-based, and distribution sum-based. The evaluation shows that these heuristics significantly outperform existing (deterministic) methods in terms of the number of cores required to ensure that probabilistic timing guarantees are met (i.e. that all tasks meet their deadlines with an acceptable probability). Later work by Ren et al. [128] built upon the above work addressing some of its drawbacks. In particular, the partitioning approach employed by Wang et al. [149, 148] can suffer from fragmentation. Ren et al. address this issue by combining a consideration of both harmonicity and probabilistic workload. Their approach first orders the tasks by decreasing expected utilisation and selects the highest expected utilisation task as a reference task. It then selects tasks to add to the subset containing the reference task based on harmonicity. Tasks are added until no further task can be added without the subset failing the probabilistic schedulability test for a single processor. The selected subset of tasks is then assigned to one processor and the method repeats for the remaining unassigned tasks. Evaluation using synthetic task sets shows that this approach is both more effective and has a shorter average runtime than the previous partitioning approach of Wang et al. [149, 148].

9.3 Miscellaneous Models and Techniques

In this subsection we review work relating to miscellaneous models and techniques such as randomised job dropping and imprecise computation.

In 2001, Hu et al. [68] studied fixed priority preemptive systems with task execution times described by random variables. They argue that finding the probability of each task missing its deadline does not give the full picture for a system and can be misleading. Instead, they propose that the probability of failures is assessed over *state cycles* corresponding to the intervals between job releases for periodic tasks. They reason that a *state* is only feasible if all of the jobs that are ready in that state meet their deadlines. (A job with a long deadline may thus affect the feasibility of multiple states). The overall probability of system feasibility is assessed by determining the expected number of feasible states over the total number of states in the hyperperiod, or as an approximation a shorter interval such as the task period. The focus on states ensures that correlations between different jobs missing their deadlines are captured. We note that this method does not consider the backlog at the end of the hyperperiod and thus is only applicable if the worst-case processor utilisation does not exceed 1. Constrained deadline periodic tasks are assumed. The method is also extended to EDF. The complexity of the method is exponential in the number of values in the execution time distributions, with the exponent being the number of releases of the task within the hyperperiod. It therefore seems unlikely that the method is viable for systems that do not have both a small number of tasks and a short hyperperiod.

Also in 2001, Hamann et al. [65] integrated a probabilistic description of execution times with the imprecise computation model based on *mandatory* and *optional* components [90]. They assume that each task is composed of a single mandatory part that must be guaranteed to complete by its deadline and multiple optional parts for which only soft (probabilistic) guarantees are required. It is assumed that the execution time distribution is provided for each part of a task, and that the WCET is known for the mandatory part. A simple analysis is given that determines the size of the reservation required to guarantee the mandatory part and to provide the desired probabilistic guarantee that a required percentage of the optional parts complete. This is achieved by convolving the execution time distributions, hence their independence is assumed. The approach is motivated by multimedia examples involving the decoding of MPEG frames, with I and P frames mandatory and B frames optional.

In 2002, Kim et al. [77] proposed an alternative to task-level or job-level isolation based on *randomised dropping*. The motivation for this approach is that isolation does not allow sharing of processing resources when a job executes in less time than reserved for it. In the model addressed in this work, periodic tasks are assumed to have independent execution times with known probability distributions, and are assigned an execution time budget that they can use without triggering the dropping mechanism. Typically this corresponds to the expected or average execution time. Job scheduling is via EDF; however, each job also has one or more trigger values on the execution time that it uses. When one of these trigger values is reached, there is an associated probability that the job will be dropped. By tuning these values and the probabilities of dropping, the interference on other jobs can be limited in a probabilistic way. The authors propose a stochastic analysis for their model, based on a Markov process. They derive the Markov chain over multiple hyperperiods, and compute the stationary backlog distribution. This is then used to determine the response time distribution and deadline miss probability for each job, and hence also for each task. The randomised dropping is modelled as an adjustment to the execution time distribution for each task. Evaluation shows that the method is successful in achieving a high probability of deadlines being met in an otherwise overloaded system.

In 2009, Gopalakrishnan [61] explored the idea of sharp utilisation thresholds in fixed priority preemptive scheduling of periodic tasks. They show that for task sets chosen uniformly at random, there is a transition around some utilisation U where the probability of an implicit deadline task set being schedulable under rate-monotonic scheduling changes from 1 to 0. The width of this transition depends on the cardinality of the task set and tends to a sharp threshold (i.e. an interval of zero width) as the number of tasks tends to infinity. A similar result giving a sharp synthetic utilisation (or density) threshold was obtained for aperiodic tasks, where a task's density is given by its execution time divided by its relative deadline. This work provides a highly efficient means of admitting task sets at runtime based on a simple utilisation-based test, while ensuring that there is a high probability that the task sets will be schedulable. For soft real-time systems this approach could be much more effective than using hard utilisation bounds [89], below which there are no task sets that are unschedulable.

9.4 Position Papers

The following works discuss some requirements for probabilistic schedulability analysis to be useful in practice, as well as issues relating to independence.

In 2012, Quinton et al. [125] set out four requirements (or conditions) that must be satisfied by probabilistic analysis for it to be useful: (i) it must be efficient enough to scale to real systems, (ii) it must provide meaningful results for system designers, (iii) the model used must be practical (i.e. simple enough to be provided by the designer) or automatically generated, (iv) any assumptions made by the analysis must be formally described so they can be validated. They note that most existing probabilistic approaches determine the distribution of response times, but can say nothing about the behaviour of the system in a short and bounded time window. In other words, they cannot answer the question, "can deadline misses occur in a burst?" (See the discussion in Section 2.5).

In 2013, Cucu-Grosjean [42] considered different types of *independence* in the context of probabilistic real-time systems. A key aspect of this work is the discussion covering the definition of, and the differences between, probabilistic execution time distributions (pET) of jobs and probabilistic worst-case execution time distributions (pWCET) of tasks. The author notes that since the pWCET distribution upper bounds all the pET distributions for the jobs of a task (in the sense of the greater than or equal to operator $\succeq$ on random variables defined in [55]), then the pWCET distribution of a task is *by definition* probabilistically independent with respect to jobs

of the same or different tasks. They also highlight the differences between independence between tasks (as required by the Liu and Layland model [89]), probabilistic independence between random variables (needed so that basic convolution may be used to determine probabilistic response times), and statistical independence between observations of execution times. In a paper accompanying an invited talk at the ETR summer school in 2013, Cucu-Grosjean [43] discusses different types of independence (probabilistic, statistical) [42] and also recaps the work on probabilistic schedulability analysis [54] and probabilistic WCET analysis [44] for real-time systems, re-sampling techniques [109], and priority assignment policies [105].

9.5 Summary and Perspectives

In this section, we reviewed works that explore different aspects of scheduling and schedulability analysis for probabilistic real-time systems. Here, we highlight the work of Liu et al. [92] that provides a means of dealing with dependencies between the execution times of jobs of a task via an *independence threshold*, and the work of Gopalakrishnan [61] that provides a simple but highly effective probabilistic admission test for soft real-time task sets. We note that while there is substantial literature on conventional schedulability analysis for multiprocessor systems (see Davis and Burns [49] for a survey), with a handful of exceptions, research into probabilistic schedulability analysis has focused on uniprocessor systems. Further, there are now a large number of papers focused on providing conventional schedulability analysis for tasks running on multi-core platforms, taking into account the effects of contention for shared hardware resources (interconnect, memory hierarchy, I/O system etc.) between tasks running in parallel on different cores. There appears to be little if any published work on probabilistic schedulability analysis for such systems. This is an important area that could benefit from future research.

10 Addressing Issues of Intractability

Much of the research into probabilistic schedulability analysis relies on combining execution time distributions via basic convolution. A naive assessment of basic convolution would assume that it has exponential complexity $O(m^n)$ where m is the number of points in each distribution and n is the number of distributions convolved. While this is correct in terms of the theoretical worst-case, in practice the range of values in each distribution is such that the overall complexity is far lower. For example, assuming that the largest (integer) value in an execution time distribution is N, then the number of points in the intermediate distribution after m convolutions is at most mN, and hence the overall complexity of m convolutions is $O(Nm^2)$. Nevertheless, probabilistic response time analysis involving realistic numbers of tasks with a spread of periods of a few orders of magnitude can involve significant computation. In this section, we review works that seek to reduce the amount of computation required, while in some cases trading faster calculation for pessimism in the results.

10.1 Re-sampling

Re-sampling reduces the number of values present in a discrete probability distribution, and therefore reduces the amount of computation required in convolution operations involving that distribution. This improvement in efficiency comes at a cost of reduced accuracy or pessimism in the results.

In 2010, Refaat et al. [127] presented a method of reducing the complexity of the analysis of Diaz et al. [54, 55] and Kim et al. [76] by re-sampling the execution time distributions used. Their method involves taking k random samples from the probability distribution and assigning

the probability for all excluded points to the worst-case value from the distribution, which is always kept. This method provides a sound but pessimistic approximation [55]. Subsequently, later in 2010, Maxim et al. [110] presented an alternative approach to re-sampling execution time distributions that improves on the method introduced by Refaat et al. [127]. This approach keeps the $k-1$ values with the largest probability as well as the largest value. It then reassigns the probability mass for each removed point to the retained point with the next larger value. This is a sound approximation, with less pessimism than assigning the probability mass of excluded points to the worst-case value.

Building on the earlier work in this area, in 2012, Maxim et al. [109] considered the need for re-sampling probabilistic worst-case execution time (pWCET) distributions when computing probabilistic worst-case response time (pWCRT) distributions. This computation involves repeated use of the convolution operator. The runtime of convolution of *arbitrary* distributions can grow rapidly, as the number of points (distinct execution times) in the existing distribution can in the worst case be multiplied by the number of points in each pWCET distribution that is convolved onto it. In theory, this leads to an exponential growth in the runtime. In practice, the number of points in a discrete distribution cannot exceed max-et – min-et, where max-et and min-et are the maximum and minimum values in the distribution. The authors address the complexity issues with convolution by introducing sound ways of re-sampling the distributions created. A sound re-sampling is one that does not move probability values right-to-left, thus never allocating them to a smaller execution time. Although moving some values left-to-right introduces pessimism, it ensures that the pWCRT produced over-approximates that which would be obtained without re-sampling. The work considers *uniform spacing*, a re-sampling technique that is widely used in other contexts. This method selects sample points that are uniformly spaced in terms of probabilities, i.e. at equally spaced percentiles throughout the distribution. It provides a good fit in terms of overall pessimism; however, the shape of the tail, which is important in pWCRT calculation, can be heavily compromised. Two new re-sampling techniques are introduced: *reduced pessimism* re-sampling, which seeks to minimise the probability mass moved to larger execution time values, and *domain quantisation*, which re-samples at equally spaced points in the execution time domain. Domain quantisation has the desirable property that it greatly reduces the number of points in the distributions produced *after* convolution, and also provides a good fit to the tail of the pWCET distributions. Evaluation shows that it gives the best compromise between runtime and accuracy.

In 2015, Milutinovic et al. [114] examined methods of speeding up the discrete convolution operations that are used a large number of times in SPTA. They consider methods that are precision preserving, such as power operations used when the same distribution needs to be convolved multiple times, and precision non-preserving methods such as re-sampling as proposed by Maxim et al. [109]. They found that the precision preserving techniques speeded up convolution by approximately a factor of two, while the precision non-preserving techniques traded off a minimal amount of over-approximation ($< 3\%$) for an order of magnitude increase in speed.

