Formal Analysis of Sporadic Bursts in Real-Time Systems

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

***Abstract*—In this paper we propose a new method for the analysis of response times in uni-processor real-time systems where task activation patterns may contain sporadic bursts. We use a burst model to calculate how often response times may exceed the worst-case response time bound obtained while ignoring bursts. This work is of particular interest to deal with dual-cyclic frames in the analysis of CAN buses. Our approach can handle arbitrary activation patterns and the static priority preemptive as well as non-preemptive scheduling policies. Experiments show the applicability and the benefits of the proposed method.**

I. Introduction

Formal performance verification of systems with real-time constraints is becoming essential as their complexity increases. The interferences between concurrently executing tasks are very hard to capture by simulation, thus making formal analysis essential in order to obtain reliable information on the system behavior. Methods focusing on providing upper bounds on the worst-case timing scenario for a given system are proven efficient and safe [2], [1], but their results do not reflect the frequency of worst-case occurrences. In many practical cases, however, embedded system applications accept occasional deadline violations, if the maximum number of occurrences can be bounded. Recently, a new formalism has been introduced to bound such typical worst-case behavior [3] based on the concept of *weakly-hard constraints* [4].

In this paper we build upon the method proposed in [3] for providing quantitative information in addition to worst-case results when these are due to a sporadic overload at the input of the system. The general idea consists in performing for each task a response time analysis of the system using a "typical worst-case" model in which the sporadic overload is ignored. Based on the result of the usual worst-case analysis and a formal representation of the ignored overload activations at the input, a safe bound on the number of response times for a given task which may be out of the range obtained using the "typical" activation model is then computed. This *error model* is a function err such that for any sequence of k consecutive executions of the task under consideration ($k \in \mathbb{N}^+$) at most $err(k)$ response times may be larger than the bound obtained while ignoring the overload. This approach can be used for example to analyze deadline misses in systems where tasks execute completely even if they have missed their deadline.

978-3-9815370-0-0/DATE13/©2013 EDAA

The approach of [3] ignores dependencies between overload activations. In presence of bursts, this will lead to overly pessimistic results, as each activation in the burst is assumed to impact a complete busy window. Furthermore, the overload model that was used in [3] assumes that activation traces can be decomposed into typical-case and overload activation traces; this is usually impossible for bursts. We solve these issues by introducing a more general model for representing bursts of activations, for which we provide a formula to compute safe error models.

Our work is motivated by the need to handle dual-cyclic frames, also called cyclicIfActiveFast [5], in the analysis of CAN buses [6]. Such frames (tasks) execute according to two different periods, depending on the context. Traditional worst-case analysis only considers the smallest (most pessimistic) period even if it is used seldom during execution. The extension of [3] to bursts and to static-priority non-preemptive (SPNP) [6] scheduling presented here covers such cases.

This paper is organized as follows. Section II presents related work. Section III describes our system model and Section IV recalls the basics of the busy window analysis for SPP and SPNP scheduling. Section V introduces the principle of our approach, while Section VI describes the burst model and Section VII shows how to compute the error model. Finally Section VIII presents experimental results while Section IX concludes.

II. Related work

Our approach uses the response time analysis of [2]. A comparable method based on a similar model of systems is Real-Time Calculus [1]. In particular, [7] also provides a model for bursts. However, so far no other work than [3] using either [2] or [1] to compute weakly-hard response times of tasks has been published.

Both (m, k)-firm [8] and weakly-hard real-time systems [4] provide guarantees similar those described in this paper, allowing at most m out of k consecutive deadlines to be missed. However, (m, k)-firm constraints are enforced by resorting to dedicated scheduling mechanisms. Our approach is therefore more in the line of weakly-hard constraints which are only used for analysis purposes. One major difference between [4] and the work that we present here is that we are not limited to periodic systems in contrast to [4] which is based on an analysis of the system hyperperiod.

