Learning from Probabilities: Dependences within Real-Time Systems

Alessandra Melani, Eric Noulard and Luca Santinelli
ONERA Toulouse, name.surname@onera.fr

Abstract

Realistic real-time systems experience variability and unpredictabilities, which can be compensated by potentially very pessimistic worst-cases. Recent trends apply measurement-based approaches in modeling worst-cases with a certain confidence. While observing system evolution it is possible to extract probabilistic models to the task execution with a guaranteed probabilistic version of worst-case execution time. In this work we exploit such probabilistic models in order to study the effect of dependences on the task execution time, and we apply the developed probabilistic framework to few relevant cases studies.

1. Introduction

Real-time relies on the Worst-Case Execution Time (WCET) to model task execution behaviors: a real-time system becomes predictable by always accounting for the worst-case at every task execution.

As the input space for the task code is finite and the hardware behavior is assumed to be deterministic, it is reasonable to argue about the exact worst-case execution time. The \( WCET_{exact} \) and its estimation \( \overline{C} \) are upper-bounds for any possible execution behavior of that code [33, 15].

Unfortunately, realistic real-time systems are unpredictable [30]. Their environment can be diverse and dynamic [20], with multiple possible evolutions in time. Both hardware and software elements may experience some variability or even randomness\(^1\), e.g. [multi-]processor cache, branch predictors, DRAM refresh, interruptions that occur whenever they are most inappropriate [12, 6, 17]. Besides, the interferences between interacting elements in the system lead to the dependences that emphasize unpredictabilities and variability. Those are some of the reasons why \( WCET_{exact} \) is in general unknown and potentially unknowable. Consequently, the estimation \( \overline{C} \) could be extremely pessimistic.

Probabilities are becoming a consistent way of representing the time evolution of systems. They are able to model indeterminacy and unpredictabilities that systems have, since a probabilistic representation captures multiple behavior together with their frequencies. Such a more fine grained system representation reduces the pessimism brought by deterministic models, where only the estimation of the worst-case is taken into account. Then, the challenge in real-time is to make predictability out of probabilities [5, 7], and build with the probabilities a safe alternative to the deterministic real-time.

1Randomness intended in the common sense as lack of pattern or predictability in events

Contributions: In this work, we apply probabilistic models and statistical analysis to describe unpredictabilities and dynamics that realistic real-time systems have. We target the dependences that relate system elements and their concurrent execution. We aim to improve the knowledge of the systems characterizing some of their dependences thus reducing the pessimism of their timing models. In doing that, we rely on measurements and rare events for a probabilistic version of the WCET. This would result into more accurate real-time analyses without the need of overlaying pessimistic safe bounds and the consequent waste of resources. To our knowledge, this work is among the firsts to provide safe and sharp probabilistic WCET bounds for realistic systems without the need of complicated and artifact hardware elements and models.

Organization of the paper: In Section 2 we introduce the probabilistic model we intend to apply to characterize tasks execution times and their worst-cases. Section 3 tackles with the dependences within realistic real-time systems and their effect on the task executions. Section 4 and 5 describe both the experiments and their setup we applied to support the theory developed. Section 6 is for the conclusion and the future work.

1.1 Timing Analysis for Real-Time System: the State of the Art

Within the WCET research community a main debate is on the approaches to the WCET estimation, [33].

On one hand there is the use of static analysis [1, 22, 9, 8], applying models of the code and the underlying hardware for \( WCET_{static} \) estimation. Those analytical approaches can of course be in error, however it is usual to guarantee that \( WCET_{exact} < WCET_{static} \) with a possible extra margin added to the \( WCET_{static} \).

On the other hand there are the measurement-based approaches, with a simpler model of the code and the observations of the task execution time, charging the system effects. The \( WCET_{measured} \) is the maximum value observed during measurements, but in general \( WCET_{measured} \leq WCET_{exact} \), due to the not assured exhaustive execution condition coverage and thus variability [3]. Even when \( WCET_{measured} \) is computed using test generation techniques which ensure feasible path coverage, they usually make assumptions which negate variability, [12, 34].

A more “analytical” approach to measurements makes use of some form of extreme value statistical analysis [13, 14, 21] to construct a predicted WCET value, \( WCET_{predicted} \). Recent works have formally approached Extreme Value Theory (EVT) for the WCET problem [11, 7]. They claim that, with a probabilistic hardware architecture and measurement-based approaches, it is possible to guarantee an accurate
WCET\textsubscript{predicted}.

Figure 1 gives a further intuition of the differences between the measurements, the WCET\textsubscript{exact} (exact) and its safe estimation $\overline{C}$ (safe).

