Fast and Tight Response-Times for Tasks with Offsets

Jukka Mäki-Turja Mikael Nolin
Mälardalen Real-Time Research Centre (MRTC)
Västerås, Sweden
jukka.maki-turja@mdh.se

Abstract

In previous work, we presented a tight approximate response-time analysis for tasks with offsets. While providing a tight bound on response times, the tight analysis exhibits similarly long execution times as does the traditional methods for calculating response-times for tasks with offsets. The existing method for fast analysis of tasks with offsets is not applicable to the tight analysis.

In this paper we extend the fast analysis to handle the distinguishing trait of the tight analysis; continuously increasing interference functions. Furthermore, we provide another speedup; by introducing pessimism in the modelling of interference at certain points, we speed up the convergence of the numerical solving for response-times without increasing the pessimism of the resulting response-times.

The presented fast-and-tight analysis is guaranteed to calculate the same response-times as the tight analysis, and in a simulation study we obtain speedups of more than two orders of magnitude for realistically sized tasks sets compared to the tight analysis. We also demonstrate that the fast-and-tight analysis has comparable execution time to that of the fast analysis. Hence, we conclude that the fast-and-tight analysis is the preferred analysis technique when tight estimates of response-times are needed, and that we do not need to sacrifice tightness for analysis speed; both are obtained with the fast-and-tight analysis.

1 Introduction

Response-Time Analysis (RTA) [1] is a powerful and well established schedulability analysis technique. RTA is a method to calculate upper bounds on response-times for tasks in hard real-time systems. In essence RTA is used to perform a schedulability test, i.e., checking whether or not tasks in the system will satisfy their deadlines. RTA is applicable for, e.g., systems where tasks are scheduled in priority order which is the predominant scheduling technique used in real-time operating systems today.

Fast RTA has several practical implications, e.g., facilitating the use of response time calculations in an iterative workflow including automatic priority assignment and/or task allocation, or for admission control in on-line scheduling algorithms. Tighter response time allow for more efficient hardware utilization. Consequently, analysis speed and tight response time are desirable features in engineering resource constrained real-time systems.

To be able to calculate less pessimistic response times in systems where tasks may have dependencies in their release times, Tindell introduced RTA for a task model with offsets [11]. Palencia and Harbour formalized and extended the work of Tindell in [7]. In [5] we have shown that the RTA for task with offset presented in their work calculates unnecessarily pessimistic response-times. As a remedy, we presented our tight analysis. The main source for this improvement comes from more accurate modelling of inter-task interference. In [7, 11] the interference only increases at discrete points in time, whereas in our tight analysis the interference can increase continuously over time. There is, however, a slight price to pay for this accuracy, slower fix-point convergence which can result in longer analysis time.

In this paper we extend our previous fast analysis for tasks with offsets [4] to enable its application to the tight analysis, providing a new method that calculates tight response times at fast analysis speed. The fast analysis has been shown to achieve two orders of magnitude speedup for realistically sized task sets [4]. The essence of this approach is to statically store the discrete points in time during the first period where the interference increases, and during equation solving use a simple and fast table lookup.

However, the approach taken in [4] is not directly applicable to the tight analysis since it uses a more accurate interference model where interference does not increase at discrete points in time. As a consequence, this introduces an additional problem; the interference does no longer exhibit a simple periodic pattern. Hence, the basic assumption of the fast analysis does not hold for the interference model of the tight analysis. One of the main contributions of this paper is to extend the fast analysis to cope with these traits of the tight analysis, enabling a fast-and-tight analysis.

Another main contribution is that we introduce, for the tight analysis, a method to speed up the numerical convergence during equation solving when calculating response-times. The method is based upon the insight that response-time equations cannot have solutions at arbitrary points in time (which we formally prove). At such points we modify the interference functions in such a way that numerical convergence is accelerated. Since the modifications are done only at times where no response-time solutions exist, they do not affect the final calculated response-time. Hence, the resulting analysis will calculate exactly the same response-times as does the tight analysis. This method is incorporated into the fast-and-tight analysis method.

Our third main contribution is a simulation study where we show that applying above methods to our tight method, the execution times of the resulting fast-and-tight analysis are comparable to those of the fast analysis. That is, we conclude that one does not have to sacrifice analysis speed to achieve accuracy, or vice versa, when using fast-and-tight analysis.

Paper Outline: Sec. 2 revisits our tight offset RTA [5]. In Sec. 3 we present our tight and fast RTA. Sec. 4 presents an evaluation study, followed by conclusions in Sec. 5.

2 Tight offset RTA

This section revisits our existing tight response-time analysis for tasks with offsets [5] and illustrates the intuition behind the analysis and the formulae.

2.1 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 considered mutually independent, i.e., phasing between them are arbitrary. A transaction Γ_i contains $|\Gamma_i|$ number of 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:

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

There are no restrictions placed on offset, deadline or jitter, e.g., they can each be either smaller or greater than the period. In [7] dynamic offsets are introduced, however they are modelled with the static offset and jitter parameters, and therefore the analysis technique presented here also straightforwardly applies to tasks with dynamic offsets. We assume that the load of the system, and each of the transactions, is less than 100%.[1]

Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in Fig. 1. The offset denotes the earliest release time of a task relative to the start of its transaction and jitter (illustrated by the shaded region) denotes the variability in the release of the task. The upward arrows denote earliest possible release of a task and the size of the arrow corresponds to the released tasks execution time.

