Formal Analysis of Sporadic Overload in Real-Time Systems

Sophie Quinton, Matthias Hanke, Rolf Ernst
Institute of Computer and Network Engineering,
TU Braunschweig, 38106 Braunschweig, Germany
Email: {quinton, hanke, ernst}@ida.ing.tu-bs.de

Abstract—**This paper presents a new compositional approach providing safe quantitative information about real-time systems. Our method is based on a new model to describe sporadic overload at the input of a system. We show how to derive from such a model safe quantitative information about the response time of each task. Experiments demonstrate the efficiency of this approach on a real-life example. In addition we improve the state of the art in compositional performance analysis by introducing execution time models which take into account several consecutive executions and by using tighter bounds for computing output event models.**

I. Introduction and related work

Formal performance verification of systems with real-time constraints is both essential and complex. Real-time systems very often consist of multiple components such as processors or buses on which multiple tasks are executing concurrently. The interferences between tasks are complex and very hard to capture by simulation, thus making formal analysis essential in order to obtain safe information on the behavior of the system. Today, two main compositional approaches have been successful in tackling this problem, namely Real-Time Calculus [14], [3] and Compositional Performance Analysis (CPA) [11], [4].

These methods are proven efficient and safe, however their results are often rather pessimistic when compared to simulations of the systems that have been analyzed. This discrepancy between analysis and simulation results has two explanations. First, the result of the analysis is an overestimation of the actual worst-case scenario possible in the system. Second, the actual worst-case scenario may be very rare and thus not appear in simulations, although it is indeed possible. In this case, one may restrict the analysis by excluding the unlikely worst-case scenario in the analysis. It is then crucial to provide a tight bound on the "error" made by the new analysis.

In this paper we follow the CPA approach. We show how to obtain tighter bounds on worst-case analysis results and we propose a method for refining worst-case results when they are due to a sporadic overload at the input of the system.

On the one hand, we improve the state of the art in compositional performance analysis by introducing a new execution time model for capturing dependencies between consecutive instances of tasks. This model is based on a generalization of the intra event contexts used in Context-Aware Analysis [6], which already improved on the multiframe model of [10]. Our model is particularly useful when some incomplete information is available about sequences of execution times. We then integrate this new execution time model into the Multiple Event Busy Time analysis introduced in [12] for which we also propose improved bounds on the task termination models.

On the other hand, we propose an approach which performs two CPAs, one with a complete model of the possible activations at the input of the system providing the worst case and another one with a restricted model covering only the "typical" activations observed at the input. The worst-case exceptions which are not covered by the restricted model are formally represented using what we call an *overload model.* Based on the result of the worst-case analysis and the overload model at the input, we are then able to provide a safe bound on the number of response times for a given task which will be out of the range obtained using the typical activation model, represented as an error model. Experiments show that this approach may result in dramatically shorter typical response times associated with safe and reasonably tight bounds on the error model when some tasks are activated both periodically (time-triggered) and aperiodically (event-triggered).

This paper is organized as follows. Section II presents our representation of systems. Section III then revisits and improves on the multiple event busy time approach to compositional performance analysis. Section IV presents our approach for a compositional error analysis. Finally Section V presents experimental results obtained with an industrial example while Section VI concludes.

Related work. Stochastic analysis [8], [9], [15] typically aims at providing a distribution of the response times of each task in a system. Although this research area has been extensively studied, existing stochastic approaches still suffer from several limitations. First, they are computationally highly expensive. Furthermore most of them rely on the assumption that the activation and execution times of tasks can be represented as independent random variables. This is in general not the case, as e.g. cache memory induces a correlation between the various execution times of a given task. Besides, a data flow between two tasks may also lead to a correlation between their

This work was funded by the ITEA2 project TIMMO-2-USE (EUREKA cluster N° 3674) through the German Ministry of Education and Research (BMBF) under the funding ID 01IS10034.

respective execution times. Ignoring these dependencies is not safe as shown in [5] while attempts at dealing with them lead to overly pessimistic results [2], [5].

Finally, any stochastic approach requires the complete (possibly infinite) set of execution traces (each trace being also possibly infinite) in order to determine whether a system satisfies a stochastic model. In other words, a stochastic analysis does not yield any safe information about the behavior of a system in a given time window. In this respect, our approach is rather related to the analysis of weakly-hard real-time systems [1] in so far as we provide a safe upper bound on the number of response times outside the specified model. Note that unlike [1] we are not restricted to periodic tasks.