2. The Probabilistic Natural Behavior

Due to variability and unpredictability of today’s real-time systems, what emerges as their behavior is close to random processes\textsuperscript{2}. The system indeterminacy is accounted by the many directions the random process may evolve. In particular, we are interested in the probabilistic nature of the task execution times, thus their probabilistic representation. With probabilities, it is possible to enrich the classical notion of worst-case execution time with further information about the multiple ways tasks can execute.

Assuming $C_i$\textsuperscript{3} a distribution of execution times for a task $\tau_i$ as a discrete random variable\textsuperscript{4}, its probabilistic representation through probabilistic distribution function (pdf) is $f_{C_i}$

$$f_{C_i} = \left( \begin{array}{ccc} c_{i}^{0} = c_{i}^{\min} & c_{i}^{1} & \cdots & \cdots & c_{i}^{k_{i}} = c_{i}^{\max}\end{array} \right)$$

and $\sum_{j=0}^{k_i} f_{C_i}(c_{i}^{j}) = 1$. All these parameters are given with the interpretation that task $\tau_i$ generates an infinite number of successive jobs $\tau_{i,j}$, with $j = 1, \ldots, \infty$, and each such job has an execution requirement described by $C_i$, where for each value $C_i$, $f_{C_i}(C_i)$ is its probability of occurrence within the execution streamlines. $P(C_i = c_i)$. The Cumulative Distribution Function (CDF) description is $F_{C_i}(c)$ = $P(C_i \leq c)$. The Cumulative Distribution Function (1-CDF) $F^\prime_{C_i}(c)$ is $F^\prime_{C_i}(c) = 1 - \sum_{x=0}^{c} f_{C_i}(c)$. A representation in terms of 1-CDF outlines the exceedence thresholds as $P(C_i \geq c)$. Figure 2 shows an example of execution time distribution with its histogram representation for the value frequencies, together with the correspondent CDF and 1-CDF representations.

The task model we consider is $\tau_i = (\Phi_i, C_i, T_i, D_i)$, with $\Phi_i$ being the task release time, $C_i$ the distribution of possible execution times and associated probabilities, $T_i$ the maximum task inter-arrival time and $D_i$ its relative deadline.

An Execution Time Profile (ETP) $C_i$ of a task $\tau_i$ is a probability distribution of execution time obtained through exhaustive $\tau_i$ execution time measurements, the empirical distribution. Figure 1 shows an example of the execution time profile (measured) with respect to the possible execution time a task can have (possible).

2.1 Dependence between Random Variables

Most of the algebra in probability theory relies on the degree of dependence between random variables, the so called statistical dependence.

Definition 1 (Independent random variables) Two random variables $X$ and $Y$ are independent if they describe two events such that the occurrence of one event does not have any impact on the occurrence of the other event.

An example is the joint probability, which expresses the composition between random variables. For a couple $X$, $Y$, the joint probability defines the probability of events in terms of both $X$ and $Y$, $P(X = x \text{ and } Y = y)$ given by $P(Y = y|X = x) \cdot P(X = x)$ or equivalently $P(X = x|Y = y) \cdot P(Y = y)$.

In terms of CDF it is

$$F_{X,Y}(x,y) = F_{X|Y}(x|y) \cdot F_Y(y) = F_Y(y) \cdot F_X(x).$$

Whenever there is independence it is $F_{X,Y} = F_X \cdot F_Y$, while in case of dependences between $X$ and $Y$, the joint probability remains in its generic form, Equation (2).

Definition 2 (Dependence $\geq$) Two random variables $X$ and $Y$ are dependent, $X \geq Y$ or equivalently $Y \geq X$, every time there is not independence, thus $F_{X|Y}(x|y) \neq F_X(x)$.

We will examine later how the dependences come into play within our probabilistic framework.

2.2 Probabilistic Worst-Case Execution Time

In a probabilistic scenario, more sound is the notion of probabilistic Worst-Case Execution Time distribution, noted as pWCET, that unlike WCET\textsubscript{X}, it is is a distribution, and not a scalar value. To converge to that notion we need some definitions first.

Definition 3 (Greater than or equal to, $\geq$) Let $X$ and $Y$ be two random variables. $Y$ is greater than or equal to $X$ (or alternatively, $X$ is less than or equal to $Y$) denoted by $Y \geq X$ (alternatively, $Y \leq X$) if $P(Y \leq v) \leq P(X \leq v)$ for any $v$ (alternatively $P(Y \leq v) \geq P(X \leq v)$ for any $v$). In terms of the CDF representation, $Y \geq X$ ($Y \leq X$) if $F_Y \leq F_X$ ($F_Y \geq F_X$). With the 1-CDF representation, $Y \geq X$ ($Y \leq X$) if $F_Y^\prime \leq F_X^\prime$ ($F_Y^\prime \geq F_X^\prime$).