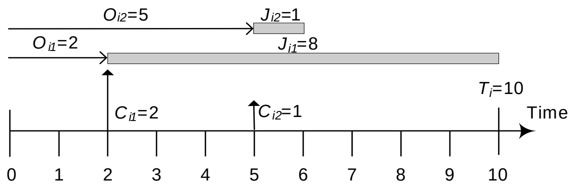


Figure 1. Example transaction

2.2 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 whose 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 [2, 3]. 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 [11] 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 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. Therefore Tindell provided an approximate RTA that still gives good results but uses a single approximation function for each transaction. Palencia Gutierrez *et al.* [7] formalized and generalized Tindell's work.

[1]This can easily be tested, and if not fulfilled, response-times may be infinite; rendering the system unschedulable.

2.3 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 (where $t = 0$ at the critical instant). 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 for 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:

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

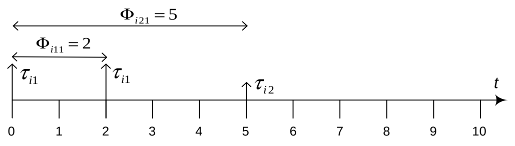
This definition implies that the first instance of a task τ_{ij} in $Set2$ will be released at time $t = \Phi_{ijc}$, and subsequent releases will occur periodically every T_i.

Fig. 2 illustrates the four different Φ_{ijc}-s that are possible for our example transaction of Fig. 1. The upward arrows denote task releases (the height of the corresponding arrow denotes amount of execution released, i.e., C_{i1} or C_{i2} respectively). Fig. 2(a) the case that τ_{i1} coincides with the critical instant, where the phasing to τ_{i1} is 2 and to τ_{i2} is 5. Fig. 2(b) shows the corresponding situation when τ_{i2} is the candidate to coincide with the critical instant.

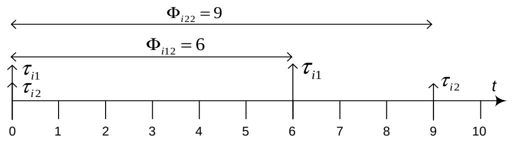
Given the two sets of task instances ($Set1$ and $Set2$) and the corresponding phase relative to the critical instant (Φ_{ijc}), the worst-case interference during a time-interval t caused by task τ_{ij} can be divided into two parts:

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

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



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



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

Figure 2. Φ-s for the two c.i. candidates

These are defined as follows:

$$I^{Set1}_{ijc} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \quad I^{Set2}_{ijc}(t) = \left\lceil \frac{t^*}{T_i} \right\rceil C_{ij} - x$$

$$t^* = t - \Phi_{ijc}$$

$$x = \begin{cases} 0 & t^* \leq 0 \\ 0 & t^* \bmod T_i = 0 \\ 0 & t^* \bmod T_i \geq C_{ij} \\ C_{ij} - (t^* \bmod T_i) & \text{otherwise} \end{cases}$$

Note that, $I^{Set2}_{ijc}(t)$ is redefined compared to [7], resulting in lower (but still safe) response times. For more details and correctness proofs see [5].

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

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I^{Set1}_{ijc} + I^{Set2}_{ijc}(t) \right) \qquad (1)$$

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

2.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination [11, 7]. However, since this is computationally intractable for anything but very small task sets the approximate analysis defines one single, upward approximated, function for the interference caused by transaction Γ_i:

$$W^*_i(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t) \qquad (2)$$

$W^*_i(\tau_{ua}, t)$ simply takes the maximum of each interference function (one for each candidate τ_{ic}). As an example consider again transaction Γ_i depicted in Fig. 1. Fig. 3 shows the interference function for the two candidates (W_{i1} and

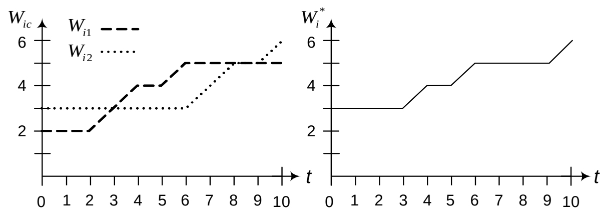


Figure 3. $W_{ic}(\tau_{ua}, t)$ **and** $W_i^*(\tau_{ua}, t)$ **functions**

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 the task under analysis (τ_{ua}), during a time interval of length t, its response time (R_{ua}) can be calculated. The complete response time formulas provided by [7] can also be found in a comprehensive technical report [6].

3 Fast and Tight Analysis

When calculating response times, the function $W_i^*(\tau_{ua}, t)$ in eq. 2 will be evaluated repeatedly. For each task and transaction pair (τ_{ua} and Γ_i) many different time-values, t, will be used during the fix-point calculations. For the traditional response-times analysis for tasks with offsets, a repetitive and periodic pattern for $W_i^*(\tau_{ua}, t)$ can easily be found, and a lot of computational effort is saved by representing the interference function statically, and during response-time calculations using a simple lookup function to obtain its value [4].