II. A TRACE-BASED REPRESENTATION OF SYSTEMS

The systems that we want to analyze consist of software components performing for example a computation or some storage or communication, which we call *tasks*. These components are mapped onto hardware components (e.g. processors, memories, buses) called *resources*, on which they execute. When a resource is shared by several tasks, a scheduling policy decides in which order tasks are executed.

The execution of a task is triggered by an input event received by the task, called *activation*. The end of the execution is indicated by the output of another event, called *termination*. As is the case in all event-based approaches, we abstract from the data often associated with an activation and identify the behavior of the system with how events occur during execution. This makes the size of our system abstraction manageable. An *event trace* describes the set of instants at which an event takes place. Note that we only use traces focusing on a specific type of event, which is either the activation or the termination of a given task τ. Such a trace is called the *activation trace* (resp. *termination trace*) of τ.

Definition 1. An *event trace* is a function $\sigma : \mathbb{N}^+ \longrightarrow \mathbb{N}$ where $\sigma(n)$ denotes the time of the n-th occurrence of an event in the trace.

Definition 2. A *system* is a tuple $Sys = (T, R, \rightarrow, \Omega)$ where:

– T is a (finite) set of *tasks*; we keep the notion of task abstract here. Tasks are arbitrarily named $\tau_1, \ldots, \tau_n$.

– R is a (finite) set of resources defined as a partition of T, i.e., we identify a resource with the tasks executed on it.

– $\rightarrow \subseteq T \times T$ is a relation describing how tasks interact; for two tasks $\tau_1, \tau_2 \in T$, $\tau_1 \rightarrow \tau_2$ means that the output of task τ_1 is connected to the input of task τ_2.

– Ω is the set of *behaviors* of the system, that is, tuples $(act_1, end_1, \ldots, act_n, end_n)$ of event traces where for each task τ_i, act_i is the activation trace of τ_i and end_i is its termination trace.

We call *source* a task τ such that $\{\tau' \in T \mid \tau' \rightarrow \tau\}$ is empty. A source is activated directly by the environment of the system rather than by another task. A *task instance* is a pair relating in a behavior one activation and its corresponding termination, as follows.

Definition 3. Consider a system Sys as above and a behavior $(act_1, end_1, \ldots, act_n, end_n) \in \Omega$. For a given $n \in \mathbb{N}^+$, the *n-th instance* of a task τ_i is the pair $(act_i(n), end_i(n))$. The *response time* of the n-th instance of τ_i in the given behavior is the time interval $RT_i(n) = end_i(n) - act_i(n)$.

The *worst-case response time* (WCRT) of τ_i in Sys, is the maximal $RT_i(n)$ over all $n \in \mathbb{N}^+$ and all behaviors in Ω.

In practice, the set of behaviors of a system is too complex to be known directly. The only reliable information available for analysis is usually related to: 1) the input of the system, that is, the activation traces of the sources; 2) the way the activation of a task is triggered by the termination of the tasks connected to it, usually some AND or OR operation; 3) the scheduling policy enforced by each resource, 4) the execution times of tasks, often represented by one interval $[BCET; WCET]$ per task between its best-case and its worst-case execution time. Now, if there exist constraints on the system, e.g. with respect to the worst-case response times of tasks, then at least an overapproximation of the set of possible behaviors is needed. This is what Compositional Performance Analysis (CPA) [4] achieves based on a fixpoint computation using the above information. This approach groups traces into *event models* to make the analysis scalable.

Definition 4. An *event model* Σ is a set of event traces. An event trace σ *satisfies* an event model Σ if $\sigma \in \Sigma$. A behavior $(act_1, end_1, \ldots, act_n, end_n)$ satisfies an activation model Σ_i^{act} if $act_i \in \Sigma_i^{act}$; similarly for termination models.

CPA supposes that an activation model (i.e. a set of activation traces) is given for each source. The following steps are then performed iteratively until a fixpoint is reached.

– For each resource $r \in R$ a *local performance analysis* function takes as input one activation model per task in r and returns one termination model per task such that any behavior satisfying the activation models given as input also satisfies the computed termination models. This local analysis of r takes into account the scheduling policy of r and all possible execution times of tasks mapped onto r.

– New activation models of tasks are computed from the termination models connected to them (except for sources).

The termination of this fixpoint computation is proven in [13]. We now focus on the problematic part of the analysis, namely local performance analysis.

III. COMPOSITIONAL PERFORMANCE ANALYSIS REVISITED

In this section, we improve on the Multiple Event Busy Time [12] approach for local performance analysis by refining the model $[BCET; WCET]$ used to describe execution times. We consider instead the best-case and worst-case execution time of a task over *several* instances, which is useful when execution times vary from one instance to the next.