With such partial ordering between random variables we can say that the pWCET distribution is greater than or equal to (meaning worse than but also safer than) the ETPs. A formal definition for the exact pWCET $C^*$ is the following.
Definition 4 (probabilistic Worst-Case Execution Time $C^*_i$)
The pWCET $C^*_i$ of a task $\tau_i$ is defined as the least upper-bound on all the distributions $C^j_i$ of the execution time of $\tau_i$, where $C^j_i$ is generated for every possible combination $j$ of input data to the program by running the program an infinite number of times. Thus $\forall j$, $C^*_i \geq C^j_i$. In terms of CDF $F^j_{C^*_i}(x) = \min_j(F^j_{C^*_i})$, while the 1-CDF is $F^j_{C^*_i}(x) = \max_j(F^j_{C^*_i})$.

$C^*_i$ is guaranteed to exist knowing all the possible ETPs $C^j_i$ for the task $\tau_i$. Unfortunately, measurement approaches cannot ensure the coverage of all the execution conditions. Therefore, the pWCET $C^*_i$ needs to be estimated, leading to a safe upper-bound $\overline{C}_i$. The reasoning is the same as with classical deterministic WCET computation, where WCET$\text{exact}$ is unknown, which leads to the computation of a possibly safe WCET$\text{static}$ and WCET$\text{predicted}$ bounds: we aim to compute safe and possibly accurate $\overline{C}_i$ estimations. With probabilities and measurements it is easier to compute distributions $\overline{C}_i$ from appropriate ETP measures, moreover the rare events theory can ensure the safety of such pWCET estimation $\overline{C}_i$. The forthcoming section explains how to get a safe pWCET estimation starting from the measurements.

2.2.1 From the Measures to the Worst-Cases: the Rare Events

An approach to the pWCET estimation refers to rare events to estimate exceedence probabilities which are smaller than $10^{-n}$, where $n$ is a required level of confidence. In this paper we refer to the extreme value theory to estimate the probability of occurrence of extreme values of execution times, known to be rare events, [13]. More precisely, EVT predicts the distribution function for the maximal (the worst-case) or minimal (the best-case) values of a set of $n$ observations, which are modelled as random variables, [11].

The EVT demands certain hypotheses to provide a “good” and “safe” pWCET estimation. The two main hypotheses are independence and identical distribution (i.i.d) of the random variables involved.

The empirical distributions through which we model the observations have to be independent (i.), meaning not sharing any relationship or not affecting one another. The presence of any dependence in a series of execution times influences the extremal behavior of the execution time. Nonetheless, such hypothesis is not that strict under certain conditions. For processes satisfying a weak mixing condition [29], the degree of long-term dependence at extreme levels is limited, hence the EVT as the variables were independent, is a safe bound.

The identical distribution (i.i.d.) states that the empirical distributions modeling consecutive observations have to repeat identically. This way the EVT would have enough information to accurately model the expected rare events from the measurements. Beware that i.i.d. is a far weaker constraint than the usual deterministic repeatability. Identical measurement-based distributions may be identical even if every single measure is not repeatable. This is another key advantage of pWCET over deterministic WCET.

3. Real-Time Dependences

The objective of our paper is to study the dependences that real-time tasks suffer within realistic systems. Cache memories, scheduling policies, interfering tasks etc. affect the task execution modifying its ETPs and consequently its worst-case, both the deterministic and the probabilistic ones. With our probabilistic framework we intend to quantify such effects studying how the task execution time distribution varies from the case where the interfering event is not present to the case where it is present.

3.1 Formalizing the Dependences

We can envision a real-time system $O$ as a set of events $O_j$, $O = \{O_1, O_2, \ldots, O_n\}$, where each $O_j$ happens within the system and participates to both its functional and non-functional behavior. Since we are interested in the execution time and what is affecting it, it is possible to narrow down the set of events to those which characterize the execution of a task.

The execution of tasks mainly depends on the code that implements it. Aside that, but not marginally, there are other system elements which influence the task execution and consequently its execution time. An event $O_j$ affects $C_i$ if its happening changes the execution time itself and its representing distribution $\overline{C}_i$. Given a task $\tau_i$, for those events affecting $C_i$ we could have a system representation as

$$O^j = \{O^j_1, O^j_2, \ldots\},$$

where $O^j$ is the system view with respect to $C_i$ and its effect over $C_i$. For the sake of simplicity, in the rest of the paper we refer to $O$ and $O_j$, respectively for the system representation from $\tau_i$, and the events affecting $C_i$.