However, since the tight analysis has continuously increasing interference functions and does not exhibit a simple periodicity, the framework in [4] is not directly applicable to the tight analysis. This section shows how to find, calculate and store the periodic interference information for the tight RTA method. We also present how the function $W_i^*(\tau_{ua}, t)$ changes using such pre-computed information.

Furthermore, the continuous nature of interference in the tight analysis gives the tight analysis a computational disadvantage compared to the original analysis [7, 11]. In this section we will show how to remove this computational disadvantage by replacing the continuous interference functions with discretely increasing functions without introducing any pessimism in resulting response times.

3.1 The Periodicity of the Interference

The fundamental pre-requisite to statically represent the interference for a transaction, is that a repetitive pattern can be found (such that it suffices to store that pattern and use it to calculate the amount of interference for any time interval t). In our previous fast analysis [4], the full interference of each task within the transaction occurs within the first period (each task is released exactly once during each period). Hence, we could straight-forwardly represent the interference during the first period and reuse it for later periods.

However, in the tight analysis, the imposed interference of a task released towards the end of the period may not be fully included within the period. Even though the task is released within the period, the slanted interference function causes some of the interference to occur in the subsequent period. Fig. 4 shows an example critical instant candidate where the interference from task z *spills* into next period.

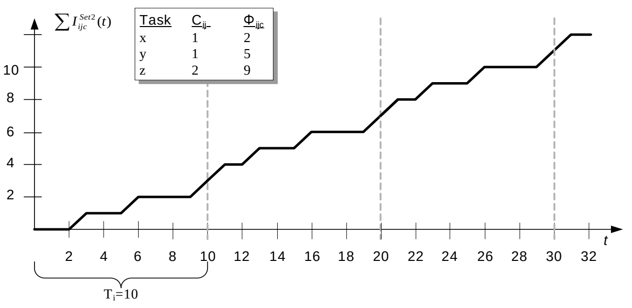


Figure 4. Interference spilling over periods

As seen in Fig. 4, the interference for the first period differs from that of later periods. Obviously, there can be no spill during the first period, since tasks arriving before the critical instant (i.e. when $t < 0$) are accounted for in I_{ijc}^{Set1}. For subsequent periods, however, the effect of a task spilling over period boundaries will be identical. This means that for $t > T_i$ the interference is repetitive (with period = T_i) and allows for a static representation. The consequence of this is that we have to represent the interference for the first and subsequent periods separately.

3.2 Preliminaries

To prepare for subsequent calculations, we define three operations (order, merge, and split) that will be performed for each critical instant candidate before we proceed with calculation of a transactions' interference pattern. These transformations will not change the load or the timing-behavior of the interference, they only help us to restructure the information within a transaction.

Operation: Order Tasks are enumerated according to their first activation after the critical instant, i.e., according to increasing Φ_{ijc} values.

Operation: Merge Each task j' that is released before a previous task j has a chance to finish its execution, i.e. $(\Phi_{ijc} + C_{ij}) \bmod T_i \geq \Phi_{ij'c}$, are merged into one task with execution time $C_{ij} + C_{ij'}$ and offset of Φ_{ijc}. This operation is performed until all possible tasks have been merged (and since the load of a transaction is less than 100% the process is guaranteed to converge).

Operation: Split When splitting a task, we define *spill* of a task j, belonging to transaction Γ_i for the critical instant candidate task c ($c \in \Gamma_i$), denoted S_{ijc}, as the amount of execution time that "spills over" into the next period. Since task j is released at time Φ_{ijc}, the amount of spill is:

$$S_{ijc} = \begin{cases} 0 & \text{if } \Phi_{ijc} + C_{ij} \leq T_i \\ \Phi_{ijc} + C_{ij} - T_i & \text{otherwise} \end{cases}$$

To make the spill explicit, we split each task j with a positive spill into 2 new tasks, denoted j' and j''. j' represents the amount of interference of task j that occurs within and at the end of the current period. j'' is called a *spill task* and represents the amount of interference that occurs at the beginning of the subsequent period. The definitions are:

$$\begin{aligned} C_{ij'} &= C_{ij} - S_{ijc} & C_{ij''} &= S_{ijc} \\ \Phi_{ij'c} &= \Phi_{ijc} & \Phi_{ij''c} &= 0 \end{aligned}$$

3.3 Jitter and time induced interference

The key to make a static representation of $W_i^*(\tau_{ua}, t)$ is to recognisee that it contains two parts:

- A jitter induced part, denoted $J_i^{ind}(\tau_{ua})$. This part corresponds to task instances belonging to $Set1$. Note that this interference is not dependant on t.
- A time induced part, denoted $T_i^{ind}(\tau_{ua}, t)$. This corresponds to task instances of $Set2$. With exception for the first period, the time induced part has a cyclic pattern that repeats itself every T_i (as proved below).