Definition 5. The *execution time model* of a task τ_i consists of two functions (ET_i^-, ET_i^+) such that $\forall q \in \mathbb{N}^+ : ET_i^-(q)$ (resp. $ET_i^+(q)$) is the best-case (resp. worst-case) cumulative execution time of q consecutive instances.

Our model is inspired by the multiframe model [10] which handles cases where the pattern of execution times is fully known and was extended in [6] to situations where only partial information is available. The latter is however less general than the model presented here: there, a predefined number of possible execution times $\{ET_1, \dots, ET_k\}$ is defined along with properties of the form "Out of m instances, at most (resp. at least) n have an execution time ET_k". From this a best-case and a worst-case sequence are built whose length is the smallest common multiple of all m appearing in some property. Within the sequence the smallest (resp. largest) execution times are always supposed to occur first. This does not allow the exploitation of more fine-grain information about the order in which execution times occur.

After the execution model, let us now focus on event models. CPA was originally introduced for periodic with jitter event models and was later generalized to *load event models*, which describe sets of traces according to the minimum and maximum size of the time interval between two events in a trace. Depending on the context it is more convenient to adopt an event-based (δ-functions) or a time-based (η-functions) view so we present both. The former is easily obtained from the latter and vice versa.

Definition 6. For a trace σ, we define $\delta_\sigma : \mathbb{N}^+ \times \mathbb{N}^+ \longrightarrow \mathbb{N}$:

$$\forall k, n \geq 1 : \delta_\sigma(n, k) = \sigma(n + k - 1) - \sigma(n)$$

Then for all $k \geq 1$, $\delta^+_\sigma(k)$ denotes the maximum $\delta_\sigma(n, k)$ and $\delta^-_\sigma(k)$ the minimum $\delta_\sigma(n, k)$ over all $n \geq 1$.

Definition 7. Two functions δ^- and $\delta^+ : \mathbb{N}^+ \longrightarrow \mathbb{N}$ define an event model (δ^-, δ^+) as the set of event traces σ such that: $\delta^- \leq \delta^-_\sigma$, i.e. $\forall k \geq 1 : \delta^-(k) \leq \delta^-_\sigma(k)$, and $\delta^+_\sigma \leq \delta^+$.

Note that to denote a non-empty set of traces an event model (δ^-, δ^+) must have some specific properties, e.g. $\delta^- \leq \delta^+$.

Definition 8. Given a trace σ, we define:

$$\forall \Delta t \geq 1, \forall t \geq 1 : \eta_\sigma(t, \Delta t) = \sum_{i=0}^{\Delta t} \textit{event-at}_\sigma(t + i)$$

where $\textit{event-at}_\sigma(t) = 1$ if $\sigma(n) = t$ for some $n \geq 1$ (there is an event occurrence at time t) and $\textit{event-at}_\sigma(t) = 0$ otherwise. Then for all $\Delta t \geq 1$, $\eta^+_\sigma(\Delta t)$ denotes the maximum $\eta_\sigma(t, \Delta t)$ and $\eta^-_\sigma(\Delta t)$ the minimum $\eta_\sigma(t, \Delta t)$ over all $t \geq 1$.

Definition 9. Two functions η^- and $\eta^+ : \mathbb{N} \longrightarrow \mathbb{N}$ define an event model (η^-, η^+) as the set of event traces σ such that: $\eta^- \leq \eta^-_\sigma$ and $\eta^+_\sigma \leq \eta^+$.

Let us assume that we are given a system Sys as in Definition 2. As we look for safe results (by contrast with stochastic approaches), it is sufficient to consider an arbitrary behavior $(act_1, end_1, \dots, act_n, end_n)$ of Sys. Worst-case response time analysis is based on the notion of *busy window* [7]. We define a level-i busy window as a time interval $[act_i(n); end_i(n + q)]$ for $n \geq 1$ and $q \geq 0$ such that:

- $\forall k \in [1; q] : act_i(n + k) < end_i(n + k - 1)$
- $end_i(n + q) \leq act_i(n + q + 1)$
- $end_i(n - 1) \leq act_i(n)$

That is, a level-i busy window is a maximal time interval during which τ_i is *busy*, where busy means that an activation is currently being processed (as opposed to idle) [1].

The notion of *multiple event busy time* was introduced in [12] to improve the results obtained by CPA. For a task τ_i mapped onto a resource r and $k \geq 1$, it is denoted $B^+_i(k)$ and represents the maximum time it may take r to process k activations of τ_i within a level-i busy window starting with the first of these k activations. We focus on the Static Priority Preemptive (SPP) [16] scheduling policy, which defines a strict order between tasks by assigning them a priority such that higher-priority tasks are executed first and may preempt (interrupt the execution of) lower-priority tasks.

