Tighter Response-Times for Tasks with Offsets

Jukka Mäki-Turja and Mikael Nolin

Mälardalen Real-Time Research Centre (MRTC)
Mälardalen University, Box 883, SE–721 23, Västerås, Sweden
{jukka.maki-turja, mikael.nolin}@mdh.se

Abstract. We present an improvement to the analysis methods for calculating approximate response times for tasks with offsets. Our improvement calculates tighter (i.e. lower) response-times than does earlier approximation methods, and simulations show that the method, under certain conditions, calculates the exact worst-case response time.
We reveal, and exploit, a misconception in previous methods concerning the interference a higher priority task poses on a lower priority task. In this paper we show how the generally accepted concept of "released for execution" interference produces unnecessary pessimistic response times for the approximate response time analysis (RTA), presented by Tindell [1] and Palencia Gutierrez *et al.* [2]. This concept of interference does not cause any pessimism in response-time analysis for tasks without offsets (neither in the exact analysis with offsets), and has thus remained undetected over the years.
Instead, we propose the concept of "imposed" interference, which more accurately captures the interference a task causes a lower priority task. We provide formal proofs that "imposed" interference is never higher than "released for execution" interference and that it never underestimates the interference caused by higher priority tasks. We also show, by simulations on randomly generated task sets, that our improvement results in response times that outperform previous approximate methods. A typical improvement results in about 12% better admission probability (more than 30% under certain circumstances can be obtained).

1 Introduction

A powerful and well established schedulability analysis technique is the *Response-Time Analysis* (RTA) [3]. RTA is applicable to systems where tasks are scheduled in priority order which is the predominant scheduling technique used in real-time operating systems today. RTA is a method to calculate worst-case response-times for tasks in hard real-time systems. Hence, RTA can be used to perform schedulability tests, i.e., testing if tasks in a system will meet their deadlines.

In this paper, we reveal and exploit a misconception concerning the interference a task causes a lower priority *task under analysis*, one of the core concepts of RTA. The essence of this misconception is that the amount of interference a higher priority task is causing is occasionally overestimated. The misconception has its origin in the original RTA presented by Joseph and Pandya [4] for Liu and

Layland's classical task model [5]. In essence, Joseph and Pandya's RTA simulates the amount of execution-time queued in the ready-queue of an operating system, i.e. when a (higher priority) task is released for execution, its execution time is added to the response time of the task-under analysis. Hence, we call this concept for "released for execution" interference. For traditional RTA, for tasks without offsets, this concept will not cause any overestimation of calculated response times. However, as we will show in this paper, this concept is an overestimation of the interference, and when performing approximate RTA for task with offsets, it results in unnecessary pessimistic response times.

Accounting for offsets between tasks gives significantly tighter response times than using the traditional notion of a critical instant where all tasks in a system are considered to be released simultaneously [5]. In fact, many systems that will be deemed infeasible by RTA without offsets will be feasible when taking offsets into account. The first RTA for tasks with offsets was presented by Tindell [1]. He provided an exact algorithm for calculating response time for tasks with offsets. However, this algorithm becomes computationally intractable for anything but small task sets due to its exponential time complexity. In order to deal with this problem, Tindell provided an approximation algorithm, with polynomial complexity, which gives pessimistic, but safe results (worst case response times are never underestimated).

Several researchers have extended the work provided by Tindell. In this paper we focus on the approximate analysis, which was generalised and formalised by Palencia Gutierrez *et al.* [2]. They introduced dynamic offsets, allowed offsets and deadlines larger than period, and made some improvement of the approximation algorithm. Palencia Gutierrez *et al.* also provided improvements in order to calculate tighter response-times in certain situations [6]. Redell further improved their work by giving a method to calculate even lower response-times [7].

However, both improvements [6, 7] are only useful in very special circumstances where task priorities are chosen in a particular way and task jitter is extremely high.[1] Hence, their improvements are of limited generality. The focus of their methods is on finding infeasible execution orders between tasks and removing these execution orders from the set of possible critical instants. The method we present in this paper is more general and can straight forwardly be combined with the above described improvements. In fact, our approach presented here is complementary to these approaches in the sense that the most improvement for our method is achieved when jitter is low.

In this paper we present a novel interpretation of higher priority task interference: "imposed" interference, with corresponding changes to the response time formulae, which will result in less pessimistic response times for tasks with offsets using the approximation algorithm. We formally prove that response times obtained with this novel method are never greater than the method presented by

[1] Priority needs to be chosen so that transactions can "interlock" each other, and the jitter needs to be in parity with, or greater than, the task's periods. Otherwise the proposed improvements will have little or no effect.

Palencia Gutierrez *et al.* [2]. Furthermore, we also show that our method does this without the risk of ever underestimating response times.

To quantify the improvements gained with our method we present an evaluation, showing that with our method presented in this paper, one can typically gain about 15% lower response times in over 50% of the cases, resulting in a 12% higher admission probability, compared to existing approximate methods. In more extreme cases (just one transaction) about 30% higher admission probability can be obtained.

Paper Outline: In section 2 we present the pessimistic "released for execution" interference and introduce our novel concept of "imposed" interference. In section 3 we revisit and restate the original offset RTA [1, 2]. In section 4 we modify this RTA to use the concept of "imposed" interference instead, and show some consequences and proofs of correctness. Section 5 presents evaluations of our method, and finally, section 6 concludes the paper and outlines future work.

2 The Concept of Interference

Classical response time analysis for Liu and Layland's periodic task model [5] (where a task τ_i has a period T_i and worst-case execution-time C_i), presented first by Joseph and Pandya [4], states that the worst case response time, for a task under analysis (τ_i), occurs when it is released at the same time as all higher priority tasks. Under this assumption the worst case response time, R_i is:

$$R_i = C_i + \sum_{\forall j \in hp(i)} interference_j(R_i) \tag{1}$$

where C_i is the execution-time of task i, $hp(i)$ is the set of higher priority tasks, and $interference_j(t)$ is the amount of interference task j causes during time-interval t. The interference formula presented by Joseph and Pandya is [4]:

$$interference_j(t) = \left\lceil \frac{t}{T_j} \right\rceil C_j$$

where the ceiling expressions calculates the number of instances of task j. Here the full interference on each task instance (C_j) occurs immediately when the task is released. We denote this concept of interference as "released for execution" interference.