Stochastic analysis [9] typically aims at providing a distribution of the response times of each task in a system, thus also refining worst-case information. The main limitation of this family of approaches is that they must rely on the assumption that 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, and a data flow between two tasks may lead to a correlation between their respective execution times. Ignoring these dependencies is not safe while attempts at dealing with them lead to overly pessimistic results [10]. In the end, stochastic approaches yield in general little information about the behavior of a system in a given time window.

III. A TRACE-BASED REPRESENTATION OF SYSTEMS

In this paper, we assume a system of *tasks* (software components) executing on a single *resource* (e.g. a processor, memory, or bus) scheduled according to the static-priority preemptive (SPP) [11] or static-priority non-preemptive (SPNP) [6] policy, which 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*. An *event trace* describes the set of instants (described as natural numbers) 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.

Definition 1. *An* event trace σ *is an increasing (possibly infinite) sequence of instants where* $\sigma(n) \in \mathbb{N}$ *denotes the time of the* n*-th occurrence of an event in the trace.*

Definition 2. *The* behavior *of a system is represented by a tuple* $(act_1, end_1, \ldots, act_n, end_n)$ *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.*

In practice, the only reliable information available about the possible behaviors of a system is related to: 1) the scheduling policy; 2) the execution times of tasks, i.e., the time they use the resource to produce their output; 3) the activation traces at the input of the system. In this paper, we describe execution times using the usual interval $[BCET; WCET]$ between a task's best-case (i.e., smallest) and worst-case (largest) execution time[1].

Besides, traces are abstractly represented as *event models* which describe the minimum and maximum size of a time interval containing a given number of events in a trace by means of δ-functions, or equivalently the minimum and maximum number of events which may occur in a given time interval (η-functions).

Definition 3. *An* event model *is represented either as a pair of functions* δ^- *and* $\delta^+ : \mathbb{N}^+ \longrightarrow \mathbb{N}$*, or* η^- *and* $\eta^+ : \mathbb{N} \longrightarrow \mathbb{N}$*.*

An event trace σ satisfies *an event model* (δ^-, δ^+) *if*

$$\forall k, n \geq 1 : \sigma(n+k-1) - \sigma(n) \in [\delta^-(k); \delta^+(k)]$$

For finite traces this condition must be satisfied for all k *and* n *such that all terms are defined.*
Equivalently a trace satisfies (η^-, η^+) *if*

$$\forall \Delta t, t \geq 1 : \sum_{i=0}^{\Delta t - 1} event\text{-}at_\sigma(t+i) \in [\eta^-(\Delta t), \eta^+(\Delta t)]$$

where $event\text{-}at_\sigma(t)$ *is the number of event occurrences at* t*.*

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

We now formally define response times, for which we want to obtain safe information. A *job* is a pair relating in a behavior one activation and its corresponding termination, as follows.

Definition 4. *Consider a system Sys as above and a behavior* $(act_1, end_1, \ldots, act_n, end_n)$ *of Sys. For a given* $n \in \mathbb{N}^+$*, the* n-th job *of a task* τ_i *is the pair* $(act_i(n), end_i(n))$*. The* response time *of the* n*-th job 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 of Sys.*

IV. CLASSIC WORST-CASE RESPONSE-TIME ANALYSIS

In this section we recall the basics of worst-case response time analysis for SPP and SPNP scheduling on a single resource, which are based on the notion of *busy window* [12] (originally called busy period). We assume given a system Sys and an arbitrary behavior $(act_1, end_1, \ldots, act_n, end_n)$ of Sys. The SPP and SPNP scheduling policies define a strict order between tasks by assigning them a priority such that higher-priority tasks execute first and may also, in the preemptive case, interrupt the execution of lower-priority tasks.