We redefine eq. 2 using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t) \tag{3}$$

This partitioning of $W_i^*(\tau_{ua}, t)$ is visualized in Fig. 5. $J_i^{ind}(\tau_{ua})$ is the maximum starting value of each of the $W_{ic}(\tau_{ua}, t)$ functions (i.e. max of $W_{ic}(\tau_{ua}, 0)$, see eq. 1) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1}$$

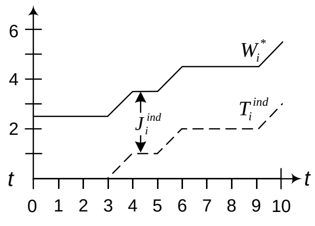


Figure 5. $W_i^*(\tau_{ua}, t)$, $J_i^{ind}(\tau_{ua})$, **and** $T_i^{ind}(\tau_{ua}, t)$

The time induced part, $T_i^{ind}(\tau_{ua}, t)$, represents the maximum interference, during t, from tasks activated after the critical instant. Algebraically $T_i^{ind}(\tau_{ua}, t)$ is defined as:

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^+(\tau_{ua}, t) \tag{4}$$

where

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

The correctness of our method requires that our new definition of $W_i^*(\tau_{ua}, t)$ in eq. 3 is functionally equivalent to the definition in eq. 2.

Theorem 1 *$W_i^*(\tau_{ua}, t)$ as defined in eq. 2 and $W_i^*(\tau_{ua}, t)$ as defined in eq. 3 are equivalent.*

Proof reference By syntactic equivalence to Theorem 1 and corresponding proof in [4].

Further, in order to be able to make a static representation of $W_i^*(\tau_{ua}, t)$, we need to ensure that we store enough information to correctly reproduce $W_i^*(\tau_{ua}, t)$ for arbitrary large values of t. Since $T_i^{ind}(\tau_{ua}, t)$ is the only part of $W_i^*(\tau_{ua}, t)$ that is dependent on t, the following theorem gives that a periodicity of T_i exists in the interference:

Theorem 2 *Assume spill tasks are accounted for, and $t = k * T_i + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_i$), then*

$$T_i^{ind}(\tau_{ua}, t) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t')$$

Proof reference The theorem is proved by algebraic equivalence in [6]

3.4 Representing time induced interference

In this section we show how the interference pattern of $T_i^{ind}(\tau_{ua}, t)$ can be calculated and represented statically. Since the first period should not account for any spill task, but subsequent periods should, we divide the presentation into two cases, one where spill task are not accounted for and one case where they are.

3.4.1 Spill task not accounted for

For each critical instant candidate, τ_{ic}, tasks are ordered, merged, and split according to Sec. 3.2. Spill tasks are removed. We define a set of points p_{ic}, where each point $p_{ic}[k]$ has an x (representing time) and a y (representing interference) coordinate, describing how the time induced interference grows over time when τ_{ic} acts as the critical instant candidate. The points in p_{ic} correspond to the convex corners of $W_{ic}^+(\tau_{ua}, t)$ of eq. 5. The following equations define the array p_{ic}:

$$\begin{aligned} p_{ic}[1].x &= 0 \\ p_{ic}[1].y &= \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} - J_i^{ind}(\tau_{ua}) \\ p_{ic}[k].x &= \Phi_{ikc} + C_{ik} \qquad k \in 2 \ldots |\Gamma_i| \\ p_{ic}[k].y &= p_{ic}[k-1].y + C_{ik} \quad k \in 2 \ldots |\Gamma_i| \end{aligned} \tag{6}$$

$p_{ic}[1].y$ gives the initial relation (i.e. vertical distance at time 0) between different critical instant candidates, and is given by the difference in jitter-induced interference. Furthermore, the time-induced interference should be zero at time zero (illustrated in Fig. 5) which is achieved by subtracting the maximum of all jitter-induced interference (stored in $J_i^{ind}(\tau_{ua})$) when initializing $p_{ic}[1].y$ in eq. 6.

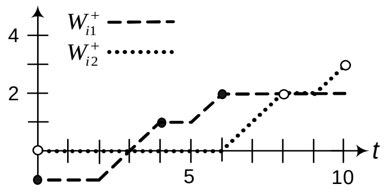


Figure 6. Visual representation of p_{ic} sets

The W_{i1}^+ and W_{i2}^+, for our example transaction, are depicted in Fig. 6 and the corresponding p_{i1} and p_{i2} sets are illustrated by black and white circles respectively. For this example transaction we get the following two p_{ic}-s:

$$p_{i1} = [\langle 0,-1\rangle,\langle 4,1\rangle,\langle 6,2\rangle] \quad \text{black circles}$$
$$p_{i2} = [\langle 0,\ \ 0\rangle,\langle 8,2\rangle,\langle 10,3\rangle] \quad \text{white circles}$$