Definition 10. The *multiple event busy time* $B^+_i(k)$ for a task τ_i under SPP scheduling is the smallest value such that

$$B^+_i(k) = ET^+_i(k) + \sum_{j \in hp(i)} ET^+_j(\eta^+_{j,in}(B^+_i(k)))$$

where

- ET^+_i is the worst-case cumulative execution time of τ_i
- $hp(i)$ is the set of tasks with higher priority than τ_i
- $\eta^+_{j,in}(\Delta t)$ is the maximum number of activations of task τ_i in a time window of size Δt.

B^-_i is defined by replacing all $^+$ by $^-$ in the above definition.

Note that the multiple event busy time does not depend on the actual activation model of τ_i. In practice, $B^+_i(k)$ is relevant only if the activation of τ_i makes it possible for k consecutive activations to arrive before the previous instance is finished, as expressed in Theorem 1.

Theorem 1. The size of a level-i busy window is bounded [2] by $B^+_i(K_i)$ where $K_i = \min\{k \geq 1 \mid B^+_i(k) < \delta^-_i(k + 1)\}$.

Proof. The activation model of τ_i is denoted here (δ^-_i, δ^+_i). Consider a level-i busy window of size greater or equal to K_i starting at $act_i(n)$. By definition of B^+_i we know that $end_i(n + K_i - 1) \leq act_i(n) + B^+_i(K_i)$. Besides, by definition of δ^-_i we know that $act_i(n + K_i) \geq act_i(n) + \delta^-_i(K_i + 1)$. Thus, as by definition of K_i we have $B^+_i(K_i) < \delta^-_i(K_i + 1)$, it follows that $end_i(n + K_i - 1) < act_i(n + K_i)$ and a level-i busy window cannot be longer than $B^+_i(K_i)$. □

Remember that it is safe to use $B^+_i(k)$ only if $k \leq K_i$ while $B^-_i(k)$ is always safe. In the rest of the section, we define bounds on the WCRT of tasks as well as for the output of our local performance analysis. The results presented here generalize those of [12] and their proofs will be useful in the next section for defining how sporadic overload propagates through the system. As from now on we consider only one task τ_i we simplify notation by omitting the i indexes. The activation and termination models of τ_i are denoted respectively $(\delta^-_{in}, \delta^+_{in})$ and $(\delta^-_{out}, \delta^+_{out})$.

1. Note that our definition slightly differs from that of [7] as we do not require higher priority tasks to be idle at the beginning of the busy window. Both definitions yield the same size of the largest busy window.

2. This result does not appear in [12] although a (potentially infinite) set K is used for the same purpose in the other proofs of the paper.

Theorem 2. The response time of τ_i is bounded by

$$WCRT = \max_{1\leq k\leq K}\{B^+(k) - \delta^-_{in}(k)\}$$

Proof. Consider the n-th instance of τ_i. As K is the maximal size of a level-i busy window, there is at least one $k \in [1; K]$ such that $act(n-k+1)$ marks the beginning of a busy window. Denote k the latest of these events. By definition of the response time, $RT(n) = end(n) - act(n)$. Besides, from the definition of B^+, $end(n) \leq act(n-k+1) + B^+(k)$ and from that of δ^- we obtain $act(n) - act(n-k+1) \geq \delta^-_{in}(k)$. Hence $RT(n) \leq B^+(k) - \delta^-_{in}(k)$. The result follows. □

Theorem 3. For $q \geq 1$, define $\delta^+_{out}(q)$ as the maximum[3] of

$$\max_{0\leq k\leq (K-q)}\{B^+(k+q) - B^-(k+1)\}$$
and
$$\max_{1\leq k'\leq \min(q-1,K)}\{\delta^+_{in}(q-k'+1) + B^+(k')\} - B^-(1)$$

Then for all $n \in \mathbb{N}^+$, $\delta_{out}(n,q) \leq \delta^+_{out}(q)$.

Proof. By definition $\delta_{out}(n,q) = end(n+q-1) - end(n)$.

Case 1. Suppose that $act(n)$ and $act(n+q-1)$ are in the same busy window as represented in Figure 1 where $k \in [0; K-q]$ is such that $act(n-k)$ is the first activation in this busy window (represented by a grey box in the figure).