Definition 5. *A level-*i *busy window is a maximal time interval during which* τ_i *or a higher-priority task has a job that has not terminated yet.*

The maximum level-i busy window for a task τ_i is built by assuming the occurrence of a so-called *critical instant*, where τ_i and higher-priority tasks are all activated at the same time — and therefore induce maximum interference with τ_i — while the task with the largest execution time among lower-priority tasks is activated just before the critical instant and therefore induces the maximum blocking time[2]. It is also assumed that all tasks are activated as early as possible and that they always use their maximum execution time. The maximum level-i busy window stops at the first instant when no job of τ_i and higher-priority tasks remains incomplete.

The worst-case response-time $WCRT_i$ of τ_i can then be safely defined as the maximum response time observed in this maximum busy window. The formal definitions and proofs of the response time analysis can be found for the periodic

[1]Note that we could as well use the more general model of [3] which considers the best-case and worst-case execution time of a task integrated over *several* executions.

[2]Note that this last condition is only needed for SPNP as lower-priority tasks can be safely ignored in the SPP case.

case in [11] (SPP) and [6] (SPNP), and for general activation patterns e.g. in [3] (SPP) and [13] (SPNP). In addition, these methods return, for the SPNP case, the *maximum queuing delay* QD_i which describes the maximum time it may take before task τ_i starts executing (remember that in the non-preemptive case this means that is cannot be blocked anymore). They also compute the maximum number of *buffered* activations bf_i, i.e. activations whose corresponding job has not *started* yet, as well as the maximum *backlog* bl_i, i.e. activations whose corresponding job has not *finished* yet.

The whole process is illustrated on an example in Figure 1, where waiting times (due to the execution of other tasks) are represented as empty rectangles while execution times are rectangles indicating their duration. The system consists of three tasks τ_1, τ_2 and τ_3 executing on the same resource scheduled according to SPNP, such that τ_1 has higher priority than τ_2, denoted $\tau_1 \succ \tau_2$, and $\tau_2 \succ \tau_3$. Tasks τ_2 and τ_3 are cyclic, i.e. periodic ($\mathcal{P} = 15$ ms), while τ_1 is dual-cyclic: it is activated sometimes according to a short period $\mathcal{P}_1 = 2,5$ ms, sometimes according to a longer period $\mathcal{P}'_1 = 5$ ms. We focus on task τ_2. Note that the analysis is performed assuming the worst-case activation pattern of τ_1, namely $\mathcal{P}_1$, even if in practice such an activation scenario might seldom occur.

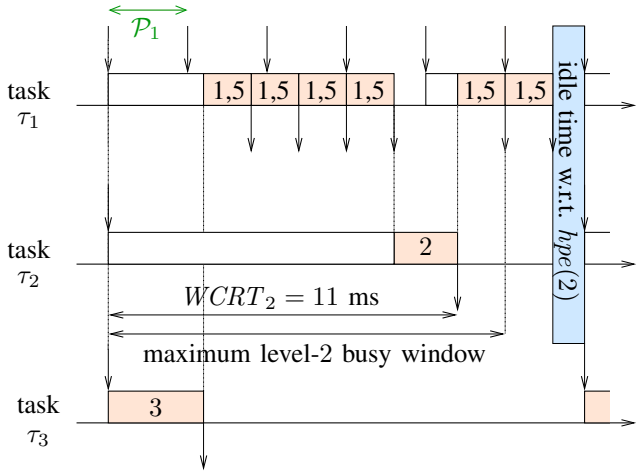

Fig. 1. Busy window analysis and worst-case response time of τ_2.

V. Principle of typical-case analysis of bursts

Let us introduce our approach on the example of Section IV. As already mentioned, task τ_1 may typically not be activated according to its smaller period $\mathcal{P}_1 = 2,5$ ms, but rather most of the time according to its larger period $\mathcal{P}'_1 = 5$ ms. Hence the idea to perform a busy window analysis of τ_2 assuming the "typical-case" activation model of τ_1, namely period $\mathcal{P}'_1$, as illustrated in Figure 2. Such an analysis yields a much smaller worst-case response time for τ_2, which we denote $\boldsymbol{TWCRT_2}$.