10.2 Analytical Methods and Other Techniques

While re-sampling, which reduces the number of values present in a discrete probability distribution, makes a trade-off between efficiency and precision in probabilistic schedulability analysis calculations, analytic methods can result in greater improvements in runtime while retaining better precision. In some cases, full precision can be retained while still making substantial improvements in runtime efficiency.

In 2017, Chen and Chen [36] considered the problem of probabilistic response time analysis and the computational complexity involved in repeated use of the convolution operator. They

proposed a more efficient approach to computing the probability of deadline misses, based on using the *moment generating function* (mgf) of random variables, and Chernoff bounds for the probability that the sum of a number of random variables (e.g. the execution times of multiple jobs) exceeds some bound (e.g. the deadline). The authors demonstrate how this approach can be applied to probability distributions consisting of two values representing normal operation, and abnormal operation in the event of a soft error. They also show how the approach can be extended to distributions with more values, and to give the probability of l consecutive deadline misses. The evaluation compares the proposed method to exact analysis [106] and to exact analysis with re-sampling [109] applied. The results show that the proposed method is able to efficiently determine slightly pessimistic bounds on the probability of deadline misses without the need to derive the whole response time distribution, which can be inefficient. Exact analysis was not suitable for more than 10 tasks in the experiments considered, while re-sampling, limiting the distributions to a maximum of 100 values, resulted in highly pessimistic deadline miss probabilities (e.g 1) for task set utilisation values $> 70\%$.

In 2018, von der Bruggen et al. [147] considered the problem of determining the worst-case probability of deadline misses for tasks under fixed priority preemptive scheduling. They note that traditional convolution-based approaches to computing pWCRT distributions quickly become intractable as the number of jobs within the deadline of a lower priority task that is being analysed increases. To address this problem, they present a novel approach based on using multinomial distributions, which they further improve via the use of a pruning technique. This method retains full precision and is viable for much larger task sets than previous approaches. In the evaluation, the technique is used for systems of up to 35 tasks and is shown to be viable for up to 100 tasks. These task sets have a range of periods spanning two orders of magnitude. Hence there can be up to 100 jobs of each higher priority task within the response time of the lower priority task under analysis. The authors also present two methods based on analytical upper bounds based on Hoeffding's inequality and Bernstein's inequality. These methods offer further substantial improvements in runtime (two orders of magnitude faster than the precise multinomial-based approach), but trade off some precision in the results.

Later in 2018, Chen et al. [37] studied the problem of determining the *expected* deadline miss rate for tasks under fixed priority preemptive scheduling. The task model used assumes that each task has a normal execution time and a longer abnormal execution time that occurs when dealing with fault conditions. As the faults are assumed to be independent, the execution time distribution for each task is i.i.d. Further, the probabilities for the different execution times reflect the probability of fault occurrence. The authors make the realistic assumption that job execution continues even if a deadline is missed. (By contrast, some other works assume that jobs are aborted on reaching their deadline). The authors show that this difference in behaviour can have a substantial effect in increasing the expected deadline miss rate, since the overrun of a job affects the probability of the next job meeting its deadline. They derive an upper bound on the expected deadline miss rate. This is done by considering the probability that a task has j consecutive deadline misses within the same busy period, for all values of j up to some limit. The method leverages the authors' prior work [36, 147] to determine the probability of j consecutive deadline misses. Using the convolution-based approach [147] results in bounds that are tighter with respect to the simulation results, compared to using the analytical bound [36]. The trade-off is however a significantly greater runtime.

10.3 Summary and Perspectives

Issues of tractability were once considered a substantial roadblock to the use of probabilistic schedulability analysis on practical systems. This issue has been addressed, first by methods based on re-sampling [109] that can reduce the amount of computation required to perform

convolution (the basic operation used repeatedly in many probabilistic schedulability analyses) while trading off additional pessimism in the results. More recently, analytical approaches have been developed that have a much shorter runtime, but still trade off pessimism in the results [36]. Finally, the work of von der Bruggen et al. [147] in 2018, provides an approach which promises an efficient method of computing deadline miss probabilities for large task sets without a significant trade-off in precision.

11 Conclusions

In this survey, we reviewed research into schedulability analysis for probabilistic real-time systems. We covered the main subject areas including probabilistic response time analysis, probabilistic analysis assuming execution time servers, real-time queuing theory, probabilities emanating from fault models, statistical analysis of the response times, and probabilistic analysis of mixed criticality systems; as well as reviewing supporting mechanisms and analyses that address issues of intractability.

We now conclude by identifying open issues, key challenges and possible directions for future research. We present these as a series of questions and statements.

- How to determine the (worst-case) execution time distribution for a task? This is the subject of probabilistic timing analysis, see the companion survey [52] for a detailed discussion. We note that in some cases the variation of the execution times over time may be such that using a single valid distribution may be too pessimistic (e.g. when the system exhibits different modes of behaviour).
- How to handle issues relating to dependences between the execution times of jobs of (i) the same task, and (ii) jobs of different tasks? The impact of these dependences may vary based on how strong they are. Appropriate statistical studies are needed to investigate the types of dependences and their impact on probabilistic schedulability analysis. Analyses are needed that can address dependencies.
- How to reconcile requirements on the maximum length of black-out periods (number of consecutive missed deadlines) with a probabilistic treatment of deadline failures? This problem relates to dependences between response times that may occur due to dependences between the execution times of jobs of the task considered, and due to dependences in the amount of interference from other tasks.
- How to provide probabilistic schedulability analysis based on probabilistic Worst-Case Execution Time (pWCET) distributions when there are dependences between execution times of consecutive jobs? This is particularly problematic in the case of pWCET distributions derived via MBPTA techniques (see the discussion in Section 2.3).
- How to provide appropriate solutions for multiprocessor schedulability analysis? While there is a wealth of research into conventional schedulability analysis for multiprocessor systems, research into probabilistic schedulability analysis has, with only a few exceptions, focussed on uniprocessor systems.
- How to adapt probabilistic models, using the richer description given by pWCET distributions, in the context of Mixed Criticality Systems. Although expanding rapidly (see Figure 1), work in this area is still in its infancy.
- How to adapt the current statistical approaches such as Extreme Value Theory in the context of response time analysis? The use of EVT has shown some promise in this area, but has not been explored in detail.

- How to ensure that probabilistic schedulability analysis methods are viable for use in practical systems? Issues here include validation of the methods, and ensuring that they can be applied to problems of a practical size.

Since the initial work in the late 1980s and 1990s, significant progress has been made in the development of probabilistic schedulability analysis techniques. However, there are still important unanswered questions and open issues to be resolved, as well as a number of interesting areas for future research that are only beginning to be explored. We end this survey with a brief discussion of an important direction for future real-time systems research which probabilistic analysis techniques may be able contribute to.

There is a continuing trend in industry sectors including avionics, automotive electronics, consumer electronics, and robotics away from development and deployment on single-core processors towards using significantly more powerful and complex Common-Off-The-Shelf (COTS) multi-core and many-core hardware platforms. This trend is driven by requirements on size, weight and power consumption, increasing cost pressures and the demand for more complex and capable functionality delivered through software. The use of COTS multi-core hardware poses significant challenges in terms of verifying timing behaviour and ensuring that real-time constraints are met. These challenges stem from the complexity of the architecture and the way in which hardware resources such as the interconnect and the memory hierarchy are shared between different processing cores. Some researchers are seeking to address these problems through approaches based on partitioning and separation (e.g. single-core equivalence [98]), while others aim for solutions based on considering the explicit interference on each hardware resource from co-running programs and how this demand can be served by the available resource supply [12, 46]. There is the potential for probabilistic schedulability analysis and probabilistic timing analysis techniques (reviewed in a companion survey [52]) to play a role in the timing verification of such complex real-time systems.

Work on probabilistic timing analysis and probabilistic schedulability analysis for multi-core and many-core systems is in its infancy with opportunities for significant advances addressing this important research challenge.

Acknowledgements. The research that went into writing this survey was funded, in part, by the Inria International Chair program and the ESPRC grant MCCps (EP/P003664/1). EPSRC Research Data Management: No new primary data was created during this study.

References

1 Y. Abdeddaim and D. Maxim. Probabilistic Schedulability Analysis for Fixed Priority Mixed Criticality Real-Time Systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2017.

2 L. Abeni and G. Buttazzo. Integrating multimedia applications in hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 4–13, December 1998. doi:10.1109/REAL.1998.739726.

3 L. Abeni and G. Buttazzo. QoS guarantee using probabilistic deadlines. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 242–249, 1999. doi:10.1109/EMRTS.1999.777471.

4 L. Abeni and G. Buttazzo. Stochastic analysis of a reservation based system. In *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001*, pages 946–952, April 2001. doi:10.1109/IPDPS.2001.925049.

5 L. Abeni, D. Fontanelli, L. Palopoli, and B. Villalba Frías. A Markovian model for the computation time of real-time applications. In *Proceedings of IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1–6, May 2017. doi:10.1109/I2MTC.2017.7969878.

6 L. Abeni, N. Manica, and L. Palopoli. Efficient and Robust Probabilistic Guarantees for Real-time Tasks. *J. Syst. Softw.*, 85(5):1147–1156, May 2012. doi:10.1016/j.jss.2011.12.042.

7 Z. Alabedin, H. Hammadeh, S. Quinton, and R. Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Proceedings of the IEEE & ACM International Conference on Embedded Software*

(EMSOFT), pages 10:1–10:10, 2014. doi:10.1145/2656045.2656059.

8 B. Alahmad and S. Gopalakrishnan. A Risk-Constrained Markov Decision Process Approach to Scheduling Mixed-Criticality Job Sets. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2016.

9 B. Alahmad and S. Gopalakrishnan. Risk-aware scheduling of dual criticality job systems using demand distributions. Leibniz Transactions on Embedded Systems. In *Leibniz Transactions on Embedded Systems (LITES)*, 2016.

10 S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Springer Real-Time Systems*, 51(1):77–123, 2015. doi:10.1007/s11241-014-9218-4.

11 S. Altmeyer and R. I. Davis. On the Correctness, Optimality and Precision of Static Probabilistic Timing Analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 26:1–26:6, 2014. URL: http://dl.acm.org/citation.cfm?id=2616606.2616638.

12 S. Altmeyer, R. I. Davis, L. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. A Generic and Compositional Framework for Multicore Response Time Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 129–138, 2015. doi:10.1145/2834848.2834862.

13 A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, pages 123–132, December 1998. doi:10.1109/REAL.1998.739737.

14 N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

15 N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001. doi:10.1016/S0020-0190(00)00165-4.

16 P. Axer and R. Ernst. Stochastic response-time guarantee for non-preemptive fixed-priority scheduling under errors. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–7, May 2013. doi:10.1145/2463209.2488946.

17 P. Axer, M. Sebastian, and R. Ernst. Probabilistic response time bound for CAN messages with arbitrary deadlines. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1114–1117, March 2012. doi:10.1109/DATE.2012.6176662.

18 H. Aysan, R. Dobrin, S. Punnekkat, and R. Johansson. Probabilistic Schedulability Guarantees for Dependable Real-Time Systems under Error Bursts. In *Proceedings of IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pages 1154–1163, November 2011. doi:10.1109/TrustCom.2011.157.

19 H. Aysan, R. Dobrin, S. Punnekkat, and J. Proenza. Probabilistic scheduling guarantees in distributed real-time systems under error bursts. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–9, September 2012. doi:10.1109/ETFA.2012.6489644.