This, however, is an overestimation of the interference that τ_i actually can experience. In fact, the interference *experienced* by τ_i during a time interval can never exceed the size of the time interval. Or more precisely, the interference experienced can never grow faster than the considered interval. Formally, the derivative of the interference cannot be greater than the derivative of the time interval:

$$\frac{\mathrm{d}interference_j(t)}{\mathrm{d}t} \leq \frac{\mathrm{d}t}{\mathrm{d}t} \qquad \Rightarrow \qquad \frac{\mathrm{d}interference_j(t)}{\mathrm{d}t} \leq 1 \tag{2}$$

Theorem 1. *Consider a task τ_j, activated at time 0 and subsequently with period T_j, having execution-time C_j ($0 < C_j \leq T_j$). For a positive time-interval $t = kT_j + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_j$), $kC_j + \min(t', C_j)$ is an upper bound on the interference τ_j can impose on any lower priority task during t.*

Proof. During kT_j, τ_j imposes an amount of interference of kC_j (task instances are activated periodically), one instance for every period. During the remaining time interval, t', τ_j can, according to equation 2, never impose more interference than the length of the interval itself. Hence, $kC_j + t'$ is an upper bound on the interference τ_j, can impose during t.

However, the last instance of τ_j (when activated $t' = 0$), cannot contribute with more interference than its execution time C_j. Hence, $kC_j + C_j$ is also an upper bound on the interference τ_j can impose during t.

Combining these upper bounds (by taking the minimum of them) we get $kC_j + \min(t', C_j)$ as an upper bound on the interference τ_j can impose during t.□

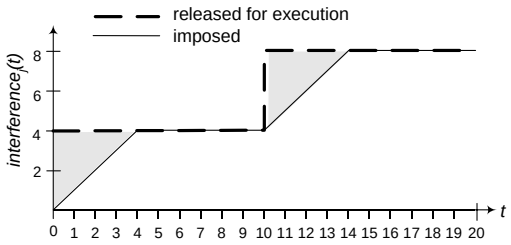


Fig. 1. Released for execution vs. imposed interference

We denote the concept of interference which is bounded by $interference_j(t)$ and theorem 1 with "imposed" interference. As an example, consider a task with $T_j = 10$ and $C_j = 4$. Figure 1 illustrates the difference between "released for execution" and "imposed" interference for $t \in 0 \ldots 20$. The released for execution interference increases in a *stepped stair* fashion, whereas the imposed interference increases in a *slanted stair* fashion (with a derivative of 1 in the slants).

In figure 1 the shaded areas represent the overestimation made by the released for execution concept. For classical response-time analysis this overestimation has no effect on the calculated response-time, and Joseph and Pandya's equation does yield exact worst case response-times. The reason for this is that the response-time analysis calculation (which is done by fix-point iteration) has no solutions in the shaded areas (as discussed further in section 4.3). Also for exact RTA of task with offsets [1] this overestimation does not yield any pessimism in the calculated response-times.

3 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets [1, 2] and illustrates some intuition behind the analysis and the formulas.

3.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i (for non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ tasks, and each task is activated (released for execution) when a relative time, *offset*, elapses after the arrival of the external event.

We use τ_{ij} to denote a task. The first subscript denotes which transaction the task belongs to, and the second subscript denotes the number of the task within the transaction. A task, τ_{ij}, is defined by a worst case execution time (C_{ij}), an offset (O_{ij}), a deadline (D_{ij}), maximum jitter (J_{ij}), maximum blocking from lower priority tasks (B_{ij}), and a priority (P_{ij}). The system model is formally expressed as follows:

$$\begin{aligned}\Gamma &:= \{\Gamma_1, \ldots, \Gamma_k\} \\ \Gamma_i &:= \langle \{\tau_{i1}, \ldots, \tau_{i|\Gamma_i|}\}, T_i \rangle \\ \tau_{ij} &:= \langle C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} \rangle\end{aligned}$$

There are no restrictions placed on offset, deadline or jitter, i.e., they can each be either smaller or greater than the period.

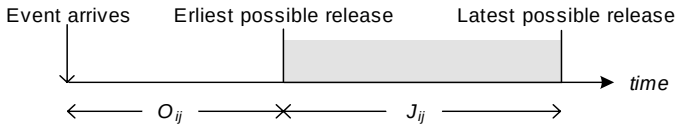


Fig. 2. Relation between an event arrival, offset, jitter and task release

The relation between event arrival, offset, jitter and task release is graphically visualised in figure 2. After the arrival of the event the task τ_{ij} is never released for execution until its offset (O_{ij}) has elapsed. The release may be delayed by jitter (maximally until $O_{ij} + J_{ij}$) making its exact release uncertain. For a more extensive explanation of task parameters see [2]. Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in figure 3.

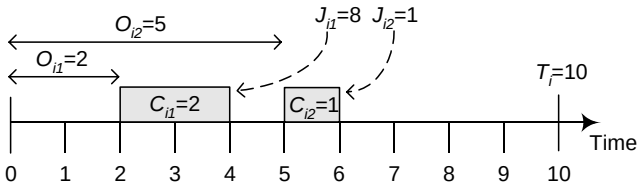


Fig. 3. An example transaction Γ_i

3.2 Response-time analysis

The goal of RTA is to facilitate a schedulability test for each task in the system by calculating an upper bound on its worst case response-time. We use τ_{ua} (task a, belonging to transaction Γ_u) to denote the *task under analysis*, i.e., the task who's response time we are currently calculating.

In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher (or equal) priority tasks [4, 5]. In a task model with offsets this assumption yields pessimistic response-times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell [1] relaxed the notion of critical instant to be:

> At least one task in every transaction is to be released at the critical instant. (Only tasks with priority higher or equal to τ_{ua} are considered.)

Since it is not known which task that coincides with (is released at) the critical instant, every task in a transaction must be treated as a *candidate* to coincide with the critical instant.