Then $end(n+q-1) \leq act(n-k) + B^+(k+q)$ and $end(n) \geq act(n-k) + B^-(k+1)$ by definition of B^+ and B^- respectively. Hence

$$\delta_{out}(n,q) \leq \max_{0\leq k\leq (K-q)}\{B^+(k+q) - B^-(k+1)\}$$

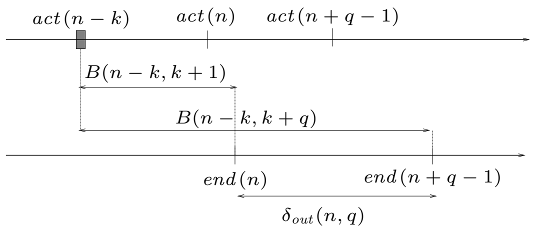

Figure 1. $\delta_{out}(n,q)$ when only one busy window is considered.

Case 2. Suppose now that $act(n)$ and $act(n+q-1)$ are in two level-i busy windows. Let $1 \leq k' \leq \min(q-1,K)$ be such that $act(n+q-k')$ is the first activation in the busy window of $act(n+q-1)$ as represented in Figure 2. Note that it is sufficient to consider that k' is smaller than $q-1$ (otherwise $act(n)$ and $act(n+q-1)$ are in the same level-i busy window) and k' is also smaller than K (otherwise $act(n+q-k')$ and $act(n+q-1)$ cannot be in the same busy window).

Then $end(n+q-1) \leq act(n+q-k') + B^+_i(k')$ by definition of B^+_i. Besides the definition of δ^+_i implies that $act(n+q-k') \leq act(n) + \delta^+_{in}(q-k'+1)$. Hence $end(n+q-1) \leq act(n) + \delta^+_{in}(q-k'+1) + B^+(k')$. We also know that $end(n) \geq act(n) + B^-(1)$. The result follows. □

3. The case where $q \leq K$ was not covered in [12].

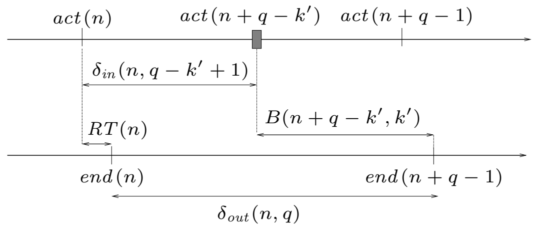

Figure 2. $\delta_{out}(n,q)$ when several busy windows are considered.

Theorem 4. For $q, n \geq 1$, define $\delta^-_{out}(q)$ as the maximum[4] of

$$\min_{0\leq k\leq K-1}\{\max_{1\leq k'\leq q}\{\delta^-_{in}(k+q-k') + B^-(k')\} - B^+(k)\}$$

and $B^-(q-1)$. Then for all $n \geq 1$, $\delta_{out}(n,q) \geq \delta^-_{out}(q)$.

Proof. This proof follows the same pattern as the previous one so we rely mostly on Figure 3 for the proof. Unlike B^+, it is always safe to use B^-, so $\delta^-_{in}(k+q-k') + B^-(k')$ is a safe approximation of $end(n+q-1) - act(n-k)$ for any k', and the presence of a max in the definition of $\delta^-_{out}(q)$. The second bound, namely $B^-(q-1)$, is a rather straightforward approximation for the case where $act(n)$ and $act(n+q-1)$ are in the same busy window.

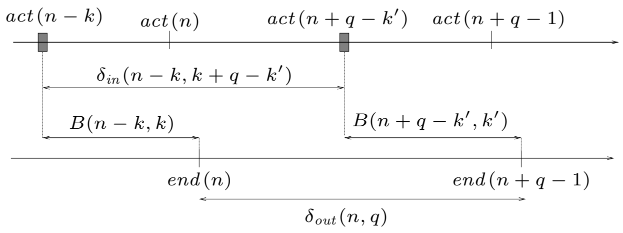

Figure 3. $\delta_{out}(n,q)$ when several busy windows are considered. □

We have now completed the part devoted to improving the worst-case response time analysis in the context of CPA. The next section combines these results with a new approach for refining worst-case results in presence of sporadic overload at the input of the system.

IV. Analysis of sporadic overload

Let us introduce our approach on an example. Consider two tasks τ_1 and τ_2 executing on the same resource and such that τ_1 has higher priority than τ_2. Suppose furthermore that τ_2 is a periodic task of period $\mathcal{P}$ while τ_1 is "almost" periodic: it is activated with period $\mathcal{P}$ with a few additional activations — but never more than one out of three periods. The execution time of τ_1 is always $\mathcal{P}/3$ and that of τ_2 always $\mathcal{P}/2$. A simple analysis finds out that the largest size of a level-2 busy window is $2 \times \mathcal{P}$ and the worst-case response time of τ_2 is $3 \times \mathcal{P}/2$, as illustrated on Figure 4 where grey rectangles denote execution and white rectangles blocking to the execution of a higher priority task.