Among the events affecting task $\tau_i$ execution there are also the other tasks $\tau_k$ concurrently executing with $\tau_i$. Then, for two events such as the task execution time $C_i$ and $O_j$ the conditional probability describes the effect that $O_j$ has on $C_i$,

$$\mathbb{P}(C_i | O_j) = \frac{\mathbb{P}(C_i \cap O_j)}{\mathbb{P}(O_j)} \tag{4}$$

as the probability of having the execution time $C_i$ once $O_j$ happens. In this paper we will not consider all the possible events affecting the task execution time; instead, we begin the analysis with some of them. With $C_i | O_j \overset{\text{def}}{=} C_i \cap O_j$, and $\mathbb{P}(C_i | O_j) \overset{\text{def}}{=} \mathbb{P}(C_i \cap O_j)$, the conditional probability joints events putting together the task execution time and the effect of $O_j$ on it. In general $C_i, O_j \neq C_i$, due to the dependence relationship between $C_i$ and $O_j$.

Lemma 1 (Independence of the CDF) Assuming two dependent execution time random variables $C_i$ and $C_k$, the random variable $C_i | C_k$ is independent from $C_k$ and $F_{C_k} = F_{C_i | C_k} | C_k$.

Proof: With $C_i$ and $C_k$ dependent, it is $F_{C_i, C_k} = F_{C_i | C_k} F_{C_k} = F_{C_i} F_{C_k}$: Considering the execution time distribution $C_i | C_k$ with $F_{C_i | C_k}$ its CDF representation, it is $F_{C_i, C_k} = F_{C_i | C_k} F_{C_k} = F_{C_i} F_{C_k}$.

$F_{C_i, C_k} = F_{C_i} C_k$ and $F_{C_i | C_k} = F_{C_i | C_k} C_k$, stating the independence between $C_i | C_k$ and $C_k$. Furthermore,

$$F_{C_i, C_k} = F_{C_i} C_k,$$

so that $F_{C_k} = F_{C_k} | C_k$; then the lemma follows.

With the improved description $C_i, O_j$ of the task execution it is also possible to conclude about the independence of the execution time conditional distribution with respect to any system.
event $O_j$. For example, with $C_i \not\supset C_k$, thanks to Lemma 1, $C_i \cup C_k$ is independent from $C_k$, being $P(C_i \cup C_k) = P(C_i)P(C_k)$. In fact, considering an event $O_j$, once its effects are included inside the conditional distribution of execution times of $\tau_i, C_i, O_j$, they will no longer affect it, i.e. the main task will run independently from that event. Obviously, Lemma 1 can be generalized to consider any kind of distribution, not only execution time distributions.

It is worth noting that a very wide variety of events can fit in the set $O$: this mostly depends on the abstraction level we may want to consider, $O_j = C_i$. For instance, the presence of another task $\tau_j$ running together with the main one could be represented at a finer level of detail as the distribution of execution times it generates. The main consequence of this reasoning is that any event could have an associated probability distribution function, even trivial such as a binary random variable. What matters to ensure the applicability of Lemma 1 is the existence of that distribution, regardless of its shape.

### 3.2 Dependence Composability

With Equation (3) and Equation (4), it is possible to decompose the dependence problem into sub-problems following the idea of set partitioning. Thus, the execution time of a task $\tau_i, C_i, O_i$, is composed by all the effects from the events $O_j$ which are related to the execution time.

In such a probabilistic scenario we can define the isolation as a system configuration where the task $\tau_i$ is not influenced by the rest of the system. While executing in isolation, the task experiences ETPs $C_i$ with no effects from other tasks or system elements. Then, it is possible to define the notion of not-isolation, as a system configuration where the task receives interference from the elements composing the system configuration, $O = \{O_1, O_2, \ldots \}$. In the next section we will see how to instantiate the isolation and not-isolation for realistic real-time systems.

### 3.3 Toward the Characterization of the Dependences

The study of dependences within a probabilistic real-time framework are mostly associated with copulas, [2]. In this paper we re-formulate the basics of copulas for the characterization of random variables dependences and enhance the analysis to realistic and complex real-time systems.