Now, the information generated by all $W_{ic}^+(\tau_{ua},t)$-functions is stored in the p_{ic}-sets. To obtain the convex corners of $T_i^{ind}(\tau_{ua},t)$, we need to extract the points that represent the maximum of all $W_{ic}^+(\tau_{ua},t)$-s. To this end, we calculate the set of points, p_i, as the union of all p_{ic}-s:

$$p_i = \bigcup_{\tau_{ic}\in\Gamma_i} p_{ic}$$

In order to determine the points in p_i corresponding to the convex corners of $T_i^{ind}(\tau_{ua},t)$, we define a *subsumes* relation: A point $p_i[a]$ subsumes a point $p_i[b]$ (denoted $p_i[a] \succ p_i[b]$) if the presence of $p_i[a]$ implies that $p_i[b]$ is not a convex corner. Fig. 7 illustrates this relation graphically with a shaded region, and the formal definition is:

$$p_i[a] \succ p_i[b] \text{ iff}$$
$$p_i[a].y \geq p_i[b].y \wedge \big(p_i[a].x - p_i[a].y \leq p_i[b].x - p_i[b].y\big)$$

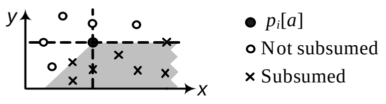


Figure 7. The subsumes relation

Given the subsumes relation, the convex corners are found by removing all subsumed points:

$$\text{From } p_i \text{ remove } p_i[b] \text{ if } \exists a \neq b : p_i[a] \succ p_i[b]$$

For our example transaction of Fig. 1 we have:

$$p_i = [\langle 0,0\rangle,\langle 4,1\rangle,\langle 6,2\rangle,\langle 10,3\rangle]$$

3.4.2 Spill task accounted for

Computing the set of points when accounting for spill tasks, denoted p_i', is analogous to computing p_i, with the following differences:

- Spill tasks from the split operation are not removed. Note that including a spill task might require an additional merge and order operation.
- In Eq. 6 on the preceding page $p_{ic}[1].y$ defines the initial relation (difference in I_{ijc}^{Set1}) between different critical instant candidates. Since p_i' represents the time induced interference, $T_i^{ind}(\tau_{ua},t)$, for $t \geq T_i$, $p_{ic}'[1].y$ should reflect this relation at the end of the first period. The interference for a critical instant c at the end of the first period is represented by $p_{ic}[|\Gamma_i|].y$, consequently we get the following modification to eq. 6:[2]

$$p_{ic}'[1].y = p_{ic}[|\Gamma_i|].y - \max_{x\in\Gamma_i} p_{ix}[|\Gamma_i|].y$$

3.5 Increasing performance by removing slants

Assume that a set of points p_i (with or without spill tasks) has been calculated, representing the convex corners of the time induced interference function $T_i^{ind}(\tau_{ua},t)$ during one period T_i. The points for our example transaction is illustrated in Fig. 8. Note that in the absence of spill tasks, the sets p_i and p_i' are identical.

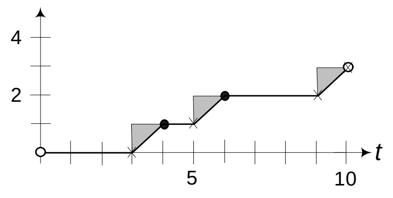


Figure 8. Removing the slants

It can be proven that the fix-point iterative solution to Eq. 28' in Appendix A, which is the equation where the interference function is used, cannot have any solution during the slants.

Theorem 3 *Equation 28' cannot have a solution at a time t where any approximate interference function has a derivative greater than or equal to one.*

Proof reference The theorem is proved in Appendix A.

No solutions to the response-time equation can exist during the slant of any interference function. Furthermore, the closest possible solution will be when the derivative of the interference function becomes zero. Hence, we can remove the slants and replace them with a stepped stair function,

[2] Analogous to eq. 6, we normalize the points to start at 0, hence we subtract the maximum of all $p_{ix}[|\Gamma_i|].y$.

as illustrated by the grey areas of Fig. 8, without introducing any pessimism in the resulting response times. However, progress in the fix-point iteration is proportionally increased with any overestimation of the interference. Hence, by adding overestimation in the grey areas of Fig. 8 we will speed up the fix-point convergence without modifying the calculated response-times.

We will remove the slants by transforming the convex corners to concave corners (illustrated by crosses in Fig. 8[3]). The rules for finding the concave corners, v_i, from a set of convex corners, p_i, is as follows:

$$
\begin{aligned}
&v_i[k].y = p_i[1].y \\
&v_i[k].x = \begin{cases} p_i[k+1].x - \textit{diff} & \text{if } k < |p_i| \\ p_i[k].x & \text{if } k = |p_i| \end{cases} \\
&where \quad k \in 1 \ldots |p_i|, \quad \textit{diff} = p_i[k+1].y - p_i[k].y
\end{aligned}
$$

The interpretation of v_i is as follows: For $t \leq T_i$, $v_i[k].y$ represents the maximum amount of time induced interference Γ_i will impose on a lower priority task during interval lengths up to $v_i[k].x$ ($k \in 1 \ldots |v_i|$). For our example transaction of Fig. 1, v_i becomes (indicated by crosses in Fig. 8 on the previous page):

$$v_i = [\langle 3,0\rangle, \langle 5,1\rangle, \langle 9,2\rangle, \langle 10,3\rangle]$$

Note, especially that the final point (denoted $v_i[|v_i|]$) contains the sum of all interference during the period T_i.