Tindell's exact RTA tries every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets (the number of possible combinations of candidates is m^n for a system with n transactions and with m tasks per transaction). Therefore Tindell provided an approximate RTA that still gives good results but uses one single approximation function for each transaction. Palencia Gutierrez *et al.* [2] formalised and generalised Tindell's work. We will in this paper use the more general formalism of Palencia Gutierrez *et al.*, although our proposed method is equally applicable to Tindell's original algorithm.

3.3 Interference function

Central to RTA is to capture the interference a higher or equal priority task (τ_{ij}) causes the task under analysis (τ_{ua}) during an interval of time t. Since a task can interfere with τ_{ua} multiple times during t, we have to consider interference from possibly several *instances*. The interfering instances of τ_{ij} can be classified into two sets:

$Set1$ Activations that occur before or at the critical instant and that can be delayed by jitter so that they coincide with the critical instant.

$Set2$ Activations that occur after the critical instant

When studying the interference from an entire transaction Γ_i, we will consider each task, $\tau_{ic} \in \Gamma_i$, as a *candidate* for coinciding with the critical instant.

RTA of tasks with offsets is based on two fundamental theorems:

1. The worst case interference a task τ_{ij} causes τ_{ua} is when $Set1$ activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.
2. The task of Γ_i that coincide with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

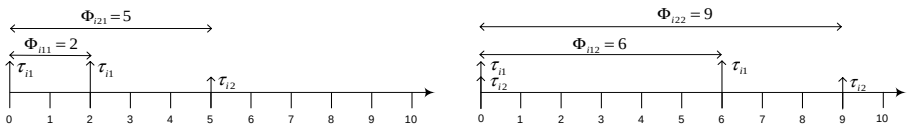
The phasing between a task, τ_{ij}, and a critical instant candidate, τ_{ic}, becomes (slightly reformulated compared to [2], see Appendix A):

$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \bmod T_i \tag{3}$$

From the second theorem we get that τ_{ic} will coincide with the critical instant after having experienced its worst case jitter delay, i.e., the critical instant will occur at $(O_{ic} + J_{ic}) \bmod T_i$, relative to the start of Γ_i. From this, the definition of Φ_{ijc} follows in order to keep the relative phasing (of releases) among tasks within Γ_i. An implication of this is that the first instance of a task τ_{ij} in $Set2$ will be released at Φ_{ijc} time units after the critical instant, and subsequent releases will occur periodically every T_i.

Figure 4 illustrates the four different Φ_{ijc}-s that are possible for our example transaction in figure 3. The upward arrows denote task releases. The height of the upward arrows denotes the amount of execution released.

Figure 4(a) shows for the case that τ_{i1} coincides with the critical instant, the invocations in $Set1$ (arriving at time 0) and the first invocations in $Set2$. Figure 4(b) shows the corresponding situation when τ_{i2} is the candidate to coincide with the critical instant.



(a) $\tau_{ic} = \tau_{i1}$ (b) $\tau_{ic} = \tau_{i2}$

Fig. 4. Φ-s for the two candidates in Γ_i

Given the two sets of task instances ($Set1$ and $Set2$) and the corresponding phase relative to the critical instant (Φ_{ijc}), the interference caused by task τ_{ij} can be divided into two parts:

1. The part caused by instances in $Set1$ (which is independent of the time interval t), I_{ijc}^{Set1}, and

2. the part caused by instances in $Set2$ (which is a function of the time interval t), $I_{ijc}^{Set2}(t)$.

These are defined as follows:

$$I_{ijc}^{Set1} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \qquad I_{ijc}^{Set2}(t) = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij} \tag{4}$$

The interference transaction Γ_i poses on τ_{ua}, during a time interval t, when candidate τ_{ic} coincides with the critical instant, is:

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) \tag{5}$$

Where $hp_i(\tau_{ua})$ denotes tasks belonging to transaction Γ_i, with priority higher or equal to the priority of τ_{ua}.

3.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination [1, 2]. However, since this is computationally intractable for anything but small task sets, the approximate analysis defines one single, upward approximated, function for the interference caused by transaction Γ_i [1, 2]:

$$W_i^*(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t) \tag{6}$$

That is, $W_i^*(\tau_{ua}, t)$ simply takes the maximum of each interference function (for each candidate τ_{ic}).

As an example, consider again transaction Γ_i depicted in figure 3. Figure 5 shows the interference function for the two candidates (W_{i1} and W_{i2}), and it shows how W_i^* is derived from them by taking the maximum of the two functions at every t.

Given the interference (W_i^*) each transaction causes, during a time interval of length t, the response time of τ_{ua} (R_{ua}) can be calculated. Appendix A shows how to perform these response-time calculations.

4 Tight offset RTA

We begin this section with an illustrative example of how the original analysis overestimates the response-time. Consider a simple transaction Γ_i depicted in figure 6 where jitter (J_{ij}) and blocking (B_{ij}) is zero.

Also consider a lower priority task, τ_{ua}, which is the single task in transaction Γ_u, with $C_{ua} = 2$. For this simplified task model where $B_{ij} = J_{ij} = 0$, $D_{ua} \leq T_u$ only one instance of the task under analysis is active at any point in

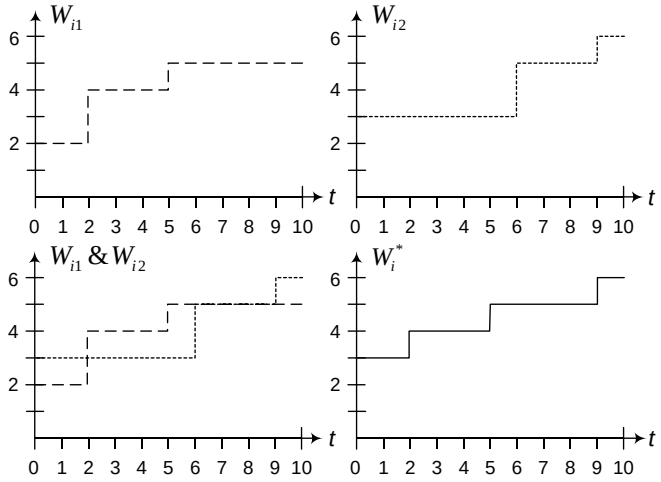


Fig. 5. $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions

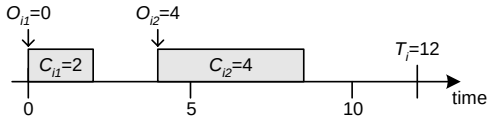


Fig. 6. A simple example transaction

time. This means that the response time formulas, for the single lower priority task, presented in appendix A, can be reduced and simplified to:

$$R_{ua} = C_{ua} + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, R_{ua}) \tag{7}$$