As we mentioned, the overload at the input of τ_1 is sporadic, so that even if the worst-case scenario $WCRT_2$ is indeed possible, it is rare and most of the observed response times for τ_2 will in fact measure $5 \times \mathcal{P}/6$.

4. This bound generalizes that of [12] which only considers $k' = 1$.

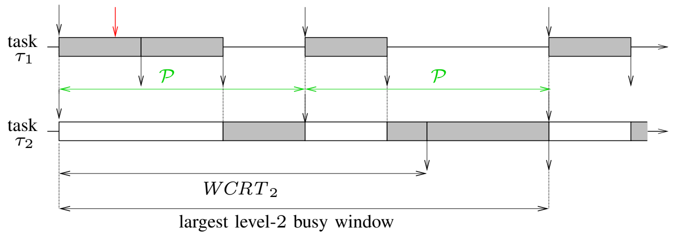

Figure 4. Largest level-2 busy window and worst-case response time of τ_2.

Rather than using a probabilistic approach for formalizing this, we choose a representation inspired by load event models.

Definition 11. An *overload model* is an event model, which we denote either δ^-_{over} or η^+_{over} depending on whether we adopt an event-based or a time-based view, and such that for $k \geq 1$, $\delta^-_{over}(k)$ is the minimum distance between k overload events while for $\Delta t \geq 1$, $\eta^+_{over}(\Delta t)$ is the maximum number of overload events in a time interval of size Δt. As we focus on WCRTs we are not interested in δ^+_{over} or η^-_{over}.

Given two event models (δ^-, δ^+) and $(\boldsymbol{\delta^-}, \boldsymbol{\delta^+})$ where the former is a safe (worst-case) model and the latter is an approximate (typical-case) model, an overload model δ^-_{over} relating them is such that any trace satisfying (δ^-, δ^+) can be decomposed into a trace satisfying $(\boldsymbol{\delta^-}, \boldsymbol{\delta^+})$ and another one satisfying δ^-_{over}.

Let us now consider a system for which a safe model is defined as in the previous sections. We focus on a source task τ_i for which a safe activation model (δ^-_i, δ^+_i) is known. Note that in this section we are only concerned with activation models so we omit the *in* indexes. A first CPA (as in Section III) on this safe (worst-case) model provides us with the maximum number K_i of instances of τ_i that can possibly be in the same level-i busy window, the size $B^+_i(K_i)$ of the longest possible level-i busy window and the worst-case response-time $WCRT_i$ of τ_i.

Suppose now that we are given an approximate model $(\boldsymbol{\delta^-_i}, \boldsymbol{\delta^+_i})$ of the activation of τ_i and an overload model $\delta^-_{i,over}$ relating it with the safe activation model (δ^-_i, δ^+_i) of τ_i. We perform a second CPA of our system where (δ^-_i, δ^+_i) is replaced by $(\boldsymbol{\delta^-_i}, \boldsymbol{\delta^+_i})$. From this we obtain approximate results $\boldsymbol{K_i}$, $\boldsymbol{B^+_i(K_i)}$ and $\boldsymbol{WCRT_i}$. In the example of Figure 4, the approximate results for τ_1 obtained by ignoring its sporadic overload activations would be as follows: $\boldsymbol{K_1} = 1$ and $\boldsymbol{B^+_1(K_1)} = \boldsymbol{WCRT_1} = \mathcal{P}/3$.

At this point, the question is: how often can $\boldsymbol{WCRT_i}$ be "wrong", meaning that the actual response time of τ_i is larger than $\boldsymbol{WCRT_i}$ (while still being smaller than $WCRT_i$)? The answer to this question is represented as an *error model* which expresses safe bounds on the number of response times which may be larger than $\boldsymbol{WCRT_i}$.

Definition 12. An *error model* for $\boldsymbol{WCRT_i}$ is a function $err_i : \mathbb{N}^+ \longrightarrow \mathbb{N}$ such that $err_i(k)$ is a safe bound on the number of instances of τ_i which can have a response time larger than $\boldsymbol{WCRT_i}$ in a window of k consecutive instances.

Let us now focus on how to obtain such an error model. Consider first the following sufficient condition for proving that the response time $RT_i(n)$ of instance n is smaller than $\boldsymbol{WCRT_i}$, based on the fact that activations can interfere with each other only if they are in the same busy window.