20 D. Y. Barrer. Queuing with Impatient Customers and Ordered Service. *Operations Research*, 5(5):650–656, 1957. doi:10.1287/opre.5.5.650.

21 S. Baruah and A. Burns. Sustainable Scheduling Analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 159–168, 2006. doi:10.1109/RTSS.2006.47.

22 S. K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43. IEEE, 2011.

23 S. Ben-Amor, D. Maxim, and L. Cucu-Grosjean. Schedulability Analysis of Dependent Probabilistic Real-time Tasks. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 99–107. ACM, 2016. doi:10.1145/2997465.2997499.

24 G. Bernat, A Burns, and A Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4):308–321, April 2001. doi:10.1109/12.919277.

25 G. Bernat and R. Cayssials. Guaranteed online weakly-hard real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 25–35, December 2001. doi:10.1109/REAL.2001.990593.

26 I. Broster and A. Burns. 1st International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems (PARTES). In *Random Arrivals in Fixed Priority Analysis*, 2004.

27 I. Broster and A. Burns. Work-in-Progress of the IEEE Real-Time Systems Symposium. In *Applying Random Arrival Models to Fixed Priority Analysis*, December 2004.

28 I. Broster, A. Burns, and G. Rodriguez-Navas. Probabilistic analysis of CAN with faults. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 269–278, 2002. doi:10.1109/REAL.2002.1181581.

29 I. Broster, A. Burns, and G. RodrÍguez-Navas. Timing Analysis of Real-Time Communication Under Electromagnetic Interference. *Springer Real-Time Systems*, 30(1):55–81, 2005. doi:10.1007/s11241-005-0504-z.

30 A. Burns, G. Bernat, and I. Broster. *A Probabilistic Framework for Schedulability Analysis*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. doi:10.1007/978-3-540-45212-6_1.

31 A. Burns and R. I. Davis. A Survey of Research into Mixed Criticality Systems. *ACM Comput. Surv.*, 50(6):82:1–82:37, November 2017. doi:10.1145/3131347.

32 A. Burns and S. Edgar. Predicting computation time for advanced processor architectures. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 89–96, 2000. doi:10.1109/EMRTS.2000.853996.

33 A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright. Probabilistic scheduling guarantees for

fault-tolerant real-time systems. In *Dependable Computing for Critical Applications 7, 1999*, pages 361–378, November 1999. doi:10.1109/DCFTS.1999.814306.

34 L. Carnevali, A. Melani, L. Santinelli, and G. Lipari. Probabilistic Deadline Miss Analysis of Real-Time Systems Using Regenerative Transient Analysis. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 299:299–299:308, 2014. doi:10.1145/2659787.2659823.

35 F. J. Cazorla, E. Quiñones, T. Vardanega, L. Cucu, B. Triquet, G. Bernat, E. Berger, J. Abella, F. Wartel, M. Houston, L. Santinelli, L. Kosmidis, C. Lo, and D. Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 12(2s):94:1–94:26, May 2013. doi:10.1145/2465787.2465796.

36 K. H. Chen and J. J. Chen. Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, June 2017. doi:10.1109/SIES.2017.7993392.

37 Kuan-Hsun Chen, Georg von der Bruggen, and Jian-Jia Chen. Analysis of Deadline Miss Rates for Uniprocessor Fixed-Priority Scheduling. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2018.

38 H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints. *Springer Real-Time Systems*, 2(3):181–194, September 1990. doi:10.1007/BF00365326.

39 S. Coles. *An Introduction to Statistical Modeling of Extreme Values*. Springer, 2001. doi:10.1007/978-1-4471-3675-0.

40 L. Cucu. Preliminary results for introducing dependent random variables in stochastic feasibility analysis on CAN. In *Proceedings of IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 271–274, Dresden, Germany, May 2008. IEEE. doi:10.1109/WFCS.2008.4638759.

41 L. Cucu and E. Tovar. A Framework for the Response Time Analysis of Fixed-priority Tasks with Stochastic Inter-arrival Times. *SIGBED Rev.*, 3(1):7–12, January 2006. doi:10.1145/1279711.1279714.

42 L. Cucu-Grosjean. Independence a misunderstood property of and for probabilistic real-time systems. *In Real-Time Systems: the past, the present and the future*, pages 29–37, 2013.

43 L. Cucu-Grosjean. Probabilistic real-time scheduling. In *ETR 2013-Ecole d'été temps réel*, 2013.

44 L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 91–101, July 2012. doi:10.1109/ECRTS.2012.31.

45 R. I. Davis. A review of fixed priority and EDF scheduling for hard real-time uniprocessor systems. *ACM SIGBED Review*, 11(1):8–19, 2014.

46 R. I. Davis, S. Altmeyer, L. S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. An extensible framework for multicore response time analysis. *Springer Real-Time Systems*, 54(3):607–661, July 2018. doi:10.1007/s11241-017-9285-4.

47 R. I. Davis and A. Burns. Robust Priority Assignment for Fixed Priority Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 3–14, December 2007. doi:10.1109/RTSS.2007.11.

48 R. I. Davis and A. Burns. Robust priority assignment for messages on Controller Area Network (CAN). *Springer Real-Time Systems*, 41(2):152–180, 2009. doi:10.1007/s11241-008-9065-2.

49 R. I Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys (CSUR)*, 43(4):35, 2011.

50 R. I Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Springer Real-Time Systems*, 35(3):239–272, 2007.

51 R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007. doi:10.1007/s11241-007-9012-7.

52 R. I. Davis and L. Cucu-Grosjean. A Survey of Probabilistic Timing Analysis Techniques for Hard Real-Time Systems. *Leibniz Transactions on Embedded Systems (LITES)*, 6(1):03:1–03:60, May 2019. doi:10.4230/LITES-v006-i001-a003.

53 R. I. Davis, L. Santinelli, S. Altmeyer, C. Maiza, and L. Cucu-Grosjean. Analysis of Probabilistic Cache Related Pre-emption Delays. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–179, July 2013. doi:10.1109/ECRTS.2013.27.

54 J. L. Diaz, D. F. Garcia, K. Kim, C-G. Lee, L. Lo Bello, J. M. Lopez, S. L. Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 289–300, 2002. doi:10.1109/REAL.2002.1181583.

55 J. L. Diaz, J. M. Lopez, M. Garcia, A. M. Campos, Kanghee Kim, and L. L. Bello. Pessimism in the stochastic analysis of real-time systems: concept and applications. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 197–207, December 2004. doi:10.1109/REAL.2004.41.

56 B. Doytchinov, J. Lehoczky, and S. Shreve. Real-time queues in heavy traffic with earliest-deadline-first queue discipline. *The Annals of Applied Probability*, 11(2):332–378, 2001. doi:10.1214/aoap/1015345295.

57 S. Draskovic, P. Huang, and L. Thiele. On the Safety of Mixed-Criticality Scheduling. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2016.

58 S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 215–224, December 2001. doi:10.1109/REAL.2001.990614.

59 B. Frias, L. Palopoli, L. Abeni, and D. Fontanelli. Probabilistic Real-Time Guarantees: There is Life Beyond the i.i.d. Assumption. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

60 M. K. Gardner and J. W. S. Liu. *Analyzing Stochastic Fixed-Priority Real-Time Systems*, pages 44–58. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. doi:10.1007/3-540-49059-0_4.

61 S. Gopalakrishnan. Sharp utilization thresholds for some real-time scheduling problems. *CoRR*, abs/0912.3852, 2009. URL: http://arxiv.org/abs/0912.3852.

62 D. Griffin, I. Bate, B. Lesage, and F. Soboczenski. Evaluating Mixed Criticality Scheduling Algorithms with Realistic Workloads. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2015.

63 H. Christian Gromoll and Łukasz Kruk. Heavy traffic limit for a processor sharing queue with soft deadlines. *The Annals of Applied Probability*, 17(3):1049–1101, June 2007. doi:10.1214/105051607000000014.

64 Z. Guo, L. Santinalli, and K. Yang. EDF Schedulability Analysis on Mixed-Criticality Systems with Permitted Failure Probability. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2015.

65 C. J. Hamann, J. Loser, L. Reuther, S. Schonberg, J. Wolter, and H. Hartig. Quality-assuring scheduling-using stochastic behavior to improve resource utilization. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 119–128, December 2001. doi:10.1109/REAL.2001.990603.

66 J. Hansen, S. A. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation . In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, volume 252, 2009.

67 J. P. Hansen, J. P. Lehoczky, H. Zhu, and R. Rajkumar. Quantized EDF Scheduling in a Stochastic Environment. In *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS '02, pages 279–, Washington, DC, USA, 2002. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=645610.660905.

68 X. S. Hu, Tao Zhou, and E. H. M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):833–844, December 2001. doi:10.1109/92.974897.

69 S. Hua, G. Qu, and S. S. Bhattacharyya. Exploring the probabilistic design space of multimedia systems. In *in IEEE International Workshop on Rapid System Prototyping, 2003*, pages 233–240, 2003.

70 M. Ivers and R. Ernst. *Probabilistic Network Loads with Dependencies and the Effect on Queue Sojourn Times*, pages 280–296. Springer Berlin Heidelberg, 2009. doi:10.1007/978-3-642-10625-5_18.

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

72 G. A. Kaczynski, L. Lo Bello, and T. Nolte. Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 101–110, September 2007. doi:10.1109/EFTA.2007.4416759.

73 D. G. Kendall. Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain. *The Annals of Mathematical Statistics*, 24(3):338–354, September 1953. doi:10.1214/aoms/1177728975.

74 D. A. Khan, L. Santinelli, and L. Cucu-Grosjean. Modeling uncertainties in safety-critical real-time systems: A probabilistic component-based analysis. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 166–175, June 2012. doi:10.1109/SIES.2012.6356582.

75 J. K. Kim and B. K. Kim. Probabilistic Schedulability Analysis of Harmonic Multi-Task Systems with Dual-Modular Temporal Redundancy. *Springer Real-Time Systems*, 26(2):199–222, March 2004. doi:10.1023/B:TIME.0000016130.91111.75.

76 K. Kim, J. L. Diaz, L. Lo Bello, J. M. Lopez, C-G. Lee, and S. L. Min. An Exact Stochastic Analysis of Priority-Driven Periodic Real-Time Systems and Its Approximations. *IEEE Trans. Comput.*, 54(11):1460–1466, November 2005. doi:10.1109/TC.2005.174.

77 K. Kim, L. Lo Bello, S. L. Min, and O. Mirabella. On Relaxing Task Isolation in Overrun Handling to Provide Probabilistic Guarantees to Soft Real-Time Tasks with Varying Execution Times. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 193–, Washington, DC, USA, 2002. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=787256.787354.

78 G. Koren and D. Shasha. Skip-Over: algorithms and complexity for overloaded systems that allow skips. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 110–117, December 1995. doi:10.1109/REAL.1995.495201.

79 L. Kruk, J. Lehoczky, K. Ramanan, and S. Shreve. Heavy traffic analysis for EDF queues with reneging. *The Annals of Applied Probability*, 21(2):484–545, 2011. doi:10.1214/10-AAP681.

80 M. Kuttler, M. Roitzsch, C-J Hamann, and Marcus Volp. Probabilistic Analysis of Low-Criticality Execution. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2017.

81 J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: exact characterization and average case behavior. In *Pro-*

ceedings of the IEEE Real-Time Systems Symposium (RTSS), pages 166–171, December 1989. doi:10.1109/REAL.1989.63567.

82 J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 201–209, December 1990. doi:10.1109/REAL.1990.128748.

83 J. P. Lehoczky. Real-time queueing theory. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 186–195, December 1996. doi:10.1109/REAL.1996.563715.