Of course, $\boldsymbol{TWCRT_2}$ is *not* a safe bound for the response time of τ_2. At this point, the question then is: how often can $\boldsymbol{TWCRT_2}$ be "wrong", meaning that the actual response time of τ_2 is larger than $\boldsymbol{TWCRT_2}$ (while still being smaller than $WCRT_2$)? Typical-case analysis will provide an answer

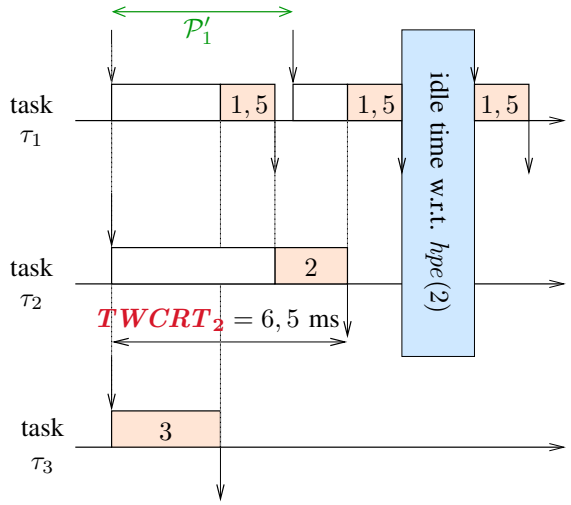

Fig. 2. Busy window analysis of τ_2 using the typical-case model.

to this question in the form of an *error model* as defined below and the next two sections are devoted to the description of how we come up with such an error model.

Definition 6. *An* error model *for a given* $\boldsymbol{TWCRT_i}$ *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* $\boldsymbol{TWCRT_i}$ *in a window of* k *consecutive instances.*

That is, out of every sequence of k (consecutive) jobs of τ_2 (for a given $k \in \mathbb{N}^+$), *at most* $err_2(k)$ response times may be larger than $\boldsymbol{TWCRT_2}$. In other words, err_2 must provide safe (i.e. worst-case) information about the behavior of the system over a time window.

VI. Burst model

We start by defining our representation of burst. Informally, a burst is a time interval during which a task is activated more often than average. The role of the burst model is to relate the worst-case and the typical-case activation models of a task and therefore the definition of an adequate burst model for a given task depends on its worst-case and typical-case activation models. For a dual-cyclic frame, the burst model would describe how often and how long the task may be activated according to its shorter period, as illustrated in Figure 3, where activations drawn in bold belong to a burst. Note that bursts start with the first activation which arrives according to the shorter period and end before the last activation arriving with the shorter period.

Definition 7. *A* burst model *is a pair* $(\Delta^+_{burst}, \delta^-_{burst})$ *where* Δ^+_{burst} *denotes the maximum time duration of a burst while* δ^-_{burst} *is an event model describing how often bursts may occur. Formally, for* $k \geq 1$, $\delta^-_{burst}(k)$ *is the minimum distance between the beginning of a burst and the beginning of the next* $(k-1)$*-th burst.*

In particular, $\delta^-_{burst}(2)$ is the minimum distance between the beginning of a burst and the beginning of the next burst.

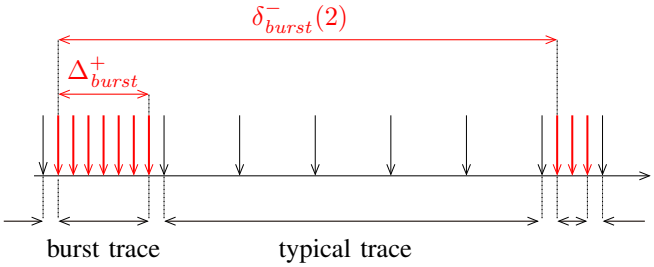

Fig. 3. Burst model for the dual-cyclic activation pattern.