Theorem 5. $\forall n \geq 1 : RT_i(n) \leq \boldsymbol{WCRT_i}$ if

$$\forall k, q \in [1; K_i] : \delta_i(n-k, q) \geq \boldsymbol{\delta_i}^-(q)$$

Proof. Suppose that $\forall k, q \in [1; K_i] : \delta_i(n-k, q) \geq \boldsymbol{\delta_i}^-(q)$. According to the proof of Theorem 2, we have the following: $\forall n \geq 1 : RT_i(n) \leq \boldsymbol{WCRT_i}$ if

– $\forall k \in [1; \boldsymbol{K_i}] : \delta_i(n-k, k) \geq \boldsymbol{\delta_i}^-(k)$
– $\forall k \in [1; \boldsymbol{K_i}] : B_i(n-k+1, k) \leq \boldsymbol{B^+_i}(k)$
– $\exists k \in [1; \boldsymbol{K_i}] : end_i(n-k) \leq act_i(n-k+1)$

The first condition is trivially satisfied. As the level-i multiple event busy time does not depend on the activation model of τ_i, the second condition always holds if overload occurs only at the input of τ_i. So let us show that the third condition is satisfied, namely that the level-i busy window containing $act_i(n)$ starts with $act_i(n-k)$ for some $k \in [1; \boldsymbol{K_i}]$.

There is at least one activation corresponding to the beginning of a busy window between $act_i(n - K_i + 1)$ and $act_i(n)$, which we denote $act_i(n - \boldsymbol{k})$. We know that for all $k \leq \boldsymbol{k}$ and $q \in [1; K_i]$, $\delta_i(n-k, q) \geq \boldsymbol{\delta_i}^-(q)$. This implies (using the same argument as in the proof of Theorem 1) that the busy window starting with $act_i(n - \boldsymbol{k})$ and those that follow until $act_i(n)$ cannot be larger than $\boldsymbol{B^+_i(K_i)}$. As a consequence, the busy window containing $act_i(n)$ cannot start earlier than $act_i(n-k)$ for some $k \in [1; \boldsymbol{K_i}]$. □

The direct corollary of this theorem is that one overload event at the input of τ_i can result in at most K_i response times larger than $\boldsymbol{WCRT_i}$ because $RT_i(n)$ depends only on its previous K_i activations. Hence the following result.

Theorem 6. If overload occurs only at the input of τ_i then

$$err_i(k) = K_i \times \eta^+_{i,over}(\delta^+_i(k + K_i))$$

Proof. We consider how many overload activations may occur during the time needed for $k + K_i$ activations. Note that we take into account the effect of the K_i activations before the first one in the considered sequence of k instances. □

Suppose now that overload may happen also in activation models of tasks with higher priority executing on the same resource, as for τ_2 in our running example.

Theorem 7. If overload concerns only tasks in $hp(i)$, then

$$err_i(k) = K_i \times \sum_{j \in hp(i)} \eta^+_{j,over}(\delta^+_i(k + K_i) + WCRT_i)$$

Proof. The same reasoning as above applies here: one overload event at the input of τ_j, where $j \in hp(i)$, interferes with one level-i busy window and thus impacts at most K_i response times of τ_i. Note that one must consider overload activations of tasks in $hp(i)$ which occur after the last of the k activations but before the end of the corresponding execution. □

We then compute err_i by conservatively adding both errors.

V. Experiments

We have applied our approach to a real-life example trace of running gear consisting of 12 tasks running on a single resource. There is no task-chaining in this system, so all tasks are activated by external events or strictly periodically. We present results related to two tasks of interest, τ_7 and τ_5.

An analysis of the trace leads to the following observation: τ_7 is almost periodic of period 2.5 ms except for additional activations, which denotes an activation both time-triggered and event-triggered. The event-based activations are rare, hence the idea to perform an approximate analysis ignoring these events. We have derived from the trace the following overload model for τ_7, which shows e.g. that the minimum interval containing 5 overload activations of τ_7 is 407 ms.

$\delta^-_{7,over}$ of	2	3	4	5	6	7	8
equals:	9	29	157	407	887	1392	1642

Let us note that the execution times of τ_7 illustrate the interest of our execution time model: most execution times follow a pattern of length 10 where a long execution alternates with a short one. On top of that, the additional activations of τ_7 result in shorter executions which introduces uncertainty. Our model is the only one able to handle efficiently such a pattern of execution times.