The response-time calculation is performed by means of fix-point iteration (starting with $R_{ua} = 0$) as follows:

Iter#	t	W_{i1}	W_{i2}	W_i^*	R_{ua}
0					0
1	0	0	0	0	2
2	2	2	4	4	6
3	6	6	4	6	8
4	8	6	4	6	8

Where column "Iter#" denotes the iteration number, "t" the time interval, "W_{i1}" and "W_{i1}" denotes $W_{ic}(\tau_{ua}, t)$ for the two candidate tasks τ_{i1} and τ_{i2} respectively. "W_i^*" denotes the value of $W_i^*(\tau_{ua}, t)$, and "R_{ua}" the calculated response-time for the iteration. In iteration number 4 the fix-point iteration terminates (R_{ua}

has the same value as in the previous iteration), and the calculated response time is $R_{ua} = 8$. However, it can easily be seen that a task with $C_{ua} = 2$ can never be preempted by both tasks τ_{i1} and τ_{i2} since both tasks are separated by at least 2 units of idle time. Hence, the actual worst case response-time is $R_{ua} = 6$ and the response-time is overestimated.

4.1 Using Imposed Interference

One property of the ceiling expression of $I_{ijc}^{Set2}(t)$ in equation 4 is that it returns the amount of interference "released for execution" at time t. This result in a *stepped stair* interference function. If we modify $I_{ijc}^{Set2}(t)$ in equation 4 so that it returns the interference "imposed" on τ_{ua} we get a *slanted stair* function (as proposed in section 2). The two slanted stair functions for our simple example transaction from figure 6 are shown in figures 8(a) and 8(b).

The slanted stairs are obtained by modifying $I_{ijc}^{Set2}(t)$ defined in equation 4 so that the "last" task instance, of the periodically activated tasks in $Set2$, does not interfere with its full execution time unless the interval t is sufficiently large. Our redefined version of $I_{ijc}^{Set2}(t)$ is:

$$
\begin{aligned}
I_{ijc}^{Set2}(t) &= \left\lceil \frac{t^*}{T_i} \right\rceil C_{ij} - x \\
x &= \begin{cases} C_{ij} - (t^* \bmod T_i) & \text{if } t^* > 0 \wedge \left(0 < t^* \bmod T_i < C_{ij}\right) \\ 0 & \text{otherwise} \end{cases} \\
t^* &= t - \Phi_{ijc}
\end{aligned}
\tag{8}
$$

where Φ_{ijc} is defined in equation 3 and x is used to generate the slants of the "imposed" interference function. Figure 7(a) illustrates a sequence of task releases, and figure 7(b) shows how the value of x varies accordingly.

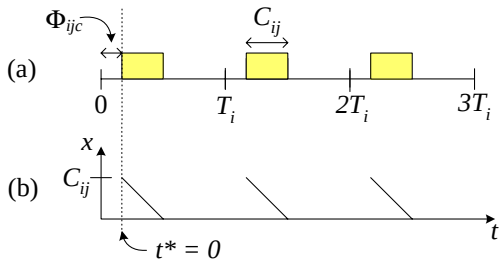


Fig. 7. Relation between task release and x

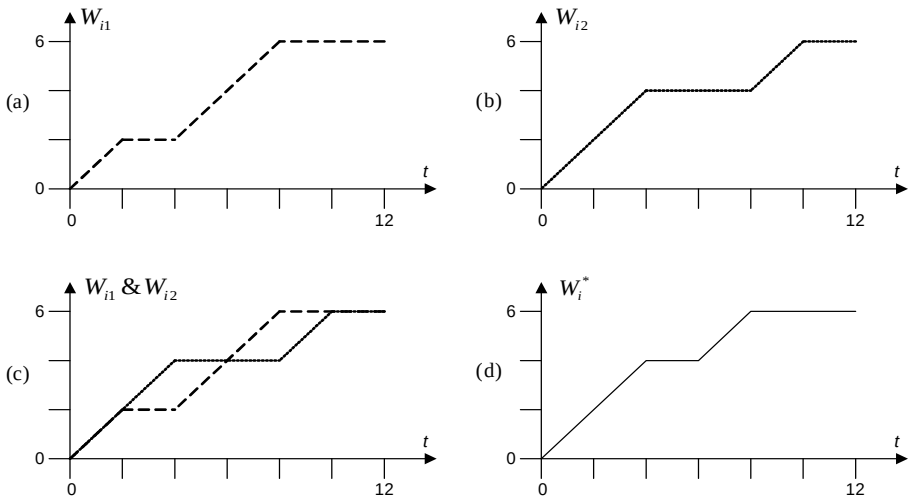


Fig. 8. Interference *imposed* by our example transaction

The slanted stairs generated by equation 8 are shown in figures 8(a) and 8(b), and figure 8(c) shows them overlaid. Using our new version of $I_{ijc}^{Set2}(t)$ in equation 5 we get the maximised slanted stairs interference function, representing the approximation function W_i^*, shown in figure 8(d).

With the new definition of interference in equation 8 we can now use equation 7 to calculate a new response-time R_{ua} for our example as follows:

Iter#	t	W_{i1}	W_{i2}	W_i^*	R_{ua}
0					0
1	0	0	0	0	2
2	2	2	2	2	4
3	4	2	4	4	6
4	6	4	4	4	6

We note that our new definition of $I_{ijc}^{Set2}(t)$ makes the analysis able to "see" the empty slot between tasks τ_{i1} and τ_{i2}, something the original analysis overlooked.

Hence, the calculated response-time (6) is lower than that of the original analysis (8), and in section 5 we will quantify this improvement in more general terms.