84 B. Lesage, D. Griffin, S. Altmeyer, L. Cucu-Grosjean, and R. I. Davis. On the analysis of random replacement caches using static probabilistic timing methods for multi-path programs. *Real-Time Systems*, December 2017. doi:10.1007/s11241-017-9295-2.

85 B. Lesage, D. Griffin, S. Altmeyer, and R. I. Davis. Static Probabilistic Timing Analysis for Multi-path Programs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 361–372, December 2015. doi:10.1109/RTSS.2015.41.

86 A. Leulseged and N. Nissanke. Probabilistic Analysis of Multi-processor Scheduling of Tasks with Uncertain Parameters. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 103–122, 2003. doi:10.1007/978-3-540-24686-2_7.

87 G. Lima and I. Bate. Valid Application of EVT in Timing Analysis by Randomising Execution Time Measurements. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

88 G. Lima, D. Dias, and E. Barros. Extreme Value Theory for Estimating Task Execution Time Bounds: A Careful Look. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016.

89 C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46–61, January 1973. doi:10.1145/321738.321743.

90 Jane W. S. W. Liu. *Real-Time Systems*. Prentice Hall, 1st edition, 2000.

91 M. Liu, M. Behnam, and T. Nolte. An EVT-based worst-case Response Time Analysis of complex real-time systems. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 249–258, June 2013. doi:10.1109/SIES.2013.6601498.

92 R. Liu, A. F. Mills, and J. H. Anderson. Independence Thresholds: Balancing Tractability and Practicality in Soft Real-Time Stochastic Analysis. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 314–323, December 2014. doi:10.1109/RTSS.2014.38.

93 J. M. López, J. L. Díaz, J. Entrialgo, and D. García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Springer Real-Time Systems*, 40(2):180–207, 2008. doi:10.1007/s11241-008-9053-6.

94 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A statistical response-time analysis of complex real-time embedded systems by using timing traces. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 43–46, June 2011. doi:10.1109/SIES.2011.5953676.

95 Y. Lu, T. Nolte, I. Bate, and L. Cucu-Grosjean. A Statistical Response-Time Analysis of Real-Time Embedded Systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 351–362, December 2012. doi:10.1109/RTSS.2012.85.

96 Y. Lu, T. Nolte, J. Kraft, and C. Norstrom. Statistical-Based Response-Time Analysis of Systems with Execution Dependencies between Tasks. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 169–179, March 2010. doi:10.1109/ICECCS.2010.55.

97 Y. Lu, T. Nolte, J. Kraft, and C. Norström. A Statistical Approach to Response-Time Analysis of Complex Embedded Real-Time Systems. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 153–160, August 2010.

98 R. Mancuso, R. Pellizzoni, M. Caccamo, L. Sha, and H. Yun. WCET(m) Estimation in Multi-core Systems Using Single Core Equivalence. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 174–183, July 2015. doi:10.1109/ECRTS.2015.23.

99 N. Manica, L. Palopoli, and L. Abeni. Numerically efficient probabilistic guarantees for resource reservations. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2012. doi:10.1109/ETFA.2012.6489566.

100 S. Manolache, P. Eles, and Z. Peng. Memory and Time-Efficient Schedulability Analysis of Task Sets with Stochastic Execution Time. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 19–, Washington, DC, USA, 2001. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=871910.871936.

101 S. Manolache, P. Eles, and Z. Peng. Schedulability Analysis of Applications with Stochastic Task Execution Times. *ACM Transactions on Embedded Computing Systems*, 3(4):706–735, November 2004. doi:10.1145/1027794.1027797.

102 S. Manolache, P. Eles, and Z. Peng. Task Mapping and Priority Assignment for Soft Real-time Applications Under Deadline Miss Ratio Constraints. *ACM Transactions on Embedded Computing Systems*, 7(2):19:1–19:35, January 2008. doi:10.1145/1331331.1331343.

103 F. Markovic, J. Carlson, R. Dobrin, B. Lisper, and A. Thekkilakattil. Probabilistic Response Time Analysis for Fixed Preemption Point Selection. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, June 2018. doi:10.1109/SIES.2018.8442099.

104 D. Maxim and A. Bertout. Analysis and Simulation Tools for Probabilistic Real-Time Systems. In *Proceedings of International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

105 D. Maxim, O. Buffet, L. Santinelli, L. Cucu-Grosjean, and R. I. Davis. Optimal Priority Assignment Algorithms for Probabilistic Real-Time Systems. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 129–138, 2011.

106 D. Maxim and L. Cucu-Grosjean. Response Time Analysis for Fixed-Priority Tasks with Multiple Probabilistic Parameters. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 224–235, December 2013. doi:10.1109/RTSS.2013.30.

107 D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic Analysis for Mixed Criticality Scheduling with SMC and AMC. In *Proceedings of Workshop on Mixed Criticality (WMC)*. York, 2016.

108 D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic Analysis for Mixed Criticality Systems using Fixed Priority Preemptive Scheduling. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, 2017.

109 D. Maxim, M. Houston, L. Santinelli, G. Bernat, R. I. Davis, and L. Cucu-Grosjean. Re-sampling for Statistical Timing Analysis of Real-time Systems. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 111–120, 2012. doi:10.1145/2392987.2393001.

110 D. Maxim, L. Santinelli, and L. Cucu-Grosjean. Improved sampling for statistical timing analysis of real-time systems. In *the 4th Junior Researcher Workshop on Real-Time Computing*, Toulouse, France, November 2010. URL: https://hal.inria.fr/inria-00544651.

111 D. Maxim, F. Soboczenski, I. Bate, and E. Tovar. Study of the Reliability of Statistical Timing Analysis for Real-time Systems. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 55–64, 2015. doi:10.1145/2834848.2834878.

112 A. F. Mills and J. H. Anderson. A Stochastic Framework for Multiprocessor Soft Real-Time Scheduling. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 311–320, April 2010. doi:10.1109/RTAS.2010.33.

113 A. F. Mills and J. H. Anderson. A Multiprocessor Server-Based Scheduler for Soft Real-Time Tasks with Stochastic Execution Demand. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, volume 1, pages 207–217, August 2011. doi:10.1109/RTCSA.2011.30.

114 S. Milutinovic, J. Abella, D. Hardy, E. Quiñones, I. Puaut, and F. J. Cazorla. Speeding up Static Probabilistic Timing Analysis. In *Proceedings of the International Conference on the Architecture of Computing Systems (ARCS)*, pages 236–247, March 2015. doi:10.1007/978-3-319-16086-3_19.

115 N. Tchidjo Moyo, E. Nicollet, F. Lafaye, and C. Moy. On Schedulability Analysis of Non-cyclic Generalized Multiframe Tasks. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 271–278, July 2010. doi:10.1109/ECRTS.2010.24.

116 N. Navet, L. Cucu, and R. Schott. Probabilistic Estimation of Response Times Through Large Deviations. In *Work-in Progress of the 28th IEEE Real-Time Systems Symposium (RTSS'2007 WiP)*, Tucson, United States, December 2007. URL: https://hal.inria.fr/inria-00191163.

117 N. Navet, Y.-Q. Song, and F. Simonot. Worst-case Deadline Failure Probability in Real-time Applications Distributed over Controller Area Network. *J. Syst. Archit.*, 46(7):607–617, April 2000. doi:10.1016/S1383-7621(99)00016-8.

118 N. Nissanke, A. Leulseged, and S. Chillara. Probabilistic performance analysis in multiprocessor scheduling. *Computing Control Engineering Journal*, 13(4):171–179, August 2002. doi:10.1049/cce:20020403.

119 T. Nolte, H. Hansson, and C. Norstrom. Probabilistic worst-case response-time analysis for the controller area network. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 200–207, May 2003. doi:10.1109/RTTAS.2003.1203052.

120 L. Palopoli, L. Abeni, and D. Fontanelli. A tool for the optimal design of soft real–time systems. In *Proceedings of International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 31–36, 2014.

121 L. Palopoli, D. Fontanelli, L. Abeni, and B. V. Frías. An Analytical Solution for Probabilistic Guarantees of Reservation Based Soft Real-Time Systems. *IEEE Transactions on Parallel and Distributed Systems*, 27(3):640–653, March 2016. doi:10.1109/TPDS.2015.2416732.

122 L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni. An Analytical Bound for Probabilistic Deadlines. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 179–188, July 2012. doi:10.1109/ECRTS.2012.19.

123 S. S. Panwar, D. Towsley, and J. K. Wolf. Optimal Scheduling Policies for a Class of Queues with Customer Deadlines to the Beginning of Service. *J. ACM*, 35(4):832–844, October 1988. doi:10.1145/48014.48019.

124 S. Punnekkat, R. I. Davis, and A. Burns. Sensitivity analysis of real-time task sets. In *Annual Asian Computing Science Conference*, pages 72–82. Springer Berlin Heidelberg, 1997.

125 S. Quinton, R. Ernst, D. Bertrand, and P. Meumeu Yomsi. Challenges and new trends in probabilistic timing analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 810–815, March 2012. doi:10.1109/DATE.2012.6176605.

126 P. Ramanathan and M. Hamdaoui. A Dynamic Priority Assignment Technique for Streams with

(M, K)-Firm Deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, December 1995. doi:10.1109/12.477249.

127 K. S. Refaat and P. E. Hladik. Efficient Stochastic Analysis of Real-Time Systems via Random Sampling. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 175–183, July 2010. doi:10.1109/ECRTS.2010.29.

128 J. Ren, R. Bi, X. Su, Q. Liu, G. Wu, and G. Tan. Workload-aware harmonic partitioned scheduling for probabilistic real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 213–218, March 2018. doi:10.23919/DATE.2018.8342005.

129 L. Santinelli. Probabilistic Component-based Analysis for Networks: Invited Paper. *SIGBED Rev.*, 13(3):65–72, August 2016. doi:10.1145/2983185.2983197.

130 L. Santinelli and L. Cucu-Grosjean. Toward Probabilistic Real-time Calculus. *SIGBED Rev.*, 8(1):54–61, March 2011. doi:10.1145/1967021.1967028.

131 L. Santinelli and L. Cucu-Grosjean. A Probabilistic Calculus for Probabilistic Real-Time Systems. *ACM Transactions on Embedded Computing Systems*, 14(3):52:1–52:30, April 2015. doi:10.1145/2717113.

132 L. Santinelli and L. George. Probabilities and Mixed-Criticalities: the Probabilistic C-Space. In *Proceedings of Workshop on Mixed Criticality (WMC)*, 2015.

133 L. Santinelli, F. Guet, and J. Morio. Revising Measurement-Based Probabilistic Timing Analysis. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, April 2017.

134 L. Santinelli, Z. Guo, and L. George. Fault-aware sensitivity analysis for probabilistic real-time systems. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 69–74, September 2016. doi:10.1109/DFT.2016.7684072.

135 L. Santinelli, J. Morio, G. Dufour, and D. Jacquemart. On the Sustainability of the Extreme Value Theory for WCET Estimation. In *Proceedings of the Workshop on Worst-Case Execution Time Analysis (WCET)*, pages 21–30, 2014. doi:10.4230/OASIcs.WCET.2014.21.

136 L. Santinelli, P. M. Yomsi, D. Maxim, and L. Cucu-Grosjean. A component-based framework for modeling and analyzing probabilistic real-time systems. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2011. doi:10.1109/ETFA.2011.6059013.