In a first approximation, we can think about a measure of the dependence as a distance between the two random variables. Then, a difference $\Delta_i$ between two execution time distributions $C_i$ and $C'_i$, in terms of 1-CDFs, could be represented as $\Delta_i = \max_x |F_{C_i}(x) - F'_{C_i}(x)|$. A meaningful difference among 1-CDFs would be the one that considers the shift of a distribution,

$$\Delta_i, O_j = \min\{\Delta |\forall c \in \mathcal{F}_{C_i}(c - \Delta) \geq \mathcal{F}_{C_i, O_j}(c)\}.$$  

(6)

$\Delta_i, O_j$ characterizes the effect that $O_j$ has on $\tau_i$ and its execution time; hence it is a measure of the dependence between $O_j$ and $C_i$. Then $\Delta_i, O_j$ can be seen as the cost required in order to create independence between $C_i$ and $O_j$. With Equation (6), it is possible to upper-bound the dependence between $C_i$ and any of the events $O_j$ composing $O$ with a distribution $C'_i, O_j$, such that

$$F^{\text{sup}}_{C'_i, O_j}(c) = \mathcal{F}_{C'_i}(c - \Delta_i, O_j).$$  

(7)

We have now a distribution ables to bound the independent case $C_i | O_j$.

#### Theorem 1 (Independence Bound)

Given a task $\tau_i$ with execution time distribution $C_i$ in isolation. In case $\tau_i$ executes together with another event $O_j$ the resulting execution time distribution is $C_i, O_j$. The distribution $C'_i, O_j$, such that $F^{\text{sup}}_{C'_i, O_j}(c) = F_{C'_i}(c - \Delta_i, O_j)$ with $\Delta_i, O_j = \min\{\Delta |\forall c \in \mathcal{F}_{C'_i}(c - \Delta) \geq \mathcal{F}_{C'_i, O_j}(c)\}$ represents an upper-bound to the independence case between $C_i$ and $O_j$, $C_i | O_j$.

**Proof:** We distinguish three cases. In the case $\Delta_i, O_j = 0$, which states the independence between $C_i$ and $O_j$, $\forall c \in \mathcal{F}_{C_i}(c) = F_{C_i}(c)$, and no influence comes from the event $O_j$ to $C_i$. Then $\forall c \in \mathcal{F}^{\text{sup}}_{C'_i, O_j}(c) = \mathcal{F}^{\text{up}}_{C'_i, O_j}(c) = \mathcal{F}_{C'_i}(c)$, upper-bounding such independence case.

With $\Delta_i, O_j > 0$, it is $\forall c \in \mathcal{F}_{C'_i}(c) > \mathcal{F}_{C'_i}(c)$ and

$$F_{C'_i}(c) \leq F^{\text{up}}_{C'_i, O_j}(c) \leq F^{\text{sup}}_{C'_i, O_j}(c);$$

(8)

then, $F^{\text{sup}}_{C'_i, O_j}$ upper-bounds the independence case $C_i | O_j, C_i, O_j$.

Finally, with $\Delta_i, O_j < 0$, it is $\mathcal{F}^{\text{sup}}_{C'_i, O_j}(c) < \mathcal{F}^{\text{up}}_{C'_i, O_j}(c) \forall c$, and

$$F^{\text{up}}_{C'_i, O_j}(c) \leq F^{\text{sup}}_{C'_i, O_j}(c) \leq F_{C'_i}(c)$$

(9)

by the definition of bounding, Equation (6). Although $F_{C'_i}$ is a safe bound, it could be very pessimistic; then $C'_i, O_j$ as $F^{\text{sup}}_{C'_i, O_j}$ represents a more accurate upper-bound to the independence $C_i | O_j$.

With such bound $F^{\text{sup}}_{C'_i, O_j}$, it is possible to safely carry on the probabilistic real-time analyses with the independence hypothesis, see [24] as an example that requires task independence assumption.

### 4. Experimental setup

In this section we expose the characteristics (hardware and software) of the experiments we set up in order to begin investigating the dependences within real-time systems.

#### 4.1 Tracing Instrumentation

The main goal of our experimental setup is the appropriate measurement of execution time profiles for various real-time task sets. Since we want to address any kind of execution target, we cannot rely on simulation: neither a cycle accurate one, nor any hardware-specific assisted observation, because those may not be available. Therefore, we have to go for instrumented real-time execution, often called tracing.

The performance monitoring tool we have chosen, LTTng (Linux Trace Toolkit new generation) [16], adheres to the paradigm of static tracing, which is usually performed by adding logging statements to the code and compiling them with the program. This kind of instrumentation can be suitable to support real-time critical activities, since it achieves low overhead at the only payback of a small increase in the code size. It allows static collection of timestamped traces, for both kernel and user-space events, together with a new powerful feature, namely the possibility of appending to trace events performance monitoring counters [32]. This possibility allows to monitor with great precision the behavior of the cache, which is among the events affecting tasks executions that we intend to investigate.