As usual, we switch whenever convenient to a time-based view of δ^-_{burst}, i.e., a function η^+_{burst} such that for $\Delta t \geq 0$, $\eta^+_{burst}(\Delta t)$ is the maximum number of bursts which may start in a time interval of size Δt. Note also that we are not interested in δ^+_{burst} or η^-_{burst} because we focus on WCRTs.

Definition 8. *Given two event models (δ^-_s, δ^+_s) and (δ^-_t, δ^+_t) where the former is a safe (worst-case) model and the latter is a typical-case model, a burst model δ^-_{burst} relating them is such that any trace satisfying (δ^-_s, δ^+_s) can be partitioned into a sequence of finite traces, alternately typical and burst (see Figure 3), such that:*

- *each typical finite trace satisfies (δ^-_t, δ^+_t);*
- *no burst finite trace is longer than Δ^+_{burst};*
- *the distance between the beginning of successive bursts satisfies δ^-_{burst}.*

Equipped with such a definition we can now focus on how to compute error models.

VII. Computation of the Error Model

The error model computation uses two main observations:

- Even when a burst occurs, response times satisfy the worst-case pattern, because the latter has been computed while taking these bursts into account. In particular, this implies that the impact of a burst after it ends cannot last longer than the worst-case busy window.
- If there is no burst activation in a busy window, response times in that busy window are smaller than or equal to the typical-case $\boldsymbol{TWCRT_i}$. Indeed, in such a case the worst possible scenario is the one computed by the busy window analysis on the typical-case model as in Figure 2.

Let us focus on the error model err_i of a given task τ_i. Remember that for $k \in \mathbb{N}^+$, $err_i(k)$ means the following: for any k consecutive jobs of τ_i, called a *k-sequence*, at most $err_i(k)$ response times may be larger than $\boldsymbol{TWCRT_i}$. At first we consider the error $err^j_i(k)$ for that k-sequence induced by bursts at the input of a single task τ_j with a priority higher than or equal to that of τ_i, as represented in Figure 4. We will show later how to add up errors resulting from bursts at the input of several tasks.

The computation of $err^j_i(k)$ goes as follows:

1) We first compute the impact of a burst at the input of τ_j, that is, how many response times of τ_i may be larger than $\boldsymbol{TWCRT_i}$ because of this burst.

2) We then compute the time interval ΔT^j_i during which a burst at the input of τ_j may have an impact on the response times in the considered k-sequence of τ_i.

3) Finally we compute how many bursts may occur at the input of τ_j during this time interval ΔT^j_i and take into account the impact of each burst on the response times of the k-sequence.

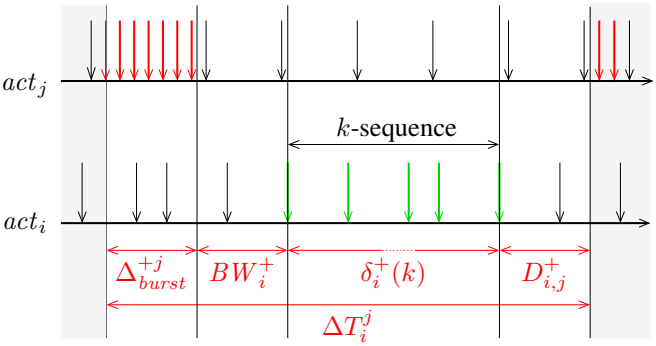

Fig. 4. Impact of a burst of τ_j on the response time of τ_i.

Step 1: Computation of the impact of a burst

Theorem 1. *The number of jobs of τ_i which may see their response time impacted by a given burst at the input of task τ_j is bounded by*

$$N^j_i = b_i + \eta^+_i(\Delta^{+j}_{burst} + BW^+_i)$$

where

$$b_i = \begin{cases} 0 & \text{if } i = j \\ bf_i & \text{if } i \neq j \text{ and the scheduling is SPNP} \\ bl_i & \text{if } i \neq j \text{ and the scheduling is SPP} \end{cases}$$

and bf_i and bl_i denote respectively the maximum number of buffered activations and the maximum backlog of τ_i.