4.2 Correctness Criteria

For our proposed modification to $I_{ijc}^{Set2}(t)$ in equation 8 to be correct, and not produce greater response-times than the original analysis, three criteria have to be fulfilled:

– The new definition of $I_{ijc}^{Set2}(t)$ is not allowed to be greater than the old definition (for any t). If this condition holds, the analysis performed with the new definition is guaranteed not to yield larger response-times than the old definition does.
– The new definition of $I_{ijc}^{Set2}(t)$ must not underestimate the interference caused by $Set2$-tasks. If the interference is underestimated, analysis performed with the new definition could yield unsafe response-time estimates.
– The new definition of $I_{ijc}^{Set2}(t)$ must yield a monotonically increasing interference function $W_i^*(\tau_{ua}, t)$. Monotonicity is required to guarantee that at least one solution to the response-time formula exists and that fix-point iteration finds the smallest existing solution [8].

Theorem 2. *For a given task under analysis, τ_{ua}, and one candidate task, $\tau_{ic} \in \Gamma_i$, our new definition of I_{ijc}^{Set2} (equation 8) is never greater than the old definition (equation 4).*

Proof. x, as defined in equation 8, is used to decrease the calculated value of I_{ijc}^{Set2}. Since x, by definition, is never negative, it can never contribute to making equation 8 greater than equation 4. □

Theorem 3. *For any time interval $t \geq 0$ our new definition of $I_{ijc}^{Set2}(t)$ (equation 8) never underestimates the interference caused by $Set2$ task instances.*

Proof. $Set2$ task instances arrive periodically (per definition) with period T_i, with the first instance arriving at Φ_{ijc}.

We first treat the time before the first invocation in $Set2$, i.e. $t < \Phi_{ijc}$. During this time interval $t^* < 0$ and hence $x = 0$. Since, $t < \Phi_{ijc} < T_i$ then $t^* > -T_i$ and the ceiling expression in equation 8 evaluates to zero. Hence, the whole equation 8 is also zero. Since the interference before the first invocation obviously is zero, equation 8 does not underestimate the interference before the first invocation.

For times at or after the first invocation, i.e. $t \geq \Phi_{ijc}$, we have $t^* \geq 0$. Now, assume $t^* = kT_i + t'$, where $k \in \mathbb{N}$ and $0 \leq t' < T_i$ (the relation between t, t^* and t' is graphically visualised in figure 9). If the interference calculated by equation 8 is not below the *safe upper bound* defined by theorem 1:

$$\textit{safe upper bound} = kC_{ij} + \min(t', C_{ij})$$

then the interference is not underestimated.

We divide the proof into three cases depending on the value of t' for a time-interval t (the three different cases are depicted graphically in figure 10):

– $t' \geq C_{ij}$: The ceiling expression in equation 8 evaluates to $k + 1$ and the interference is thus $(k+1)C_{ij} - x$. Further, when $t' \geq C_{ij}$ then $t^* \bmod T_i \geq C_{ij}$ resulting in $x = 0$, hence the interference is $(k + 1)C_{ij}$, which is not below *safe upper bound*.

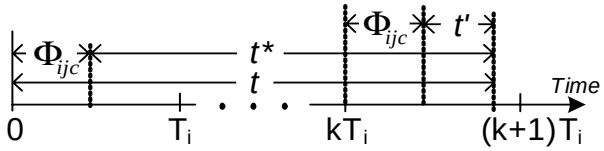


Fig. 9. Relation between t, t^* and t'

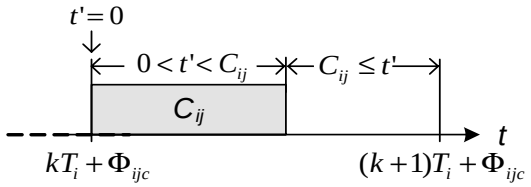


Fig. 10. Three proof cases for t'

- $0 < t' < C_{ij}$: The ceiling expression in equation 8 evaluates to $k+1$ and the interference is thus $(k+1)C_{ij} - x = kC_{ij} + C_{ij} - x$. Further, when $0 < t' < C_{ij}$ then $0 < t^* \bmod T_i < C_{ij}$ and $x = C_{ij} - t'$, hence the interference is $kC_{ij} + C_{ij} - (C_{ij} - t') = kC_{ij} + t'$, which is not below *safe upper bound*.
- $t' = 0$: The ceiling expression in equation 8 evaluates to k and the interference is thus $kC_{ij} - x$. Further, when $t' = 0$ then $t^* \bmod T_i = 0$ and $x = 0$, hence the interference is kC_{ij}, which is not below *safe upper bound* (since $t' = 0$).□

Theorem 4. *Our new definition of $I_{ijc}^{Set2}(t)$ (equation 8) is (non-strictly) monotonically increasing with the time interval t.*

Proof. We prove this by showing that the derivative of equation 8 is never negative. First, we conclude that a negative derivative of x cannot contribute to make the derivative of equation 8 negative (since x is *subtracted* in equation 8). We also conclude that if x is disregarded (i.e. assumed to be 0), then equation 8 does not have a negative derivative in any point.

We divide the proof into three cases, depending on the value of $t^* \bmod T_i$ for times t:

- $t^* \bmod T_i \geq C_{ij}$: In this case x is continuously 0, hence the derivative of x is 0, and equation 8 cannot have a negative derivative.
- $0 < t^* \bmod T_i < C_{ij}$: In this case the derivative of x is -1, hence the derivative of equation 8 cannot have a negative derivative.
- $t^* \bmod T_i = 0$: For this case we conclude that equation 8 is continuous, since at time $t+\epsilon$ (for an arbitrary small and positive ϵ) the ceiling expression has increased with C_{ij} and x has increased with $C_{ij} - \epsilon$, hence equation 8 has increased with exactly ϵ. Thus, the derivative of equation 8 at such times t is 1. □

4.3 Discussion

At first glance, it is not obvious that lowering the interference function $W_{ic}(\tau_{ua}, t)$ should automatically give lower response-times. In fact, the stepped-stair interference function has been used for many years to represent the interference in RTA [3, 9], without introducing any pessimism.

The reason stepped stairs (in analysis without offsets) does not introduce pessimism can be found in our previous work [8]. In short, the fix-point iteration will terminate when the sum of all interference functions (demand) meets the line from origin with slope 1 (supply). Hence, replacing stepped stairs with slanted stairs (with slope 1) will not contribute to earlier fix-point convergence.