In the special case that some task τ_{ij} has $\Phi_{ijc} = 0$ (e.g. in the case for spill tasks), $v_i[1].x$ will not be zero. However, since $T_i^{ind}(0) = 0$ (follows from eq. 4), the first element of v_i needs to have x-value that is zero. In such cases we add the point $\langle 0,0\rangle$ to v_i (stating that there will be 0 time induced interference for any time interval of length up to 0).

Discussion: Removal of slants

By removing the slants, we essentially revert to the stepped-stair interference functions used in the original analysis [7, 11]. This could seem surprising, since the tight analysis is based on the insight that stepped-stair interference functions are overly pessimistic. However, as theorem 3 states, there could be no response-time solutions during a slant. Hence, using slants *during fix-point equation solving* does not increase the precision of the analysis.[4]

However, when *deriving* the interference function it is imperative to use a faithful model (using slants) for the different sources of interference. Hence, once we have derived the interference function (as done when creating the point set p_i), we no longer need to represent the slants and can revert to a stepped-stair interference function.

An analogy could be made to calculations using floating-point values. If rounding values up before each calculation step, the resulting error will be greater than if the calculation is done using floating-point values, and only the final result is rounded up.

3.6 $T_i^{ind}(\tau_{ua}, t)$ using lookup

Since we need to represent the interference for the two first periods separately we will calculate the two point sets p_i (first period) and p'_i (second period) according to Sec. 3.4. Next we will remove the slants for both these point sets as described in Sec. 3.5 and store the new points in v_i and v'_i respectively.

Using the point sets v_i and v'_i we can calculate the interference from Γ_i for an arbitrary time t. For the first period the interference in v_i is used, and when $t > T_i$ we will start using the interference in v'_i. Using these point sets $T_i^{ind}(\tau_{ua}, t)$ can be reduced to a fast lookup function:

$$
\begin{aligned}
&T_i^{ind}(\tau_{ua}, t) = \begin{cases} v[n].y \text{ if } k < 1 \\ V \text{ if } k \geq 1 \end{cases} \\
&V = v_i[|v_i|].y + (k-1) * v'_i[|v'_i|].y + v'_i[n'].y \\
&k = t \text{ div } T_i \\
&t^* = t \text{ rem } T_i \\
&n = \min\{m \;:\; t^* \leq v_i[m].x\} \\
&n' = \min\{m \;:\; t^* \leq v'_i[m].x\}
\end{aligned}
\tag{7}
$$

where k represents the number of whole periods (T_i) in t, and t^* is the part of t that extends into the final period. It could be noted that $v_i[|v_i|].y$ contains the sum of all interference during the first period, and $v'_i[|v'_i|].y$ contains the sum of all interference during the length of one period for subsequent periods.

3.7 Space and Time Complexity

The number of points to calculate (p_i) is quadratic with respect to the number of tasks in the transaction Γ_i ($2|\Gamma_i|$ points for each of the $|\Gamma_i|$ candidate tasks). Thus, storing v_i and v'_i results in a quadratic space complexity since, theoretically, no points from the p_{ic} sets will be removed when calculating p_i.

The method presented in this paper divides the calculation of W_i^* into a pre-calculation and a fix-point iteration phase. A naive implementation of the removal procedure in eq. 7 requires comparison of each pair of points; resulting in cubic time-complexity ($O(|\Gamma_i|^3)$) for pre-calculating v_i and v'_i.[5] During the fix-point iteration phase, a binary

[3]While the last point in the set does not strictly represent a concave corner, it is still necessary for us to keep track of the amount of interference at the end of the period, hence that point will be included among the concave corners and is thus marked with a cross in the figure.

[4]This is why the original response-time analysis [2] and exact analysis for tasks with offsets [7, 11] does not overestimate response times.

[5]In Sec. 4 we use an $O(|\Gamma_i|^2 log|\Gamma_i|)$ implementation based on sorting the points and making a single pass through the sorted set.

search through a quadratically sized array is performed (either v_i or v'_i in eq. 7), resulting in $O(\log |\Gamma_i|^2)$ time complexity for calculating W^*_i according to eq. 3. The original complexity for calculating W^*_i according to eq. 2 is $O(|\Gamma_i|^2)$.

In a complete comparison of complexity, the calculation of $W^*_i(\tau_{ua}, t)$ must be placed in its proper context (see the response time formulas in [6, 7]). Assume X denotes number of fix-point iterations needed, then the overall complexity for the original approach (eq. 2) is $(O(X|\Gamma_i|^2))$, whereas our method (eq. 3 & eq. 7) yields $(O(|\Gamma_i|^3 + X \log |\Gamma_i|^2))$. Typically the size of a transaction ($|\Gamma_i|$) is small (less than 100) and the number of fix-point iterations (X) is large (tens or hundreds of thousands), hence our method results in a significant reduction in complexity.