We now focus on the impact of the overload in the activation of τ_7 on τ_5, a task with a very low priority. τ_5 is fully periodic (period 10 ms). The following table shows the obtained WCRTs for both the original model M and its approximated model $\boldsymbol{M}$, computed without, then with, cumulative execution times (denoted with the index c).

model	M	M_c	$\boldsymbol{M}$	$\boldsymbol{M}_c$
$WCRT_5$	non-schedulable	8.616	4.473	4.288

This shows first that using cumulative execution times significantly improves the results but also that a relatively small approximation in the model (about 0.4% of the events in the activation trace of τ_7 are ignored) has a dramatic impact on the WCRT. Note that the worst-case response time observed in the trace is 4.142 ms, while most response times are between 3.9 and 4.0 ms. Our approximate result is rather accurate.

The worst-case analysis of τ_5 indicates that a level-5 busy window never contains more than 1 activation since the WCRT of τ_5 is smaller than its period. As a result, one overload event in τ_7 cannot impact more than one response time in τ_5. Hence the following error model[5] for $\boldsymbol{WCRT_5}$. For example, we conclude that out of 163 consecutive instances of τ_5, at most 7 have a response time larger than 4.287 ms — roughly 4%. Note that this result is a *safe* bound.

err_5 of	1	2	14	39	87	138	163
equals	1	2	3	4	5	6	7

5. For readability we only show the step values. In fact, one can read for example that for any sequence of $15 \leq k \leq 39$ activations, $\boldsymbol{WCRT_5}$ is incorrect at most 4 times.

VI. Conclusion

We have presented new results for formal performance analysis of real-time systems to obtain safe response time bounds for systems with sporadic higher loads. Our solution provides two WCRTs, the baseline "typical" WRCT and a larger maximum WCRT when higher load is applied. Given a bound for the sporadic overload, analysis gives an exact bound of the maximum WCRTs occurrences in a given number of instances. The result is significantly more expressive than using maximum WCRTs times only and can, e.g., be used by control engineers to better bound the system behavior. Furthermore this approach does not assume any of the strict contraints of typical probabilistic approaches, such as randomness, independence or stationarity, and can therefore be used more flexibly and requires less knowledge about the system to be analyzed. Future work includes the study of overload propagation in order to provide a fully compositional approach.

References

[1] G. Bernat, A. Burns, and A. Llamosí, "Weakly hard real-time systems," *IEEE Trans. Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[2] G. Bernat, A. Burns, and M. Newby, "Probabilistic timing analysis: An approach using copulas," *J. Embedded Computing*, vol. 1, no. 2, pp. 179–194, 2005.

[3] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proceedings of DATE'03*. IEEE Computer Society, 2003, pp. 190–195.

[4] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," in *IEE Proceedings Computers and Digital Techniques*, 2005.

[5] M. Ivers and R. Ernst, "Probabilistic network loads with dependencies and the effect on queue sojourn times," in *Proceedings of QSHINE'09*, ser. LNCS, vol. 22. Springer, 2009, pp. 280–296.

[6] M. Jersak, R. Henia, and R. Ernst, "Context-aware performance analysis for efficient embedded system design," in *Proceedings of DATE'04*. IEEE Computer Society, 2004, pp. 1046–1051.

[7] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of RTSS'90*. IEEE Computer Society, 1990, pp. 201–213.

[8] J. M. López, J. L. Díaz, J. Entrialgo, and D. F. García, "Stochastic analysis of real-time systems under preemptive priority-driven scheduling," *Real-Time Systems*, vol. 40, no. 2, pp. 180–207, 2008.

[9] S. Manolache, "Analysis and optimisation of real-time systems with stochastic behaviour," Ph.D. dissertation, Linköpings Universitet, 2005.

[10] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," *IEEE Trans. Software Eng.*, vol. 23, no. 10, pp. 635–645, 1997.

[11] K. Richter, "Compositional scheduling analysis using standards event models," Ph.D. dissertation, TU Braunschweig, 2005.

[12] S. Schliecker, J. Rox, M. Ivers, and R. Ernst, "Providing accurate event models for the analysis of heterogeneous multiprocessor systems," in *Proceedings of CODES+ISSS'08*. ACM, 2008, pp. 185–190.

[13] S. Stein, J. Diemer, M. Ivers, S. Schliecker, and R. Ernst, "On the convergence of the SymTA/S analysis," TU Braunschweig, Tech. Rep., 2008.

[14] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of ISCAS'00*, vol. 4. IEEE Computer Society, 2000, pp. 101–104.

[15] T.-S. Tia, Z. Deng, M. Shankar, M. F. Storch, J. S. 0002, L.-C. Wu, and J. W.-S. Liu, "Probabilistic performance guarantee for real-time tasks with varying computation times," in *Proceedings of RTAS'95*. IEEE Computer Society, 1995, pp. 164–173.

[16] K. Tindell, A. Burns, and A. J. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, pp. 133–151, 1994.