137 L. Sha, T. Abdelzaher, K-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *RTSJ*, 28(2-3):101–155, 2004.

138 M. Short and J. Proenza. Towards Efficient Probabilistic Scheduling Guarantees for Real-Time Systems Subject to Random Errors and Random Bursts of Errors. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 259–268, July 2013. doi:10.1109/ECRTS.2013.35.

139 B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for Hard-Real-Time systems. *Springer Real-Time Systems*, 1(1):27–60, 1989. doi:10.1007/BF02341920.

140 B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. Probabilistic Timing Analysis for the Dynamic Segment of FlexRay. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 135–144, July 2013. doi:10.1109/ECRTS.2013.24.

141 B. Tanasa, U. D. Bordoloi, P. Eles, and Z. Peng. Probabilistic Response Time and Joint Analysis of Periodic Tasks. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 235–246, July 2015. doi:10.1109/ECRTS.2015.28.

142 L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century*, volume 4, pages 101–104 vol.4, May 2000. doi:10.1109/ISCAS.2000.858698.

143 T. S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L. C. Wu, and J. W. S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 164–173, May 1995. doi:10.1109/RTTAS.1995.516213.

144 S. R. S. Varadhan. Large deviations. *The Annals of Probability*, 36(2):397–419, March 2008. doi:10.1214/07-AOP348.

145 S. Vestal. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007. doi:10.1109/RTSS.2007.47.

146 B. Villalba Frías, L. Palopoli, L. Abeni, and D. Fontanelli. The PROSIT tool: Toward the optimal design of probabilistic soft real-time systems. *Software: Practice and Experience*, 0(0), 2018. doi:10.1002/spe.2604.

147 G. von der Brüggen, N. Piatkowski, K-H. Chen, J. J. Chen, and K. Morik. Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, volume 106, pages 6:1–6:22, 2018. doi:10.4230/LIPIcs.ECRTS.2018.6.

148 T. Wang, S. Homsi, L. Nui, S. Ren, O. Bai, G. Quan, and M. Qiu. Harmonicity Aware Task Partitioning for Fixed Priority Scheduling of Probabilistic Real-Time Tasks on Multi-Core Platforms. *ACM Transactions on Embedded Computing Systems*, 2016.

149 T. Wang, L. Niu, S. Ren, and G. Quan. Multi-core fixed-priority scheduling of real-time tasks with statistical deadline guarantee. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1335–1340, March 2015.

150 F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 241–248, June 2013. doi:10.1109/SIES.2013.6601497.

151 R. C. Williamson and T. Downs. Probabilistic arithmetic. I. Numerical methods for calculating convolutions and dependency bounds. *International Journal of Approximate Reasoning*, 4(2):89–158, 1990. doi:10.1016/0888-613X(90)90022-T.

152 M. H. Woodbury and K. G. Shin. Evaluation of the probability of dynamic failure and processor utilization for real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 222–231, December 1988. doi:10.1109/REAL.1988.51117.

153 H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Stochastic Analysis of CAN-Based Real-Time Automotive Systems. *IEEE Transactions on Industrial Informatics*, 5(4):388–401, November 2009. doi:10.1109/TII.2009.2032067.

154 H. Zhu, J. P. Hansen, J. P. Lehoczky, and R. Rajkumar. Optimal partitioning for quantized EDF scheduling. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 212–222, 2002. doi:10.1109/REAL.2002.1181576.

Elastic Scheduling for Parallel Real-Time Systems*

James Orr
Washington University in St. Louis, 1 Brookings Dr, St. Louis, MO 63130, USA
james.orr@wustl.edu

Chris Gill
Washington University in St. Louis, 1 Brookings Dr, St. Louis, MO 63130, USA
cdgill@wustl.edu

Kunal Agrawal
Washington University in St. Louis, 1 Brookings Dr, St. Louis, MO 63130, USA
kunal@wustl.edu

Jing Li
New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
jingli@njit.edu

Sanjoy Baruah
Washington University in St. Louis, 1 Brookings Dr, St. Louis, MO 63130, USA
baruah@wustl.edu

Abstract

The elastic task model was introduced by Buttazzo et al. in order to represent recurrent real-time workloads executing upon uniprocessor platforms that are somewhat flexible with regards to timing constraints. In this work, we propose an extension of this model and apply it to represent recurrent real-time workloads that exhibit internal parallelism and are executed on multiprocessor platforms. In our proposed extension, the *elasticity coefficient* – the quantitative measure of a task's elasticity that was introduced in the model proposed by Buttazzo et al. – is interpreted in the same manner as in the original (sequential) model. Hence, system developers who are familiar with the elastic task model in the uniprocessor context may use our more general model as they had previously done, now for real-time tasks whose computational demands require them to utilize more than one processor.

2012 ACM Subject Classification Software and its engineering → Real-time schedulability, Computer systems organization → Real-time system architecture, Computer systems organization → Real-time system specification, Computer systems organization → Embedded software

Keywords and Phrases Parallel real-time tasks, multiprocessor federated scheduling, elasticity coefficient

Digital Object Identifier 10.4230/LITES-v006-i001-a005

Received 2018-09-24 **Accepted** 2019-03-08 **Published** 2019-05-14

1 Introduction

Advances in parallel real-time scheduling theory and concurrency platforms over the last couple of decades have allowed for previously unachievable combinations of high computational demands and fine-grained time-scales, in high-performance parallel real-time applications such as those in autonomous vehicles [13] and real-time hybrid simulation systems [9, 11]. However, current parallel real-time systems usually assign parallel tasks to fixed sets of processors and release them at statically determined periodic rates [13, 9, 10]. For systems that need to adjust individual tasks' computational requirements at run-time (e.g., control algorithms with multiple modes of

* This research was supported in part by NSF grant CCF-1337218 titled "XPS: FP: Real-Time Scheduling of Parallel Tasks" and NSF CNS1814739 titled "Dynamically Customizable Safety Critical Embedded Systems."

operation), current approaches may need to incorporate excessive pessimism to support such forms of dynamic and adaptive resource allocation.

The *elastic task model* was introduced in [4] with the specific aim of providing dynamic flexibility during run-time. The model is derived from an analogy to the expansion and contraction of a contiguous collection of springs when a common force is applied to them all, in order to bring their cumulative length down below a specified bound. The computational demand of a task is analogous to the length of a spring, and the available computational capacity to the bound on the cumulative length of the springs (see [4] for details).

In the elastic task model, each recurrent task is characterized by a worst-case execution time (WCET), lower and upper bounds on the values that the task period parameter may take, and an '*elasticity coefficient*' that represents the flexibility of the task (relative to other tasks) to reduce its run-time computational demand by increasing its effective period. Given a system comprising a collection of such tasks executing upon a shared processor, the elastic scheduling algorithm seeks to choose a value for each task's period parameter within the task's specified range, such that the overall system is schedulable.

The elastic task model was originally defined for task systems such as multimedia systems, control systems, and ad-hoc communication networks implemented on preemptive uniprocessors [1, 8, 5]. However, today's high-performance real-time applications (e.g. real-time hybrid simulation [9, 11]) must often execute upon multiprocessor platforms so as to be able to exploit internal parallelism of these tasks across multiple processors to meet high computational demand. Therefore, the original elastic task model, as well as algorithms that were developed by Buttazzo et al. [4, 5] along with accompanying schedulability analysis and run-time scheduling techniques, need to be appropriately extended in order to be useful for these kinds of high-performance real-time applications. In this paper, we consider multiprocessor scheduling under the *federated scheduling* paradigm (in which each task whose computational demand exceeds the capacity of a single processor is granted exclusive access to multiple processors); we propose a parallel multiprocessor extension to the elastic task model, and provide appropriate algorithms for federated schedulability analysis and federated scheduling of systems represented using our proposed model.

The central idea of elastic scheduling, originally defined by Buttazzo et al. [4], is that if the overall computational demand of a system exceeds the capacity of the implementation platform to accommodate it all, then individual tasks' computational demands are reduced and the available platform capacity is allocated in a flexible manner to accommodate these reduced demands. Upon multiprocessor platforms, there are several different interpretations possible, as to what an *elastic* manner of distributing the processors may mean. Our proposed extension aligns with earlier work in the sense that we are interpreting the elasticity coefficient parameters according to the semantics assigned to them in the uniprocessor context. We believe that this is a critical issue: the elasticity parameters characterize the relative flexibility –the 'hard-real-time'ness– of the tasks, and should bear common interpretation regardless of whether implemented on uni- or multi-processors. We, therefore, believe that the preservation of this interpretation is one of the major benefits of our extended model.

The remainder of this paper is organized in the following manner. We briefly provide some needed background and related work concerning the elastic task model and federated scheduling in Section 2; and describe the parallel workload model we are proposing for the representation of parallel elastic tasks. In Section 3 we present a relatively simple and efficient algorithm for scheduling such tasks upon multiprocessor platforms, which preserves the semantics that were intended for elastic tasks in the uniprocessor context. We also point out how this simple approach may result in an unnecessary degree of platform resource under-utilization. In Section 4 we propose an alternative approach that is able to make more efficient use of the platform to provide

a superior scheduling solution, at the cost of not being as faithful to the semantics of elasticity as originally defined for the uniprocessor case. We conclude in Section 5 with a brief summary, and place this work within a larger context of ongoing research efforts towards achieving dynamic flexibility in multiprocessor scheduling of parallelizable workloads.

2 Background, Related Work, and Task Model

In this paper, we extend the definition and applicability of real-time elastic scheduling to parallel real-time systems. We start out in this section by providing some background on both the elastic task model and the federated paradigm of parallel real-time scheduling on multiprocessor platforms. Doing so enables us to define our proposed elastic model for the federated scheduling of systems of parallel real-time tasks.

2.1 The Elastic Task Model

The elastic task model was first proposed by Buttazzo et al. in [4]. Tasks in this model may dynamically adapt their periods in response to system behavior, in order to keep system-wide utilization below a user-specified desired value U_d (which may be at or below a scheduling algorithm's threshold, e.g., 1.0 for preemptive uniprocessor EDF scheduling). The task model is a generalization of the implicit-deadline sporadic task model [15]: each task $\tau_i = (C_i, T_i^{(\min)}, T_i^{(\max)}, E_i)$ is characterized by a worst-case execution requirement C_i, a minimum (and preferred) period $T_i^{(\min)}$, a maximum period $T_i^{(\max)}$, and an elastic coefficient E_i that quantitatively characterizes how amenable a task is to a change in its period (similar to a measure of a spring's resistance to changes in length). A higher elastic coefficient implies a more elastic task, which is more willing to adapt its period. Any task τ_i that should not vary its period (and therefore its utilization) at all can set $T_i^{(\min)} = T_i^{(\max)}$, and τ_i will act like an ordinary (i.e., not elastic) implicit-deadline sporadic task with WCET C_i and period $T_i^{(\min)}$. An actual period must be assigned to each task; a task's assigned period is denoted as T_i and must fall within the range $[T_i^{(\min)}, T_i^{(\max)}]$. Furthermore, a task τ_i is considered to have an *implicit deadline* where the *relative deadline* D_i of τ_i is equal to its actual period, i.e., $D_i{=}T_i$.

Recall that the *utilization* U_i of an (ordinary – not elastic) implicit-deadline task $\tau_i = (C_i, T_i)$ is defined to be the ratio of its WCET to its period ($U_i = C_i/T_i$), and that the utilization $U(\Gamma)$ of an implicit-deadline sporadic task system $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is the sum of the utilizations of all the tasks in the system ($U(\Gamma) = \sum_{\tau_i \in \tau} U_i$). Buttazzo et al. have derived an iterative algorithm in [4] for task compression which (if possible) finds a way to assign each task τ_i in a system Γ of elastic tasks a period T_i in a manner that is compliant with the semantics of spring compression, such that $\sum_i (C_i/T_i) \leq U_d$ and $T_i^{(\min)} \leq T_i \leq T_i^{(\max)}$ for all tasks τ_i. (As stated above, U_d is a user-defined threshold, perhaps according to the scheduling algorithm that is used, e.g., $U_d = 1$ is suitable for preemptive EDF scheduling.)