4 Evaluation

In order to evaluate and quantify the efficiency (with respect to execution time of RTA) of our proposed method, we have implemented a set of approximate response-time techniques, using the complete set of response-times equations in [6, 7]. We use these implementations to perform an extensive simulation study. We compare five RTA methods:

- *fast-tight*, presented in this paper and is the method that is optimized the farthest with respect to both analysis speed and tightness. The goal of this simulation study is to quantify its efficiency with respect to execution time of the analysis.
- *fast-slanted*, presented in this paper but without removing the slants (see Sec. 3.5). The reason for including it in the analysis is to investigate the impact of reverting back to a stepped stair interference function during response time calculations.
- *tight*, presented in Sec. 2 and [5]. It is only optimized towards tightness. These three methods all produce the exact same tight response times.
- *orig*, presented by Palencia Gutierrez *et al.* [7], which is not optimized either for tightness nor for analysis speed. It is included in the evaluation to see if the relative performance degradation of *tight*, compared to *orig*, remains in *fast-tight* when compared to *fast-orig*.
- *fast-orig*, our speed-up method of *orig* presented in [4]. It is the fastest known RTA for tasks with offsets. It yields the same response times as *orig*. It is included to see if the performance gain of *fast-tight* is comparable to those of *fast-orig*

4.1 Description of Simulation Setup

In our simulator, we generate task sets that are used as input to the different RTA implementations. The generated task-sets have the following characteristics:

- Total system load is 90%.
- The number of transactions is 10.
- Jitter (J_{ij}) for each task is 20% of its transaction period.
- Blocking (B_{ij}) is zero.
- The number of tasks/transaction is a variable parameter.
- The priorities are assigned in rate monotonic order.
- Transaction periods (T_i) are randomly distributed in the range 1,000 to 1,000,000 time units (uniform distr.).
- 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. The fraction is selected so that the transaction load of 9% is obtained.

The execution time for performing the RTA in Sec. 4.2 have been obtained by taking the mean value from 50 generated task-sets for each point in each graph. We have measured the execution time on a Pentium 4 laptop. The execution times are plotted with 95% confidence interval for the mean values. Note that, for *fast-orig*, *fast-slanted*, and *fast-tight* the execution times also include the time to perform the pre-calculations presented in Sects. 3.4 and 3.5.

4.2 Simulation Results

Fig. 9(a) shows how the execution time of the five (although the 3 fast methods are indistinguishable) RTA analysis varies with varying tasks/transaction (all methods are listed in decreasing execution time order). When the number of tasks/transaction is 20, *tight* takes about 86 seconds whereas *fast-tight* takes around 0.63 seconds, which is a speed up of well over two orders of magnitude. Note also that, *tight* has a slight penalty to pay, compared to *orig*, due to more accurate interference modelling.

Zooming in on the three fast analysis methods in Fig. 9(b), we see that *fast-tight* and *fast-orig* are quite comparable in execution times. There are two, mutually opposing, factors that affect their relative timing: The *fast-tight* method shortens its execution time since it sometimes calculates lower response-times than the *fast-orig* method (and hence terminate in fewer fix-point iterations). On the other hand the *fast-tight* method has to spend more time performing pre-calculations and also perform lookup in two different arrays during each fix-point iteration. In Fig. 9(b) we see that *fast-tight* has consistently slightly longer execution time.

In Fig. 9(b) we also see that *fast-slanted* pays a price of slower fix-point convergence due to the slanted interference function as did *tight* over *orig*. We conclude from Fig. 9(a) and 9(b) that the main contribution of speeding up the response times comes from static representation and lookup,

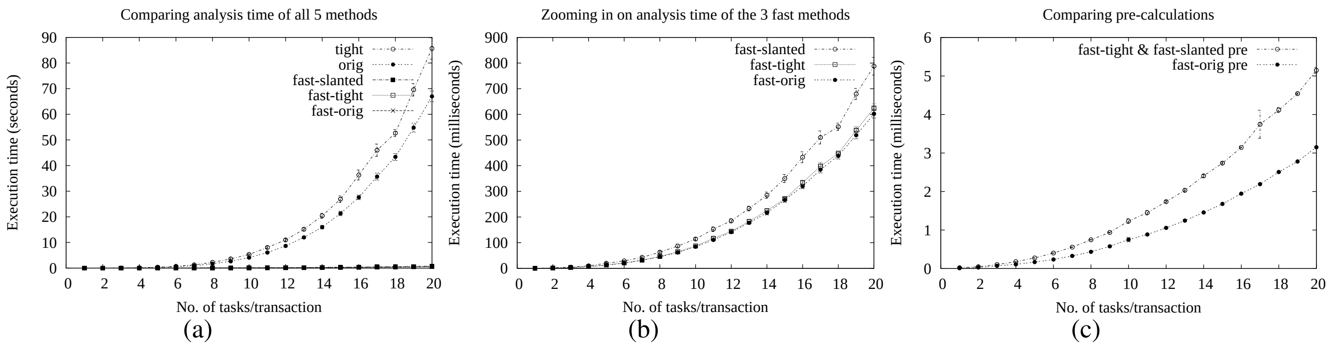


Figure 9. Simulation Results

but that reverting back to a stepped stair function gives an additional speedup of over 20%.