In our measurements we collect as less data as possible, in order to minimize perturbation, and we use hardware performance counters when available, e.g. for counting cache misses.
4.2 Architecture

All the experiments have been run on a machine having two Intel® Xeon® E5620 2.4 GHz sockets, each one with four cores and three levels of cache. The first two levels (L1 and L2) are “per core”, while the last level (L3) is “per socket”, hence shared among each group of four cores. The L1 cache is partitioned into 32 KB data cache and 32 KB instruction cache, while L2 and L3 (respectively 256 KB and 12 MB) are unified.

With the kernel boot parameter isolcpus we specify the set of cores that we may want to isolate from the general SMP balancing and scheduling algorithms. Exploiting this option, the first four cores have been reserved to our experiments, while the general scheduling was bounded to the remaining ones. Likewise, all the interrupts have been redirected on the second CPU, except those preventing the correct behavior of the system, for example timing, performance monitoring and non-maskable ones. This setup constitutes the embodiment of the definition of isolation we gave in section 3.2.

4.3 Execution Environment: SCHEDMCore

SCHEDMCore is an open source integrated development framework for critical embedded systems [10], which provides two main tools: the SCHEDMCore CONVERTER, a validation tool which, after transforming a task model description into a formally analyzable model in C or UPPAAL, can perform schedulability analysis, and the SCHEDMCore RUNNER, an execution tool which envisions precise real-time execution on a multi-core architecture, using various real-time scheduling policies. The tools take as input a file describing the task model (either as a textual description or a PRELUDE library [28]). The SCHEDMCore CONVERTER was used to ensure that the used task set was theoretically schedulable. The SCHEDMCore runtime, used by the SCHEDMCore RUNNER, is implemented as a user-space library, which means that the Linux kernel does not need to be patched and that it can be maintained without following the evolution of the underlying OS. This user-space approach has been used in the past [23] in order to avoid the burden of going inside the OS kernel; even more recently the same approach has been proven efficient as well.

Besides the system isolation setup we described previously, SCHEDMCore tools and runtime ensure usual real-time execution: memory lock, switching to real-time scheduling policy, bounding each thread (runtime or user) to its own private core, and so on.

In order to ease and automatize the experiments, SCHEDMCore framework may load during startup a scheduler plugin (FP, EDF, LLREF, etc.), which implements the scheduling policy at user level, as well as any kind of C function [25] to be used for the implementation of the real-time task. The configuration is specified in the task file description [25] for one part and the rest of it from the command line [27], i.e. the taskset description is defined in the taskfile, whereas real-time parameters like sched policy, binding, time base etc. are specified from the command line.

5 Experiments

Three are the benchmarks that we study, all of them taken from the Mälardalen Benchmark suite [19]. The first one, “jfdint”, is a simple single-path benchmark with a small data footprint, but a consistent code size. The second one, “cnt”, is more data consistent, but still single-path benchmark. Finally, we tackle with the complexity of a multi-path case by exploiting the “nsichneu” benchmark.

These three benchmarks are the beginning of a more complex analysis we intend to develop. We consider them significative since they explore various aspects such as the data and instruction cache as well as the multi-path problem.

Within a realistic real-time system, by isolation (ISO) we intend the execution of the task under analysis with the fewest possible effects from the system, i.e. reduced number of active interrupts, smallest set of system calls, single core affinity and no other functional tasks running on the same core. The tested scheduling policies range from Non-Preemptive (NP) to Fixed Priority (FP) and Earliest Deadline First (EDF). The experiments we carried on consider the isolation case as baseline for comparing execution time and cache misses values.

We further investigate the cache effects on task execution time with two distinct cache interferences in case of preemption: those coming from preemptive tasks performing simple CPU burning (“burn”), and those from preemptive tasks performing more consistent cache activities (“cache”).

In this stage of the work, we deal with an observed task \( \tau_i \) and an interfering task \( \tau_j \), either preempting or not. According to the different configurations tested, both \( \tau_i \) and \( \tau_j \) are asked to change in order to better exploit the system events under investigation. The measured quantities for each job of \( \tau_i \) are the a) response time in \( \mu \text{sec} \), b) execution time in \( \mu \text{sec} \), c) L1-Instruction cache load misses, d) L1-Data cache load misses, e) L3 cache load misses and the f) number of context switches.