Since the introduction of the elastic task model, the uniprocessor version has been expanded to include constrained deadlines [8], resource sharing [5] and unknown computational load [6]. We leave their parallel extensions as future work.

2.2 Federated Scheduling and Parallel Real-Time Task Model

Federated scheduling is a parallel real-time scheduling paradigm that was proposed by Li et al. [14] for scheduling collections of recurrent parallel tasks upon multiprocessor platforms, when one or more individual tasks may have a computational requirement that exceeds the capacity of a single processor to entirely accommodate it. Under federated scheduling, such tasks (i.e., those with

computational requirement exceeding the capacity of a single processor) are granted exclusive access to a subset of processors; the remaining tasks execute upon a shared pool of processors.

In parallel real-time task systems, the computational requirement of a task τ_i (the generalization of the WCET parameter for sequential tasks) is represented by the following two parameters:

1. The ***work*** parameter C_i denotes the cumulative worst-case execution time of all the parallel branches that are executed across all processors. Note that for deterministic parallelizable code (e.g., as represented in the sporadic DAG tasks model [2]; see [3, Chapter 21] for a textbook description) this is equal to the worst-case execution time of the code on a single processor (ignoring communication overhead from synchronizing processors).
2. The ***span*** parameter L_i denotes the maximum cumulative worst-case execution time of any sequence of precedence-constrained pieces of code. It represents a lower bound on the duration of time the code would take to execute, regardless of the number of processors available.
   The span of a program is also called the *critical-path length* of the program, and a sequence of precedence-constrained pieces of code with cumulative worst-case execution time equal to the span is a *critical path* through the program.

Algorithms are known for computing the *work* and *span* of a task represented as a DAG, in time linear in the DAG representation. The relevance of these two parameters arises from well-known results in scheduling theory concerning the multiprocessor scheduling of precedence-constrained jobs (i.e., DAGs) to minimize makespan. This problem has long been known to be NP-hard in the strong sense [16]; i.e., computationally highly intractable. However, Graham's *list scheduling* algorithm [12], which constructs a work-conserving schedule by executing at each instant in time an available job, if any are present, upon any available processor, performs fairly well in practice.

An upper bound on the makespan of a schedule generated by list scheduling is easily stated. Given the *work* and *span* of the DAG being scheduled, it has been proved in [12] that the makespan of the schedule for a given DAG upon m processors is guaranteed to be no larger than

$$\frac{work - span}{m} + span \tag{1}$$

Thus, a good upper bound on the makespan of the list-scheduling generated schedule for a DAG may be stated in terms of only its work and span parameters. Equivalently, if the DAG represents a real-time piece of code characterized by a relative deadline parameter D, $(\frac{work-span}{m} + span) \leq D$ is a sufficient test for determining whether the code will complete by its deadline upon an m-processor platform.

A parallel task τ_i is considered to be a *high-utilization task* if its *utilization* $U_i = \frac{C_i}{T_i} > 1$ and a *low-utilization task* otherwise. Each high-utilization task τ_i receives m_i dedicated processors on which to run; for implicit-deadlines tasks, we need the resulting makespan to be less than or equal to $D_i = T_i$; i.e.

$$\begin{aligned} & \frac{C_i - L_i}{m_i} + L_i \leq T_i \\ \Leftrightarrow \quad & \frac{C_i - L_i}{m_i} \leq T_i - L_i \\ \Leftrightarrow \quad & m_i \geq \frac{C_i - L_i}{T_i - L_i} \end{aligned}$$

Under federated scheduling, since the number of processors assigned to each high-utilization task is an integer, we therefore have

$$m_i = \left\lceil \frac{C_i - L_i}{T_i - L_i} \right\rceil . \tag{2}$$

Under the original federated scheduling model in [14], low-utilization tasks are treated as sequential and are scheduled using existing mechanisms such as global or partitioned EDF scheduling.

In this paper, we will consider the federated scheduling of task systems with elastic sporadic parallel tasks. Recall that each elastic task has a range of acceptable periods within the rage $[T_i^{(\min)}, T_i^{(\max)}]$. Let $U_i^{(\max)} = C_i/T_i^{(\min)}$ and $U_i^{(\min)} = C_i/T_i^{(\max)}$ denote the maximum (i.e., desired) and the minimum acceptable utilization for τ_i. Note that it is possible for some tasks to be either high-utilization or low-utilization depending on the selected period. We refer to these as tasks with *hybrid-utilization*. (Formally hybrid-utilization tasks are tasks such that $T^{(\min)} \leq C_i \leq T^{(\max)}$.) Scheduling of exclusively low-utilization elastic tasks is easily done via minor extensions to prior results [4, 5, 7, 8]. We therefore do not consider them for the remainder of this paper. Instead, *henceforth we consider only the scheduling of exclusively high-utilization tasks*. That is, we will consider a system $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n elastic parallel high-utilization tasks that is to be scheduled under federated scheduling upon m processors. We consider this to be a necessary and non-trivial step towards the scheduling of hybrid-utilization tasks, the treatment of which we leave for future work.

In the remainder of this paper we will often represent a task $\tau_i = (C_i, L_i, U_i^{(\max)}, U_i^{(\min)}, E_i)$ by its work and span parameters, its maximum and minimum utilizations,[1] and its elasticity coefficient. We will seek to compute m_i, the number of processors that are to be devoted to the exclusive use of task τ_i, for each τ_i such that $\sum_{i=1}^{n} m_i \leq m$.

3 A first attempt at elastic scheduling of parallel tasks

It is fairly straightforward to show that the desired elasticity property on the tasks that were defined in the original (uniprocessor) elastic tasks model [4] is that

$$\forall\ i, j, \quad \left(\frac{U_i^{(\max)} - U_i}{E_i}\right) = \left(\frac{U_j^{(\max)} - U_j}{E_j}\right) \tag{3}$$

That is, the elasticity coefficient E_i of task τ_i is a scaling factor on the amount by which it may have its actual utilization reduced from the desired value of $U_i^{(\max)}$.

We use λ to denote the desired equilibrium value for all tasks demonstrated in Expression (3); for all tasks $\lambda = \left((U_i^{(\max)} - U_i)/E_i\right)$. Expression (3) suggests that

$$U_i \leftarrow U_i^{(\max)} - \lambda E_i$$

However, we also require $U_i \geq U_i^{(\min)}$; hence for a given value of λ we choose

$$U_i(\lambda) \leftarrow \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)}\right) \tag{4}$$

Equation (4) suggests an algorithm for the federated scheduling of parallel task system $\Gamma = \{\tau_1, \ldots, \tau_n\}$ upon m processors. It is evident from visual inspection of Equation (4) that the 'best' schedule – the one that compresses tasks' utilizations the least amount necessary in order to achieve schedulability – is the one for which λ is the smallest. Now for a given value of λ, Algorithm 1 can determine, in time linear in the number of tasks, whether the task system can be scheduled upon the m available processors using federated scheduling.

[1] Note that representing the task by its maximum and minimum utilizations is equivalent to representing it by its minimum and maximum periods, since given C_i, one set of parameters can be derived from the other set.

Algorithm 1 Elastic-1(Γ, m, λ).

▷ Γ is the task system and m the number of processors that are available
▷ λ is the compression factor permitted
$m' \leftarrow 0$ ▷ Number of processors needed
for ($\tau_i \in \Gamma$) **do**
  $U_i = \max\left(U_i^{(\max)} - \lambda E_i, U_i^{(\min)}\right)$ ▷ See Eqn 4
  $T_i = C_i/U_i$
  $m_i = \lceil (C_i - L_i)/(T_i - L_i) \rceil$
  $m' \leftarrow m' + m_i$
end for
if ($m' > m$) **then** ▷ Not enough processors.
  return UNSCHEDULABLE
else
  return $\langle m_1, m_2, \ldots, m_n \rangle$ ▷ τ_i gets m_i processors
end if

Note the value of λ can be bounded to the range of $[0, \phi]$ where $\lambda = 0$ represents all tasks receiving their maximum utilizations and ϕ is the maximum value among all tasks of the equation $\left(\frac{U_i^{(\max)} - U_i^{(\min)}}{E_i}\right)$. $\lambda = \phi$ thus represents all tasks receiving their minimum utilization. By bounding the potential values of λ, we can use binary search within this range and make repeated calls to Algorithm 1 and thereby determine, to any desired degree of accuracy, the smallest value of λ for which the system is schedulable.

3.1 Discussion

Semantics-preservation. Algorithm 1 for the federated scheduling of parallel elastic tasks that we have presented above is *semantics preserving* in the following sense: the assignment of actual period values to the tasks (the T_i's) is done in accordance with Equation (4), which is the same manner in which periods are assigned in uniprocessor scheduling of elastic tasks. Hence the system developer who seeks to use our proposed elastic task model to implement flexible parallel tasks upon multiprocessor platforms need not 'learn' new (or additional) semantics for the elasticity coefficient: this coefficient means exactly the same thing in the parallel multiprocessor case as it did in the system designer's previous experiences with sequential uniprocessor tasks (the value of this parameter for each task is a relative measure of its degrees of tolerance to having its period increased and its computational demand thereby reduced).

Run-time platform capacity under-utilization. Despite these advantages, however, one can identify two sources of resource under-utilization by Algorithm 4.

- First, observe that the number of processors assigned to a task must be *integral*, and is hence equal to the ceiling of an expression. If the expression $(C_i - L_i)/(T_i - L_i)$, which lies within the ceiling operator ($\lceil \cdot \rceil$) when computing the number of processors assigned to task τ_i, is not itself an integer, then one could further reduce the actual period (the T_i value) that is assigned to the task τ_i and thereby assign τ_i more computational capacity than is afforded by Algorithm 1. However, we do not permit this to happen since the resulting assignment may no longer be semantics-preserving in the sense that different tasks may see a reduction in allocated capacity that is not consistent with their relative elasticity coefficients. This difference between $\lceil (C_i - L_i)/(T_i - L_i) \rceil$ and $(C_i - L_i)/(T_i - L_i)$ is thus 'wasted' capacity.

- Second, consider the case with two identical elastic tasks, and an odd number of processors. Semantics-preservation dictates that both tasks be treated in the same manner; however, doing so would correspond to assigning the same number of processors to each task and therefore leaving one processor unused. More generally, Algorithm 1 may leave up to $n-1$ processors unallocated to n identical tasks.

Thus, the simple semantics-preserving scheme presented in this section may under-utilize platform resources. In Section 4 we discuss an alternative scheme that makes more efficient use of platform capacity at the cost of additional complexity in the semantics of elasticity.