However, in approximate response-time analysis with offsets, the interference functions, W_{ic}-s, are not used directly in the fix-point iterations. Instead they are first subjected to a maximisation function (equation 6). This situation can be compared to floating point addition: if you round up the floating point numbers at each calculation step, instead of just in the end, you will loose precision. This corresponds to passing released for execution interference, instead of more precise imposed interference, to the maximisation function. Another view of this is that by using slanted-stair functions as input to the maximisation function, one essentially "delays" the time it takes for one low-interference scenario to overtake a high-interference scenario.

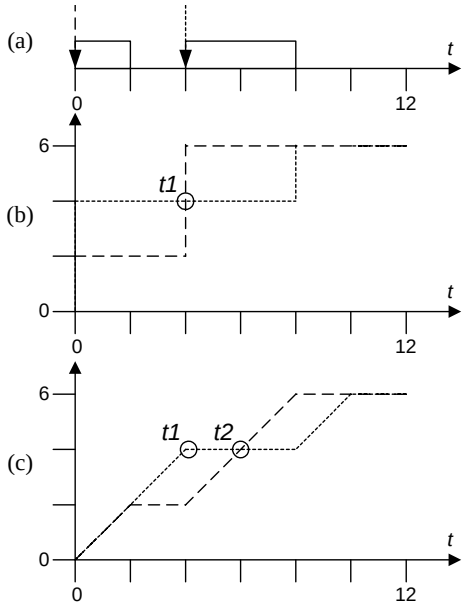


Fig. 11. Stepped stairs vs. slanted stairs

Figure 11(a) shows our simple example transaction from figure 6 with two arrows denoting the two possible scenarios for the critical instant (one "dashed" scenario and one "dotted" scenario). Figures 11(b) and 11(c) shows the stepped stairs and slanted stairs interference functions, respectively, for both scenarios. For times $t < t1$, the dotted scenario is the one with highest interference. Time $t1$ corresponds to the release of the second task in the dashed scenario. For the stepped stairs case, this means immediately adding another 4 units of interference to the dashed scenario, hence immediately making it the scenario with the highest interference. However, for the slanted stairs case, the time $t1$ means that the dashed line starts to increase, but not until time $t2$ it catches up with the dotted scenario. Hence, the interval between $t1$ and $t2$ represents the time by which the slanted stairs "delay" the dashed scenario to catch up with the dotted scenario. If fix-point convergence can be achieved during this interval, then RTA with imposed interference will calculate a lower response time than does RTA with released for execution interference.

5 Evaluation

In order to evaluate and quantify our proposed improvement, we have implemented the approximate response-time equations of appendix A, using both the original definition of $I_{ijc}^{Set2}(t)$ from section 3 and our tighter version of $I_{ijc}^{Set2}(t)$ from section 4. Furthermore, we have also, as a comparison, implemented the exact analysis.

Using these implementations and a task-generator we have performed simulations of all three approaches by calculating the response time for a single low priority task, e.g., corresponding to an admission control situation.

5.1 Description of Task Generator

In our simulator we generate task sets that are used as input to the different implementations. The task-set generator takes the following parameters as input:

- Total system load (in % of total CPU utilisation),
- The number of transactions to generate,
- The number of tasks per transaction to generate, and
- Jitter fraction (in % of the transaction periods).

Using these parameters a task set with the following properties is generated:

- The total system load is proportionally distributed over all transactions.
- Periods (T_i) are randomly distributed in the range 1.000 to 1.000.000 time units (uniform distribution).
- Each offset (O_{ij}) is randomly distributed within the transaction period (uniform distribution).
- The execution times (C_{ij}) are chosen as a fraction of the time between two consecutive offsets in the transaction. The fraction is the same throughout one transaction, and is selected so that the transaction load (as defined by the first property) is obtained.

– The jitter is set to the jitter fraction of the period ($J_{ij} = f * T_i$).
– Blocking (B_{ij}) is set to zero.
– The priorities are assigned in rate monotonic order [5].

5.2 Description of Simulation Setup

The heart of the improvement made to the approximate response time analysis is a new definition of $I_{ijc}^{Set2}(t)$. We have implemented the response-time equations of appendix A which will show the effects of our improvements in a realistic scenario. However, neither interference from other tasks in Γ_u nor interference from previous instances of τ_{ua} comes into play in the admission control situation that we simulate. Taking interference from other tasks of Γ_u into account would yield less improvement of our methods, since W_i^* is not used for them (see appendix A).

The setup of the simulation is as follows: a task set is generated according to input parameters (system load, number of tasks within a transaction, number of transactions, jitter). To simulate an admission control situation, we calculate the response time for a low priority task subjected to admission control.

We have calculated and compared response times for our tighter analysis (Tight), Palencia Gutierrez *et al.*'s original analysis (Orig) and the exact analysis (Exact). The results in section 5.3 have been obtained by taking the mean value from 1000 generated task-sets for each point in each graph. The graphs in the left and in the right columns also show the 95% confidence interval for these mean values.

We have measured three metrics from the simulations:

– "Admission probability (%)" — This metric measures the fraction of cases, out of the 1000 generated task sets, the admission control task passes the admission test (its response time is lower than its deadline).
– "Response-time improvement (%)" — This metric measures the average and maximum improvement (over Original) in response time for the task subjected to admission control. Improvement in response time for the tight analysis, R_{ua}^{Tight}, is defined as $1 - R_{ua}^{\text{Tight}}/R_{ua}^{\text{Orig}}$ (and analogous for the Exact analysis). Note that for this metric the original acts as baseline and thus only maximum and average improvement of (Tight) and (Exact) (over (Orig)) are plotted. Also note that the maximum value is one value (the maximum) out of 1000, which makes the behaviour in these graphs statistically uncertain (they show what is possible without quantifying probability of occurrence).
– "Fraction of tasks with improvement (%)" — This metric measures the fraction of admission control tasks that results in a lower response time, compared to the original analysis (Orig). As for previous metric, the original approximate analysis is used as a baseline, hence no curve is plotted for that method. Note that this metric says nothing about the size of the improvements.

The first metric is to show what effect an improvement in response time could have in a realistic situation. The purpose of the last two metrics is to quantify the difference in response time between the three analysis methods.