Proof. A burst at the input of τ_j may only influence the response time of jobs of τ_i which are in one of the following situations: 1) still pending when the burst at the input of τ_j starts; 2) activated during the burst; 3) activated after the end of the burst but within the same busy window as the last activation of the burst.

- The impact of a burst after it ends lasts at most until the end of the busy window in which the last activation of the burst occurred (see Figure 4). Therefore, aside from the activations which are still pending at the beginning of the burst at the input of τ_j (see next item) the activations which may be impacted are those arriving during the burst (of length at most Δ^{+j}_{burst}) or within the following busy window (of length at most BW^+_i): that is, at most $\eta^+_i(\Delta^{+j}_{burst} + BW^+_i)$.
- Let b_i be the maximum number of activations of τ_i occurring before the beginning of a burst (at the input of τ_j), whose response time may be influenced by that burst. If $i = j$, the burst cannot influence the response time of previous activations of τ_i so $b_i = 0$. Similarly, if $i \neq j$ and the scheduling policy is SPNP then the burst can impact only the activations which are buffered when it starts, while

all backlogged (i.e. buffered or executing) activations may be impacted in the SPP case. □

An obvious consequence of this theorem is that a burst at the input of τ_j can result in at most N_i response times of the k-sequence of τ_i being larger than $\boldsymbol{TWCRT_i}$.

Step 2: Computation of the interval of impact ΔT_i^j

Consider the activation traces of Figure 4. The bursts at the input of τ_j cannot influence the response times of the represented k-sequence of activations of τ_i, as we explain now.

Theorem 2. *The time interval during which a burst at the input of τ_j may have an impact on the response times in the k-sequence of τ_i is bounded by*

$$\Delta T_i^j = \underbrace{\Delta_{burst}^{+j} + BW_i^+}_{(b)} + \underbrace{\delta_i^+(k)}_{(a)} + \underbrace{D_{i,j}^+}_{(c)}$$

where

$$D_{i,j}^+ = \begin{cases} 0 & \text{if } i = j \\ QD_i^+ & \text{if } i \neq j \text{ and the scheduling is SPNP} \\ WCRT_i & \text{if } i \neq j \text{ and the scheduling is SPP} \end{cases}$$

Proof.

(a) As already mentioned a burst starting during the k-sequence of activations of τ_i (see Figure 4) may have an impact on the response times in the k-sequence. The corresponding time interval is of length at most $\delta_i^+(k)$.
On top of that, we have to consider what happens if a burst starts just before the first activation in the k-sequence or after the last activation in the k-sequence.

(b) What happens before the beginning of the busy window containing the first activation in the k-sequence has no impact on the response times of the k-sequence. Therefore a burst starting more than $\Delta_{burst}^{+j} + BW_i^+$ before this first activation (see left part of Figure 4) will finish more than BW_i^+ earlier than the first activation of the k-sequence and therefore not impact it.

(c) If $i = j$ then a burst of activations of τ_j starting after the last activation in the k-sequence has no impact on the response times in the k-sequence at all, because activations of a task are handled in a FIFO order.
If $i \neq j$, what happens after the last activation in the k-sequence starts executing if the scheduling policy is SPNP (respectively finishes executing if it is SPP) has no impact on the response times of the k-sequence (see right part of Figure 4). The maximum interval of impact after the k-sequence is then the maximum queuing delay QD_i^+ of τ_i (respectively maximum response time $WCRT_i$).
The result follows directly from these considerations. □

Step 3: Computation of the error model $err_i^j(k)$

In order to compute a safe $err_i^j(k)$ we now simply compute the maximum number of bursts that may start during ΔT_i^j and multiply it by the maximum impact of each burst.