5.1 EVT Applicability

We make use of the Gumbel distribution to describe the tail behavior of execution times. On the measurements we have carried out few tests to verify the EVT applicability. The first is the exponential tail test [18], to check that the execution time observations effectively fit a Gumbel distribution [11]. All our empirical distributions ETP pass that test.

Then, we have tested the i.d. hypothesis with the two-sample Kolmogorov-Smirnov test [31], in which two samples of 5000 execution time observations are compared. The test verifies that two samples out of the ETPs are identically distributed.

Finally, the independence hypothesis has been tested with the runs test [4], looking for randomness between consecutive observations. For most of the observations such test fails due to the small variability of the execution time we have. The only exception is the multi-path benchmark, where multiple paths are exercised at each execution. This increases the randomness degree of consecutive observations. Although not independent, our measurements follow a linear dependence or stationarity, that we verified with linear correlation indexes. The linear dependence guarantees the EVT to be a good tail projection for the ETPs, [29].

Table 5.1 has the results from the Kolmogorov-Smirnov test, the run test and the correlation (Pearson’s correlation coefficient) test to samples of 10,000 execution time observations, applied to some of the observations considered. Represented is the probability of obtaining a test statistic at least as extreme as the one that was actually observed assuming the null hypothe-

---

5 see <kernelsrc>/Documentation/kernel-parameters.txt
6 https://forge.onera.fr/projects/schedmcore
sis true, in our case the hypotheses of being i.d., i., and linear dependence. The p-value less than the predetermined significance level, often 0.05, indicates that the observed result would be highly unlikely under the null hypothesis. Thus, we can accept both the i.d. and the linear dependence hypotheses but not the i. hypothesis.

5.2 Simple Benchmark: jfdctint

The jfdctint benchmark performs a JPEG integer implementation of a forward discrete-cosine transform on a reduced data structure of 8x8 pixel block.

We begin considering the effects produced by the interference of a preempping task \( \tau_i \), in both “burn” and “cache” configurations. Figure 3 shows that the latter exhibits a larger execution time profile, especially in the tail-end region. In the same figure, we also represent the upper-bound to the task ETPs, which takes into account the preemption costs relative to both “burn” and “cache” configurations. Such upper-bound \( F_{\tau_i}^{up} \) (upper-bound in the figure), defined in Equation (7), where \( O_j \) is the preemption event. The preemption effect to the jfdctint \( \tau_i \) can be bounded by \( \Delta_i,O_j = 80 \mu \text{s} \) out of around 1.2sec as maximum execution time, Equation (6). With an ETP as the one defined by \( F_{\tau_i}^{up} \), we can assume \( \tau_i \) to be independent from the preemption effect.

By exploring the scheduling policy and its effects on the task execution time we notice how the higher scheduling overhead from EDF with respect to the FP scheduler, determines a substantial increase in the execution time profile, as can be seen in Figure 4. An upper-bound to the scheduler effects is quantified with \( F_{\tau_i}^{up} \) and \( \Delta_i,O_j = 108 \mu \text{s} \).

We have also taken into account the number of preemptions \( \tau_i \) experiences. In particular, the overhead related to the context switch and the indirect effects on cache may determine a further increase in the execution time. For a simple benchmark as the one we are considering, only a small variation is perceived while increasing the number of preemptions, Figure 5. This is due to the almost negligible context switch overhead with respect to the task computation time jfdctint has. We can notice that multiple preemptions do not change considerably the task largest observed values; nonetheless, their main effect remains the increase of the average task execution time, between 6000\( \mu \text{s} \) and 6040\( \mu \text{s} \).

5.3 Heavy Data-structured Benchmark: cnt

The cnt benchmark counts the number of non-negative values inside a square matrix, which is randomly initialized. We apply it to analyze in detail the contribution of the cache effects on the task execution time. In particular, leaving constant the data structure of the observed task \( \tau_i \), we play with the data size of the interfering task \( \tau_j \) implemented with the cnt function, in order to draw conclusions on the effect that cache misses have on the execution time.