5.3 Simulation Results

In the simulations we have varied our four task-generator parameters in different ways. Figure 12 shows a subset of the simulation results. The exact analysis can only be run on small task sets; hence it is not present for larger tasks sets. For every parameter that is varied we show all three metrics described in the previous section, corresponding to column one, two and three respectively in figure 12. (In figure 12, note that "Tasks = x" denotes "x tasks/transaction".)

Figures 12 (a–i) corresponds to a base configuration where the number of tasks per transaction is 6, the number of transactions is 3, system load is 80% and the load of task under admission control is 2%. From this base configuration we vary the number of tasks/transaction (a–c), number of transactions (d–f), jitter (g–i), while keeping the other parameters constant.

Figures (a–c) shows the results when the number of tasks is varied between 1 and 13. For more than 5 tasks we can see in (a), that the admission probability for (Tight) is around 12% higher than for (Orig). In (b) we see that the average response time improvement of (Tight) is for 10 tasks over 15%, and that there are task sets (although rare) where improvement of more than 50% can be obtained. In (c) we see that when the number of tasks grows, so does the probability of a response time improvement.

For figures (d–f), where the number of transactions is varied, a quite different picture emerges. The difference between (Orig) and (Tight) gets smaller as the number of transactions grows. This is not surprising, since in the case where the tasks/transaction ratio approaches 1, there are very few offset relations among tasks and the analysis approaches the analysis for tasks without offsets.

Figures (g–i) show what happens when jitter is varied. Not only does the admission probability decrease drastically, but also the relative improvement of (Tight) over (Orig). This is mainly due to the fact that jitter contributes to I_{ijc}^{Set1}, whereas our improvement only affects $I_{ijc}^{Set2}(t)$. As I_{ijc}^{Set1} account for an increasingly larger fraction of the total response-time, the relative improvement of (Tight) decreases. However, the absolute response-time improvement (not shown) and the number of improvements (figure (i)) is not noticeably affected by the jitter. As the jitter grows larger than the period (or larger than several periods) the effects of our improvements diminish further. However, systems with such large jitter are rare in control-systems (which constitute the majority of real-time systems), where the jitter is typically only allowed to be a few percent of the period. Also, for system with such large jitters (such as multimedia applications), other methods [6, 7] to reduce the estimated response-time can be used.

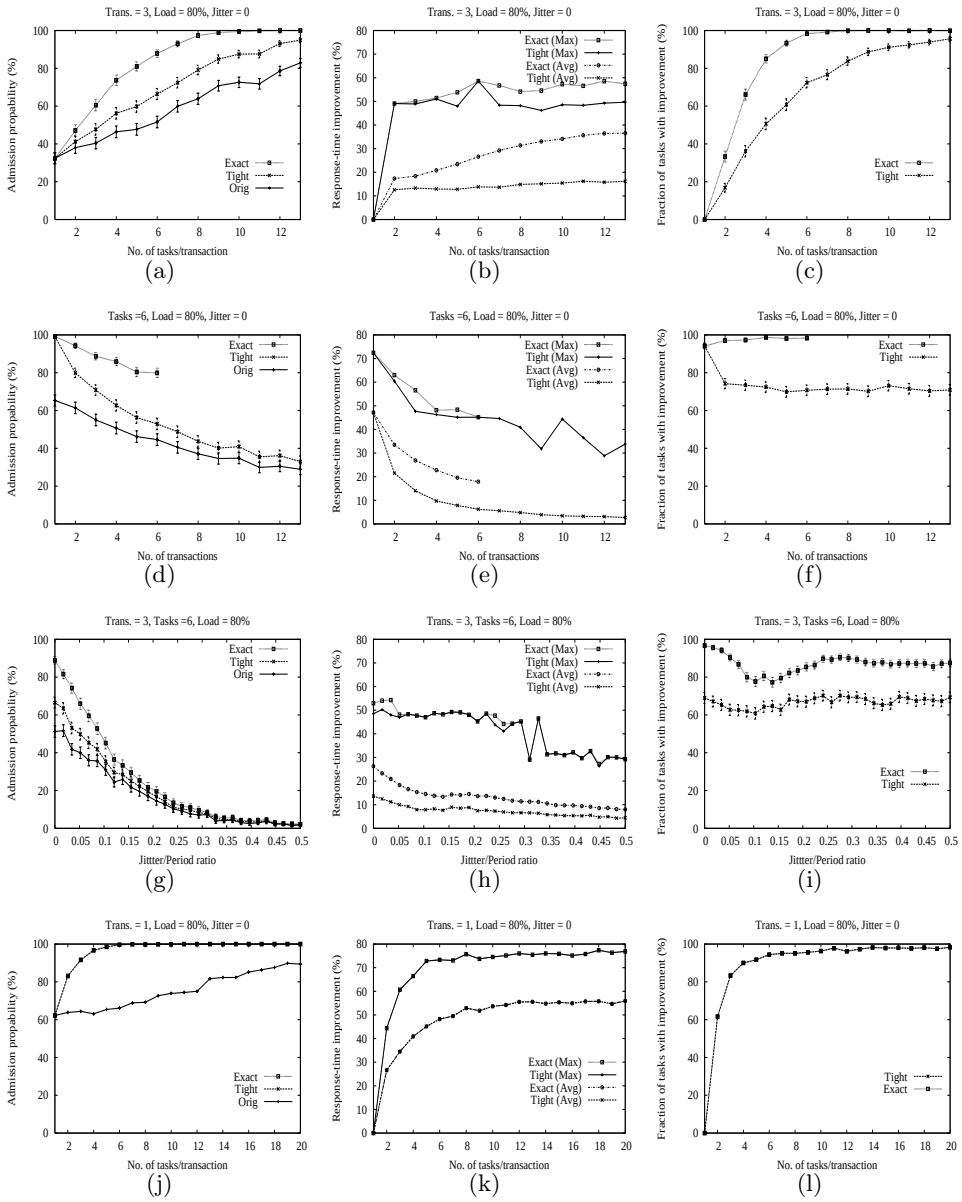


Fig. 12. Simulation Results

Finally, figures 12(j–l) correspond to a configuration where the number of transactions is 1, system load 80%, and load of the task under admission control is 2%. This type of scenario would occur in a system using a hybrid scheduling method, supporting both static cyclic scheduled tasks (corresponding to the single high priority transaction) and priority scheduled tasks running in the background of the static schedule [10]. This situation shows where our method excels. All tasks have offset relations among them, resulting in well over 30% better admission probability (4–9 tasks) over (Orig) and an average improvement of over 50% when the number of tasks/transaction is more than 8. Another interesting thing is that (Exact) and (Tight) always yield exact response-times. This comes from the fact that when considering a transaction in isolation (no interference among several transactions) the slanted stair interference function captures the worst case interference exactly.