Definition 9. $err_i^j(k) = N_i^j \times \eta_{burst}^{+j}(\Delta T_i^j)$

Theorem 3. *For any sequence of k jobs of τ_i, if bursts occur only at the input of τ_j then at most $err_i^j(k)$ response times in the k-sequence of τ_i may be larger than $\boldsymbol{TWCRT_i}$.*

Proof. This follows directly from Theorems 1 and 2. □

We now define the error model $err_i(k)$ in the general case, i.e., when bursts may occur at the input of multiple tasks, as the sum of the error models induced by single tasks.

Definition 10. $err_i(k) = \sum_{j \in hpe(i)} err_i^j(k)$

Theorem 4. *For any sequence of k jobs of τ_i, at most $err_i(k)$ response times may be larger than $\boldsymbol{TWCRT_i}$.*

Proof. The scenario where bursts at the input of different tasks occur in different busy windows represents the worst case because overlapping bursts will in fact impact the same response times. Therefore adding the task-specific error models provides a safe over-approximation of the global error model. □

Improvement for handling short bursts

We can refine Theorem 1 in order to account better for bursts of small size, by replacing b_i in the formula with

$$\max_{0 \leq q \leq b_i} \{\, q \mid q \times BCET_i \leq \Delta_{burst}^{+j} \,\}$$

Theorem 5. *The number of jobs of τ_i which may see their response time impacted by a given burst at the input of task τ_j is bounded by*

$$N_i^j = \max_{0 \leq q \leq b_i} \{\, q \mid q \times BCET_i \leq \Delta_{burst}^{+j} \,\} + \eta_i^+(\Delta_{burst}^{+j} + BW_i^+)$$

where

$$b_i = \begin{cases} 0 & \text{if } i = j \\ bf_i & \text{if } i \neq j \text{ and the scheduling is SPNP} \\ bl_i & \text{if } i \neq j \text{ and the scheduling is SPP} \end{cases}$$

and bf_i and bl_i denote respectively the maximum number of buffered activations and the maximum backlog of τ_i.

Proof.

- The main part of the proof is the same as that of Theorem 1.
- Furthermore, if b_i activations cannot all finish executing before the end of the burst because $b_i \times BCET_i \leq \Delta_{burst}^{+j}$, then they necessarily belong to the same busy window as the last activation of the burst. In that case, the burst would be included in a level-i busy window which cannot be larger than BW_i^+. To avoid an overestimation because of this, we only need to consider the maximum number of pending activations at the beginning of the burst which may be in a different busy window as the last activation of the burst. □

Note that this improved version of the error computation is a generalization of the approach presented in [3]. In particular, an overload model, which describes the minimum distance between overload (i.e., additional) activations, can be used as a burst model by assigning the value 0 to Δ_{burst}^+.

task ID	activation pattern	priority
τ_1	cyclic	1 (highest)
τ_2	dual-cyclic	2
τ_3	dual-cyclic	3
τ_4	cyclic	4
τ_5	sporadic	5
τ_6	cyclic	6 (smallest)

TABLE I
STRUCTURE OF THE EXPERIMENTAL SYSTEM

VIII. EXPERIMENTS

We have applied our approach for typical-case analysis of bursts to a uni-processor system with 6 tasks scheduled according to SPNP, as described in Table I. In order to study a large variety of systems while limiting ourselves to configurations which are relevant for the evaluation of typical-case analysis, we have randomly defined parameters so as to ensure a 50% base load (i.e., the load when sporadic tasks are ignored and only larger periods of dual-cyclic frames are considered) and a temporary load higher than 100% when shorter periods occur. More precisely: 1) the base load is randomly distributed among the non-sporadic tasks; 2) the periods of cyclic tasks (only larger periods for dual-cyclic frames) are randomly generated between 10 ms and 500 ms; 3) the (constant) execution time of each task is then computed based on the task's load and period; 4) shorter periods for dual-cyclic frames are defined, in order to ensure high load when they are used, as $1.5 * WCET$, thus making dual-cyclic tasks alone require more than 100% load when they are activated according to their shorter period; 5) the sporadic activation model of task τ_5 is deterministic; 6) bursts have a maximum size determined in terms of numbers of activations, in this case 4 or 5 (randomly chosen); 7) burst models are determined based on the shorter and larger periods of the task so as to occur less and less frequently in a sporadic fashion.