The set of ETPs of Figure 6 captures the great variability that we achieve when the data structure of the preempping task is progressively augmented. We observe a substantial increase in the execution time for average values, because the heavier the data structure of \( \tau_j \) is the more cache trashing is created during the preemptions. Nonetheless, this trend does not translate into a proportional growth of the largest execution time values. Indeed the isolation configuration has larger execution time than the “burn” case as well as the “cache” case with 1000x1000 data structure. This is due to other phenomena which characterize modern processor architectures such as prefetch and branch prediction. Although not yet able to properly model those effects, our measurement-based approach can observe them and the ef-

<table>
<thead>
<tr>
<th>jfdctint ISO</th>
<th>cnt ISO</th>
<th>nsichneu ISO</th>
<th>nsichneu FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>i.d.</td>
<td>0.95</td>
<td>0.93</td>
<td>0.97</td>
</tr>
<tr>
<td>linear</td>
<td>&lt; 2.2e−16</td>
<td>&lt; 2.2e−16</td>
<td>&lt; 2.2e−16</td>
</tr>
<tr>
<td>upper-bound</td>
<td>0.3</td>
<td>−0.8</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 1. p-values for the i.i.d. and linear correlation tests. jfdctint in isolation, cnt in isolation, nsichneu in isolation and nsichneu with preemptions are reported.

Figure 3. jfdctint: isolation, non-preemptive and preemptive case comparison.

Figure 4. jfdctint: scheduling policy comparison.

Figure 5. jfdctint: number of preemptions comparison. The execution times are much larger than the previous jfdctint tests due to few modifications required in order to exploit multiple preemptions from other tasks.
fects they provoke. In the future, we will make an effort to complete our probabilistic modeling framework and include them in. An \( F^\text{up}_{C_{i},O_{j}} \) built as in Equation (7) (the upper-bound in the figure) with \( \Delta_{i,O_{j}} = 124 \mu \text{sec} \) upper-bounds all these effects.

3. “inner” configuration, Figure 8, where the input is randomly-chosen in order to explore many different sub-paths. The resulting distribution has only one peak and large variance.

Comparing the three configurations in the “cache” preemptive case, we notice that the first two exhibit a similar behavior, with slightly larger values in the “wc” configuration, Figure 8. The third one, instead, highlights the typical problems of coverage related to multi-path cases: since a huge number of combinations of sub-paths is allowed, with very low probability the largest values will be observed, preventing a real confidence on the measurements. For the worst-case configuration, we build the upper-

\[ F^\text{up}_{C_{i},O_{j}} \] (upper-bound on Figure 9), which takes in account the interference of a “cache” task \( \tau_{i} \), and leads to a value of \( \Delta_{i,O_{j}} = 100 \mu \text{sec} \). The application of EVT on this bound is shown to be very accurate with a difference of around 50\( \mu \text{sec} \) out of the 7.1\( \mu \text{sec} \) of maximum measured execution time, Figure 9. This is due to the good knowledge of the task execution

5.4 Multi-path Benchmark: nsichneu

Multi-path benchmarks imply an increased level of complexity for the probabilistic analysis, because the input choice may determine substantial effects on the execution time and the involved system elements. Nevertheless, the proposed approach can be easily applied also to multi-path cases.

The multi-path benchmark nsichneu we aim to analyze, simulates an extended Petri net, exploiting automatically generated code. Its small data structure will result in small variations of the execution time whenever the intereririencing load increases. Each of its jobs is built exploring three fundamental input patterns:

1. “outer” configuration, Figure 8, where the input is chosen randomly in a set of three, including the one generating the worst-case. The resulting distribution shows three peaks corresponding to the three explored paths;

2. “wc” configuration, Figure 8, where the input is the one which always exercises the worst path among the three, resulting in the largest execution times. This represents the worst-case configuration;

3. “inner” configuration, Figure 8, where the input is randomly-chosen in order to explore many different sub-paths. The resulting distribution has only one peak and large variance.

In terms of extreme value theory this pWCET estimation \( C_{i}' \), Figure 7 shows that we can achieve great accuracy with our observed values due to the accuracy of our ETP measurements. Then, with the \( C_{i} \) and the EVT applied, we can safely perform probabilistic scheduling analysis [24], assuming independence between tasks for the configurations studied with \( \text{cnt} \), as stated by Theorem 1.

![Figure 6. cnt: execution time comparison.](image6)

![Figure 7. cnt: extreme value theory of the execution time.](image7)

![Figure 8. nsichneu: ET comparison for the three configurations.](image8)

![Figure 9. nsichneu configuration 3: upper-bound and extreme value theory.](image9)
quantifying dependence effects due to concurrent tasks and interferences, scheduling policies, as well as hardware elements, i.e. cache memories for few benchmarks.

We leave as future work the investigation of other system effects, such as the presence of multiple cores or others typical for modern processor architectures. We will also focus on the improvement of the dependence metrics, in order to reduce the pessimism of the pWCET estimation. Our framework will be applied to a wider set of benchmarks with different features, in order to enhance the knowledge about today’s complex and unpredictable real-time systems.

References