4 More resource-efficient scheduling

The notion of semantic preservation with uniprocessor elastic task scheduling presented in Section 3 is simple and intuitive, and very strong: the elasticity coefficient of a task directly indicates the task's tolerance to having its period parameter increased. However, as we saw, remaining faithful to such a strong notion of semantic equivalence comes at the cost of some computing capacity loss and cannot guarantee full utilization of a platform's computing capacity. We now consider a more generalized interpretation of the semantics of uniprocessor elastic tasks. This interpretation was provided by Chantem et al. [8], who proved that the algorithm of Buttazzo et al. [5] for scheduling sequential elastic tasks upon preemptive uniprocessors is equivalent to solving the following constrained optimization problem:

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left(U_i^{(\max)} - U_i\right)^2 \tag{5}$$

such that:

$$U_i^{(\min)} \leq U_i \leq U_i^{(\max)} \quad \text{for all } \tau_i, \text{ and}$$
$$\sum_{i=1}^{n} U_i \leq U_d$$

where U_d is the desired system utilization. We believe that this is a somewhat less natural interpretation of elasticity in task scheduling than the interpretation considered in Section 3: it is unlikely that a typical system designer is thinking of the elasticity coefficients (the E_i parameters) that they assign to the individual tasks as coefficients to a quadratic optimization problem. Nevertheless, we adopt this notion of elastic interpretation in this section; for this interpretation, we are able to derive a federated scheduling algorithm that makes far more efficient use of platform computing capacity than was possible under the earlier more intuitive interpretation considered in Section 3.

Note that sequential elastic task scheduling only considers CPU utilization when attempting to schedule tasks on a single processor. Specifically, system-wide utilization $\sum_{i=1}^{n} U_i$ must stay below a desired utilization U_d at all times in order to maintain schedulability. As such, task utilizations are decreased by (when possible) increasing individual task periods in proportion to their fraction of system-wide elasticity until either (1) an acceptable schedule is found such that $\sum_{i=1}^{n} U_i \leq U_d$ or (2) each task τ_i has period $T_i = T_i^{(\max)}$ with $\sum_{i=1}^{n} U_i > U_d$. If a schedule cannot be found the taskset is declared unschedulable.

In federated scheduling of high-utilization tasks, however, system schedulability is no longer a function only of cumulative utilization but rather whether n tasks can be successfully scheduled on m cores. We now give an algorithm for determining processor allocation and schedulability of a task system that allocates the processors *one at a time* to the tasks, Algorithm 2. Algorithm 2

Algorithm 2 Task_compress_par(Γ, m).

for ($\tau_i \in \Gamma$) **do**
  $m_{i_{min}} = \lceil (C_i - L_i)/T_{i_{max}} - L_i) \rceil$ ▷ Minimum number of processors
  $m_{i_{max}} = \lceil (C_i - L_i)/(T_{i_{min}} - L_i) \rceil$ ▷ Maximum number of processors
  $m_i = m_{i_{min}}$
  while $m_i <= m_{i_{max}}$ **do** ▷ Compute the shortest period for τ_i
    ▷ for each possible value of m_i
    $T_{(i,m_i)} = (C_i - L_i)/(m_i) + L_i$ ▷ $T_{(i,m_i)}$ = shortest with m_i processors
    $m_i = m_i + 1$
  end while
  $m_i = m_{i_{min}}$ ▷ Assign minimum number of processors
  $T_i = T_{(i,m_i)}$ ▷ Assign corresponding shortest period
  $m = m - m_{i_{min}}$ ▷ m keeps count of processors remaining
end for
if ($m < 0$) **then** ▷ There weren't enough processors
  return UNSCHEDULABLE
else if ($m == 0$) **then**
  return *processor allocation with m_i values*
end if

The remainder of this pseudocode
allocates processors one at a time

for ($\tau_i \in \Gamma$) **do**
  Determine δ_i, the potential
  decrease to Problem 7 for each task
end for

Make a max heap of all tasks, with the δ_i values as the key

while m > 0 and heap not empty **do** ▷ Assign remaining processors
  $\tau_{most} = heap.pop()$ ▷ Task that would most benefit
  $m_{most} = m_{most} + 1$ ▷ Permanently assign processor
  $m = m - 1$
  $T_{most} = T_{(most,m_{most})}$
  if ($m > 0$ and $m_{most} < m_{most_{max}}$) **then** ▷ Able to receive more processors?
    Determine δ_{most}, the potential
    decrease to Problem 7 for task τ_{most}
    Reinsert τ_{most} into heap
  end if
end while
return *the processor allocation with m_i values*

starts out by determining, for each task τ_i, the minimum number of processors $m_{i_{min}}$ needed to be meet its minimum acceptable computational load (i.e., having $T_i \leftarrow T_i^{(\max)}$) in Line 2, and the number $m_{i_{max}}$ needed to meet its desired computational load (i.e., having $T_i \leftarrow T_i^{(\min)}$) in Line 3. Since the assigned period T_i satisfies $T_i^{(\min)} \leq T_i \leq T_i^{(\max)}$, the actual number of CPUs m_i assigned to τ_i is also bounded by $m_{i_{min}} \leq m_i \leq m_{i_{max}}$.

Because of the ceiling function in Equation (2), each range of values for T_i maps to a given m_i for each task. In this work we assume that it is beneficial for each task to run as frequently as possible. As such, we assign task τ_i the minimum period T_i available on m_i allocated processors. We denote this period value as $T_{(i,m_i)}$, which is derived directly from Equation (2):

$$T_{(i,m_i)} = \frac{C_i - L_i}{m_i} + L_i \tag{6}$$

All possible values of $T_{(i,m_i)}$ for $m_{i_{min}} \leq m_i \leq m_{i_{max}}$ are computed first and stored in lookup tables. This is accomplished during the while loop (Lines 5–9) in Algorithm 2.

Next (Lines 10–12), each task is assigned the minimum number of processors it needs, and this number of processors is subtracted from m; hence at the end of the loop, m denotes the number of processors remaining for additional assignment (above and beyond the minimum needed per task). If $m < 0$ the instance is unschedulable, while if $m = 0$ there is nothing more to be done – the system is schedulable with each task receiving its minimum level of service. These conditions are tested in Lines 14–18 of the pseudocode.

If $m > 0$, however, we will individually assign each of these remaining m processors to whichever task would benefit 'the most' from receiving it. This is determined in the following manner. Similar to scheduling sequential tasks [8], our goal is to find task utilizations (and therefore periods) that solve the optimization problem:

$$\textbf{minimize} \sum_{i=1}^{n} \frac{1}{E_i} \left(U_i^{(\max)} - U_i\right)^2 \tag{7}$$

such that:

$$U_i^{(\min)} \leq U_i \leq U_i^{(\max)} \text{ for all } \tau_i, \text{ and}$$
$$\sum_{i=1}^{n} m_i \leq m$$

In allocating each processor we calculate, for each task τ_i, a quantity δ_i which represents the *decrease* in $\frac{1}{E_i}\left(U_i^{(\max)} - U_i\right)^2$ if the next processor were to be allocated to task τ_i – this is done in Lines 23–26 of Algorithm 2. We then assign the processor to whichever task would see the biggest decrease. (As a consequence, the objective of the optimization problem 7 would decrease the most.) To accomplish this efficiently, we

- Place the tasks in a max heap indexed on the value of δ_i (Line 28); and
- while there are unallocated processors and the heap is not empty (checked in Line 30)
  - assign the next processor to the task at the top of the heap (Lines 31–34) and, if this task is eligible to receive more processors (checked in Line 35), recompute δ_i for this task (Line 36) and reinsert into the heap (Line 38).

Run-time complexity. The first for-loop in the algorithm (Lines 1–13 in the pseudocode listing in Algorithm 2) takes $\Theta(m{*}n)$ time. The for-loop in Lines 23–26 and the making of the max heap (Line 28) each take $\Theta(n)$ time. The running time of the remainder of the algorithm (Lines 30–40) is dominated by the max-heap operations; the overall running time is therefore $\Theta(n * m + m \log n)$.

4.1 Proof of Optimality

In this section we prove in Theorem 3 that Algorithm 2 solves the optimization problem given in Equation (7) optimally. The optimality of Algorithm 2 then follows from the result of Chantem et al. [8] showing the equivalence of uniprocessor elastic scheduling of sequential tasks with the optimization problem given in Equation (5).

The dependency amongst the three results in this section – Lemma 1, Lemma 2, and Theorem 3 – is strictly linear: Lemma 1 is needed to prove Lemma 2, which is needed to prove Theorem 3.

▶ **Lemma 1.** *The utilization U_i of elastic task τ_i strictly increases towards maximum utilization as the number of processors m_i assigned to it increases.*

Proof. Since $U_i = C_i/T_i$, (and C_i is constant), U_i increases as T_i decreases. By Equation (6), $T_i = ((C_i - L_i)/m_i) + L_i$. C_i and L_i are constant for task τ_i. Therefore, T_i strictly decreases as m_i increases. Therefore, an increase of m_i decreases T_i and increases U_i. ◀

▶ **Lemma 2.** *In assigning processors one at a time (in the while loop of Lines 30–40 of Algorithm 2), the consecutive assignment of the $(k+1)$'st and $(k+2)$'nd to the same task τ_i with k currently assigned processors will result in diminishing returns of δ_i, the decrease in $\frac{1}{E_i}(U_i^{(\max)} - U_i)^2$ for τ_i. (i.e., the benefit of assigning a processor to a task is never as high as the already-incurred benefit of assigning prior processors.)*

Proof. This is readily observed by algebraic simplification.[2] Let x_k be the value of $\frac{1}{E_i}(U_i^{(\max)} - U_{i_k})^2$ where U_{i_k} is the task utilization with k processors. Let x_{k+1} be the value of $\frac{1}{E_i}(U_i^{(\max)} - U_{i_{k+1}})^2$ with new utilization $U_{i_{k+1}}$ after assigning processor $k+1$ to τ_i, and similarly let x_{k+2} be the value of $\frac{1}{E_i}(U_i^{(\max)} - U_{i_{k+2}})^2$ with new utilization $U_{i_{k+2}}$ after subsequently assigning processor $k+2$ to τ_i. From Lemma 1, we know that $U_{i_k} < U_{i_{k+1}} < U_{i_{k+2}}$.

Define the benefit of adding processor $k+1$ to τ_i as $\delta_{i_{k+1}} = x_k - x_{k+1}$, and the later benefit of assigning processor $k+2$ as $\delta_{i_{k+2}} = x_{k+1} - x_{k+2}$. To prove diminishing returns, we must show that $\delta_{i_{k+1}} > \delta_{i_{k+2}}$.

Note that the math is equivalent, so we temporarily ignore the constant scalar $\frac{1}{E_i}$. Thus, both

$$\delta_{i_{k+1}} = (U_i^{(\max)} - U_{i_k})^2 - (U_i^{(\max)} - U_{i_{k+1}})^2 \tag{8}$$

and

$$\delta_{i_{k+2}} = (U_i^{(\max)} - U_{i_{k+1}})^2 - (U_i^{(\max)} - U_{i_{k+2}})^2 \tag{9}$$

are of the form

$$(x-z)^2 - (x-y)^2 \tag{10}$$

where $x > y > z$. We can therefore say that $z + \alpha = y$ and $y + \beta = x$.

Re-stating Equation (10) in terms of z, α, and β, we obtain:

$$(z+\alpha+\beta-z)^2 - (z+\alpha+\beta-z-\alpha)^2$$

which simplifies to

$$\alpha^2 + 2\alpha\beta. \tag{11}$$

[2] The algebra, while straightforward, is rather tedious and the reader may choose to just skim it at first reading.