We have performed our experiments on 1000 systems generated according to the parameters that we have just described for the SPNP case. We have focused on the impact of typical-case analysis on the lowest-priority task, namely τ_6, for k-sequences of length 1000. The results are summarized in Figure 5, where the x-axis describes the ratio between $\boldsymbol{TWCRT_6}$ and $WCRT_6$ and the y-axis represents the ratio between $err(k)$ and k, in this case between $err(1000)$ and 1000. As one can see, for most of the systems (in fact 655 out of 1000) typical-case analysis yields a typical worst-case response time which is less than 50% of the original worst-case response time, and with an error ratio smaller than 10%. Furthermore let us underline that such a result is more precise than a probabilistic value, as it ensures that never more than 100 out of 1000 consecutive jobs may have a response time larger than $\boldsymbol{TWCRT_6}$.

Note that the results are similar for task τ_4 which is not influenced by the sporadic task τ_5. Furthermore we have performed similar experiments for SPP scheduling and obtained comparable results.

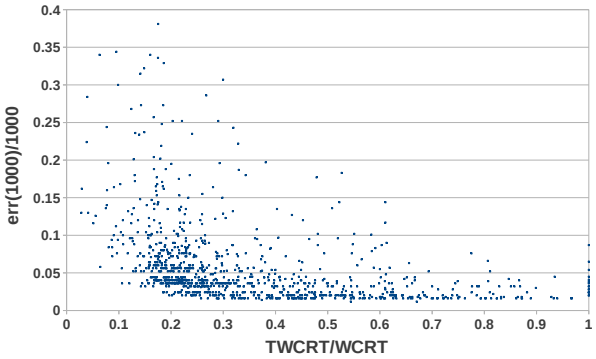

Fig. 5. Improvement w.r.t. the response time of τ_6 and corresponding error.

IX. CONCLUSION

In this paper we propose a new method for computing quantitative information about response times in uni-processor real-time systems where task activation patterns may contain sporadic bursts. We use a burst model in order to calculate how often response times may exceed the bound obtained while ignoring bursts. We have shown on experiments that this approach is well suited for the study of systems in which load is generally low but may temporarily exceed 100%. In such cases we are able to provide typical-case response times dramatically smaller than their safe counterpart while the error (i.e., how often the typical bound may be optimistic) remains at an acceptable level.

REFERENCES

[1] 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.

[2] 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.

[3] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *Proc. of DATE'12*, 2012, pp. 515–520.

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

[5] M. Traub, T. Streichert, O. Krasovytskyy, and J. Becker, "Scenario extraction for a refined timing-analysis of automotive network topologies," in *Proc. of DATE'10*. IEEE, 2010, pp. 81–86.

[6] R. I. Davis and A. Burns, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Refuted, Revisited and Revised. Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[7] K. Albers and F. Slomka, "An event stream calculus for the schedulability analysis of distributed embedded systems," in *Proc. of the International Embedded Systems Symposium*, 2009, pp. 102–114.

[8] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignement technique for streams with (m, k)-firm deadlines," *IEEE Trans. Computers*, vol. 44, no. 12, pp. 1443–1451, 1995.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] M. Negrean and R. Ernst, "Response-time analysis for non-preemptive scheduling in multi-core systems with shared resources," in *Proc. of SIES'12*, Karlsruhe, Germany, 2012.