6 Conclusions and Future Work

We have presented an improvement that calculates tighter (lower) response-times than does earlier approximation methods. We prove that our method never calculates greater response-times than the method in [2]. Furthermore we prove that our method never underestimates the interference caused by higher priority tasks. Hence, it calculates a safe and tight approximation of the actual worst-case response-time.

We exploit a misconception in previous methods concerning the interference a task poses on a lower priority one. The concept "imposed" interference is introduced, and is shown to more accurately capture this interference compared to the previously accepted concept of "released for execution" interference. This situation is analogous to floating point addition where "released for execution" interference corresponds to calculations with integer values (rounded up) whereas "imposed" interference corresponds to calculations with the more accurate floating point values (resulting in a lower total sum).

Simulations show that the improvement is significant (especially when tasks/transaction ratio is high), typically about 15% tighter response times in 50% of the cases, resulting in 12% higher admission probability for low priority task subjected to admission control. In certain circumstances the improvement is much greater, and with just one transaction (corresponds to a static schedule) our proposed method calculates exact response times.

Our tighter analysis is noticeably slower than the original analysis (even slower than the exact for small task sets). This is a natural effect of using tighter interference functions since it gives slower fix-point convergence. However, we have previously proposed a method to speed up the original analysis [11]. In our future work we will adapt that method to our tight analysis. We will also incorporate complementary improvements to RTA for tasks with offsets such as [6, 7].

References

1. Tindell, K.: Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England (1992)
2. Palencia Gutierrez, J., Gonzalez Harbour, M.: Schedulability Analysis for Tasks with Static and Dynamic Offsets. In: Proc. 19^{th} IEEE Real-Time Systems Symposium (RTSS). (1998)
3. Audsley, N., Burns, A., Davis, R., Tindell, K., Wellings, A.: Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. Real-Time Systems **8** (1995) 129–154
4. Joseph, M., Pandya, P.: Finding Response Times in a Real-Time System. The Computer Journal **29** (1986) 390–395
5. Liu, C., Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the ACM **20** (1973) 46–61
6. Palencia Gutierrez, J., Gonzalez Harbour, M.: Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In: Proc. 20^{th} IEEE Real-Time Systems Symposium (RTSS). (1999) 328–339
7. Redell, O.: Accounting for Precedence Constraints in the Analysis of Tree-Shaped Transactions in Distributed Real-Time Systems. Technical Report TRITA-MMK 2003:4, Dept. of Machine Design, KTH (2003)
8. Sjödin, M., Hansson, H.: Improved Response-Time Calculations. In: Proc. 19^{th} IEEE Real-Time Systems Symposium (RTSS). (1998) URL: http://www.docs.uu.-se/~mic/papers.html.
9. Audsley, N., Burns, A., Tindell, K., Richardson, M., Wellings, A.: Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. Software Engineering Journal **8** (1993) 284–292
10. Mäki-Turja, J., Sjödin, M.: Combining Dynamic and Static Scheduling in Hard Real-Time Systems. Technical Report MRTC no. 71, Mälardalen Real-Time Research Centre (MRTC) (2002)
11. Mäki-Turja, J., Nolin, M.: Faster Response Time Analysis of Tasks With Offsets. In: Proc. 10^{th} IEEE Real-Time Technology and Applications Symposium (RTAS). (2004)

A Complete RTA formulas

In this appendix we provide the complete set of formulas to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutierrez *et al.* [2].

The interference transaction Γ_i poses on a lower priority task, τ_{ua}, if τ_{ic} coincides with the critical instant, is defined by (see equation 5 in this paper):

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(\left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor + \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil \right) C_{ij} \qquad \text{(26 in [2])}$$

where the phase between task τ_{ij} and the candidate critical instant task τ_{ic} is defined as (see equation 3 in this paper):

$$\Phi_{ijc} = T_i - (O_{ic} + J_{ic} - O_{ij}) \bmod T_i \qquad \text{(17 in [2])}$$

The approximation function for transaction Γ_i which considers all candidate τ_{ic}-s simultaneously, is defined by (see equation 6 in this paper):

$$W_i^*(\tau_{ua}, w) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, w) \qquad \text{(27 in [2])}$$

The length of a busy period, for τ_{ua}, assuming τ_{uc} is the candidate critical instant, is defined as (Note that the approximation function is not used for Γ_u):

$$\begin{aligned} L_{uac} =& B_{ua} + (p - p_{0,uac} + 1)C_{ua} + \\ & W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac}) \end{aligned} \qquad \text{(30 in [2])}$$

where $p_{0,uac}$ denotes the first, and $p_{L,uac}$ the last, task instance, of τ_{ua}, activated within the busy period. They are defined as:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \qquad \text{(29 in [2])}$$

and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil \qquad \text{(31 in [2])}$$

In order to get the worst case response time for τ_{ua}, we need to check the response time for every instance, $p \in p_{0,uac} \ldots p_{L,uac}$, in the busy period. Completion time of the p'th instance is given by:

$$\begin{aligned} w_{uac}(p) =& B_{ua} + (p - p_{0,uac} + 1)C_{ua} \\ & + W_{uc}(\tau_{ua}, w_{uac}(p) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p)) \end{aligned} \qquad \text{(28 in [2])}$$

The corresponding response time (for instance p) is then:

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua} \qquad \text{(32 in [2])}$$

To obtain the worst case response time, R_{ua}, for τ_{ua}, we need to consider every candidate critical instant ,τ_{uc} (including τ_{ua} itself), and for each such candidate every possible instance, p, of τ_{ua}:

$$R_{ua} = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} \left[\max_{p = p_{0,uac}, \ldots, p_{L,uac}} (R_{uac}(p)) \right] \qquad \text{(33 in [2])}$$