Therefore, to prove $\delta_{i_{k+1}} > \delta_{i_{k+2}}$, it is sufficient to show that

$$\alpha_{k+1}^2 + 2\alpha_{k+1}\beta_{k+1} > \alpha_{k+2}^2 + 2\alpha_{k+2}\beta_{k+2} \tag{12}$$

where α_{k+1}, β_{k+1}, α_{k+2}, and β_{k+2} are $(U_{i_{k+1}} - U_{i_k})$, $(U_i^{(\max)} - U_{i_{k+1}})$, $(U_{i_{k+2}} - U_{i_{k+1}})$, $(U_i^{(\max)} - U_{i_{k+1}})$, respectively. (These values come from the definitions of α and β and the substitutions of x, y, and z in Equation (10) into their actual values from Equations 8 and 9.) Note that as α_{k+1}, β_{k+1}, α_{k+2}, and β_{k+2} are all positive numbers, Equation (12) will be satisfied if we can individually prove $\alpha_{k+1} > \alpha_{k+2}$ and $\beta_{k+1} > \beta_{k+2}$, which we now proceed to do.

We first prove $\beta_{k+1} > \beta_{k+2}$, where

$$\beta_{k+1} = (U_i^{(\max)} - U_{i_{k+1}}),$$

and

$$\beta_{k+2} = (U_i^{(\max)} - U_{i_{k+2}}).$$

We know from above that $U_{i_{k+2}} > U_{i_{k+1}}$. Therefore

$$(U_i^{(\max)} - U_{i_{k+1}}) > \ (U_i^{(\max)} - U_{i_{k+2}})$$

and $\beta_{k+1} > \beta_{k+2}$.

We next prove $\alpha_{k+1} > \alpha_{k+2}$. Note that

$$U_i = \frac{C_i}{T_i = \frac{C_i - L_i}{m_i} + L_i} \tag{13}$$

Consider Equation (13) which shows the complete derivation of a task's utilization as a function of the number of processors assigned to it. By definition, if $\alpha_{k+1} \overset{?}{>} \alpha_{k+2}$, [3] then

$$U_{i_{k+1}} - U_{i_k} \overset{?}{>} U_{i_{k+2}} - U_{i_{k+1}}.$$

Substituting into Equation (13), this becomes

$$\frac{C_i}{\frac{C_i - L_i}{k+1} + L_i} - \frac{C_i}{\frac{C_i - L_i}{k} + L_i} \overset{?}{>} \frac{C_i}{\frac{C_i - L_i}{k+2} + L_i} - \frac{C_i}{\frac{C_i - L_i}{k+1} + L_i}.$$

Factoring out a constant C_i and simplifying, we get

$$\frac{k+1}{C_i + kL_i} - \frac{k}{C_i + kL_i - L_i} \overset{?}{>} \frac{k+2}{C_i + kL_i + L_i} - \frac{k+1}{C_i + kL_i}.$$

Letting $X = C_i + kL_i$ (to enhance readability), this becomes

$$\frac{k+1}{X} - \frac{k}{X - L_i} \overset{?}{>} \frac{k+2}{X + L_i} - \frac{k+1}{X}.$$

We can combine fractions and simplify this further to

$$\frac{-kL_i + X - L_i}{X(X - L_i)} \overset{?}{>} \frac{-kL_i + X - L_i}{X(X + L_i)}.$$

[3] We use $\overset{?}{>}$ to indicate that the inequality is not yet proved.

Since $-k * L_i + X - L_i = -kL_i + C_i + kL_i - L_i = C_i - L_i > 0$ for high-utilization tasks, we can now factor out $-k * L_i + X - L_i$ from both sides and are left with asking whether

$$\frac{1}{X(X - L_i)} \stackrel{?}{>} \frac{1}{X(X + L_i)}.$$

This is unequivocally true. Hence, we prove that $\alpha_{k+1} > \alpha_{k+2}$. Therefore, Equation (12) is satisfied and $\delta_{i_{k+1}} > \delta_{i_{k+2}}$. The Lemma follows. ◀

▶ **Theorem 3.** *Algorithm 2 optimally minimizes the optimization problem given in Equation (7).*

Proof. For Algorithm 2 to be non-optimal, there must be some point at which our greedy algorithm and the optimal algorithm diverge. (Algorithm 2 begins optimally with the only valid assignment of processors to tasks when considering only the minimum amount of processors each task can have.) Note that each task's contribution to the sum of Equation (7) is independent of other tasks: the value of $\frac{1}{E_i}\ (U_i^{(\max)} - U_i)^2$ for a given task τ_i is independent of how many processors have been assigned to other tasks. Thanks to this property, we need only consider two tasks. Let us suppose, without loss of generality, that at the point of divergence our greedy algorithm assigns the processor to τ_i, while the optimal algorithm would assign the processor to τ_j.

Because the greedy algorithm assigns the processor to τ_i, we know that the added benefit (amount decreased from the sum) is greater than if we had given the processor to τ_j. Hence the current value of the objective function of optimization problem 7 the greedy algorithm is necessarily lower than that of the optimal algorithm upon assignment of the number of processors assigned thus far. By the assumption regarding the non-optimality of our greedy strategy, there must be some point in the future at which the optimal algorithm makes up the difference since the optimal solution to a minimization problem must end with the lowest value for the objective function.

However, we saw in Lemma 2 above that the benefits of assigning a new processor under the greedy Algorithm 2 diminish. At each iteration, the greedy algorithm chooses to assign the processor to the task with the greatest available benefit. Because tasks' benefits are considered independently and do not change regardless of the allocation of CPUs to other tasks, after the greedy algorithm assigns the k'th processor to τ_i, no other task τ_j will have a higher benefit of receiving the $(k+1)$'st processor than it did when the greedy algorithm elected to give the k'th processor to τ_i. Similarly, by Lemma 2 the diminishing returns of assigning multiple processors to the same task guarantees that the benefit of assigning the $(k+1)$'st task to τ_i is also less than the benefit gotten by assigning the k'th processor to τ_i. Therefore, if the optimal algorithm and the greedy algorithm diverge and the current value of the objective function of optimization problem 7 for Algorithm 2 is better than the optimal algorithm, it is impossible for the optimal algorithm to subsequently 'catch up' and do better than the greedy algorithm. Hence the current value of the objective function of optimization problem 7 may never diverge between an optimal algorithm and our greedy algorithm; the optimality of Algorithm 2 immediately follows. ◀

This completes the proof of optimality of Algorithm 2 for the federated scheduling of parallel elastic tasks.

5 Summary & Conclusions

In the two decades since it was first introduced, the elastic task model [4] has proved a useful abstraction for representing flexibility in the computational demands of recurrent workloads. It was originally proposed for representing sequential tasks executing upon uniprocessor platforms;

as high-performance real-time computer applications are increasingly becoming parallelizable (and need to have their parallelism exploited by being implemented upon multiprocessor platforms in order to meet timing constraints), there is a need to extend the applicability of the elastic task model to parallel tasks that execute upon multiprocessor platforms.

In this paper, we have proposed one such extension. The salient features of our model are:

- Multiprocessor scheduling under the federated paradigm, in which each task needing more than one processor is assigned exclusive access to all processors upon which it executes. Federated scheduling frameworks can generally be implemented in a more efficient manner than global scheduling (e.g., with less run-time overhead) with only limited loss of schedulability (as measured by speedup bounds of capacity augmentation bounds).
- Representation of a parallel task's workload using just the cumulative workload (its '*work*' parameter) and its critical path length (its '*span*' parameter). Such representation allows for efficient schedulability analysis in the federated scheduling framework, with a bounded loss of schedulability as compared to DAG representations (for which schedulability analysis is strongly NP-hard.
- Retention of the elasticity coefficient parameter that was the main innovation introduced in [4] to capture the flexibility in computational demands.

We have proposed and studied two schemes for assigning processors to tasks in a system of elastic parallel real-time tasks that are to be scheduled upon a given multiprocessor platform under federated scheduling. One of these schemes is completely semantics-preserving with respect to model semantics as introduced in the uniprocessor case [4]; the other allows for some deviation from uniprocessor semantics and thereby is able to better use the computational capabilities of the implementation platform.

Possible future extensions of this work include the scheduling of hybrid-utilization tasks, each of whose potential utilization range is such that it can be treated as either a low-utilization or a high-utilization task, depending on the system load. This necessarily involves the co-scheduling of low-utilization and high-utilization tasks. It may also be worth investigating different ways of scheduling low-utilization tasks on multi-core systems. Buttazzo's algorithms provide an optimal way to schedule a task set of low-utilization tasks on a single processor but say nothing about how to assign low-utilization tasks to multiple processors. Each of these can also be explored with constrained deadlines and resource sharing.

References

1 P. Antsaklis and J. Baillieul. Guest Editorial Special Issue on Networked Control Systems. *IEEE Transactions on Automatic Control*, 49(9):1421–1423, September 2004. doi:10.1109/TAC.2004.835210.

2 Sanjoy Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leem Stougie, and Andreas Wiese. A generalized parallel task model for recurrent real-time processes. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS 2012, pages 63–72, San Juan, Puerto Rico, 2012.

3 Giorgio Buttazzo, Enrico Bini, and Darren Buttle. Rate-Adaptive Tasks: Model, Analysis, and Design Issuess. In *Proceedings of DATE 2014: Design, Automation and Test in Europe*, March 2014.

4 Giorgio C. Buttazzo, Giuseppe Lipari, and Luca Abeni. Elastic Task Model for Adaptive Rate Control. In *1998 IEEE Real-Time Systems Symposium (RTSS)*, 1998.

5 Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Trans. Comput.*, 51(3):289–302, March 2002. doi:10.1109/12.990127.

6 M. Caccamo, G. Buttazzo, and Lui Sha. Elastic feedback control. In *Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000*, pages 121–128, 2000. doi:10.1109/EMRTS.2000.853999.

7 T. Chantem, X. S. Hu, and M. D. Lemmon. Generalized Elastic Scheduling. In *2006 27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 236–245, 2006.

8 T. Chantem, X. S. Hu, and M. D. Lemmon. Generalized Elastic Scheduling for Real-Time Tasks. *IEEE Transactions on Computers*, 58(4):480–495, April 2009. doi:10.1109/TC.2008.175.

9 D. Ferry, G. Bunting, A. Maqhareh, A. Prakash, S. Dyke, K. Aqrawal, C. Gill, and C. Lu. Real-

time system support for hybrid structural simulation. In *2014 International Conference on Embedded Software (EMSOFT)*, pages 1–10, October 2014. doi:10.1145/2656045.2656067.

10 David Ferry, Jing Li, Mahesh Mahadevan, Kunal Agrawal, Christopher Gill, and Chenyang Lu. A Real-time Scheduling Service for Parallel Tasks. In *Proceedings of the 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, RTAS '13, pages 261–272, Washington, DC, USA, 2013. IEEE Computer Society. doi:10.1109/RTAS.2013.6531098.

11 David Ferry, Amin Maghareh, Gregory Bunting, Arun Prakash, Kunal Agrawal, Chris Gill, Chenyang Lu, and Shirley Dyke. On the performance of a highly parallelizable concurrency platform for real-time hybrid simulation. In *The Sixth World Conference on Structural Control and Monitoring*, 2014.

12 R. Graham. Bounds on multiprocessor timing anomalies. *SIAM Journal on Applied Mathematics*, 17:416–429, 1969.

13 J. Kim, H. Kim, K. Lakshmanan, and R. Rajkumar. Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car. In *2013 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pages 31–40, April 2013.

14 Jing Li, Abusayeed Saifullah, Kunal Agrawal, Christopher Gill, and Chenyang Lu. Analysis Of Federated And Global Scheduling For Parallel Real-Time Tasks. In *Proceedings of the 2012 26th Euromicro Conference on Real-Time Systems*, ECRTS '14, Madrid (Spain), 2014. IEEE Computer Society Press.

15 C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

16 J. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.