In Fig. 9(c) we compare the pre-calculations of the three fast methods. Here we can see that the pre-calculations of *fast-tight* and *fast-slanted* is approximately twice that of *fast-orig*. This is expected since they calculate two sets of arrays as opposed to a single set in *fast-orig*. Comparing with Fig. 9(b) one can see that the pre-calculations constitute less than 1% of the total analysis time. One can also discern the complexity of the pre-calculations, and the slope is less steep than what would be expected of a naive implementation with worst-case complexity of $O(|\Gamma_i|^3)$, this is partly due to our (sorting based) $O(|\Gamma_i|^2 log|\Gamma_i|)$ implementation of the pre-calculations, and partly because the worst (theoretical) case, with $|\Gamma_i|^2$ elements in the pre-calculated arrays, never occurs.

We have also simulated an admission control situation. In an admission control situation, a single (low priority) task is added to an (otherwise schedulable) set of already admitted tasks, and its response-time is calculated and compared with its deadline (to decide if the task can be admitted to the system or not). In the admission control the pre-calculation of the already admitted tasks is not included in the execution time. In these simulations, for 20 tasks/transaction, the *tight* method takes about 92 milliseconds whereas the *fast-tight* takes 0.19 milliseconds, which is a speedup with a factor of almost 500. When performing admission control, the speed up in our method is isolated due to two factors: (1) pre-calculations are already done, and (2) no interference from other tasks in the same transaction needs to be accounted for. As can be seen in appendix A, the exact interference-function is used to account for interference from tasks in the same transaction. Since *fast-tight* only improves the approximate interference-function, we isolate our improvement by not needing to account for interference from tasks in the same transaction.

This evaluation shows that combining fast and tight methods for response time analysis, one gets the best of two worlds; a response time analysis method that is both fast and tight, outperforming previous methods by several orders of magnitude.

5 Conclusions

In this paper we have presented a novel method that calculates approximate worst-case response times for tasks with offsets. Distinguishing feature of the method is that it calculates tight response times in a short analysis time. We have successfully extended our framework of fast RTA [4] to be able to apply it to our tight method [5]. Our improvements are orthogonal and complementary to other proposed extensions to the original offset analysis such as [8, 9].

The main effort in performing RTA for tasks with offsets is to calculate how higher priority tasks interfere with a task under analysis. The essence to calculate fast response times is to find a repetitive pattern and store that pattern statically, and during response time calculations (fix-point iteration), use a simple table lookup. Our tight analysis [5] exploits the fact that the interference imposed by higher priority tasks is overestimated in traditional RTA. By removing this overestimation, significantly tighter response-times can be calculated. The fast-and-tight analysis presented in this paper successfully does both, resulting in a fast and tight RTA.

Faster RTA has several positive practical implications: (1) Engineering tools (such as those for task allocation and priority assignment) can feasibly rely on RTA and use the task model with offsets, and (2) on-line scheduling algorithms, e.g., those performing admission control, can use accurate on-line schedulability tests based on RTA. Tighter RTA has the practical implications to allow more efficient hardware utilization. Either more functions can be fitted into the same amount of hardware, or less powerful (cheaper) hardware can be used for the existing functions. Hence, our fast-and-tight analysis is a very attractive choice to include in engineering tools and/or admission control software for resource constrained embedded real-time systems.

In a simulation study we see that our novel analysis has very similar computational requirements to that of the fast analysis. Especially we notice that the computational disadvantage of the tight analysis (compared to the original analysis) is completely removed when comparing the fast-and-tight with the fast analysis. Example benchmarks include a speedup of over 100 times for response-time analysis of entire task-sets and a speedup of almost 500 times for single tasks, e.g., corresponding to an admission control situation.

References

[1] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed Priority Pre-Emptive Scheduling: An Historical Perspective. *Real-Time Systems*, 8(2/3):173–198, 1995.

[2] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *The Computer Journal*, 29(5):390–395, 1986.

[3] C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[4] J. Mäki-Turja and M. Nolin. Faster Response Time Analysis of Tasks With Offsets. In *Proc. 10th IEEE Real-Time Technology and Applications Symposium (RTAS)*, May 2004.

[5] J. Mäki-Turja and M. Nolin. Tighter Response-Times for Tasks with Offsets. In *Proc. of the 10th International conference on Real-Time Computing Systems and Applications (RTCSA'04)*, August 2004.

[6] J. Mäki-Turja and M. Nolin. Fast and Tight Response-Times for Tasks with Offsets – Extended version. Technical Report MRTC no. 173, Mälardalen Real-Time Research Centre (MRTC), March 2005.

[7] J. Palencia Gutierrez and M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998.

[8] J. Palencia Gutierrez and M. Gonzalez Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real-Time Systems. In *Proc. 20th IEEE Real-Time Systems Symposium (RTSS)*, pages 328–339, December 1999.

[9] O. Redell. 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.

[10] M. Sjödin and H. Hansson. Improved Response-Time Calculations. In *Proc. 19th IEEE Real-Time Systems Symposium (RTSS)*, December 1998. URL: http://www.docs.uu.-se/~mic/papers.html.

[11] K. Tindell. 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.

Appendix A Proof of Theorem 3

In proving theorem 3, we will use eq. 28 in [7], definition of $w_{uac}(p)$, the worst case response time of τ_{ua} with τ_{uc} as the one coinciding with the critical instant, simplified and rewritten as a function of time, $f(t)$:

$$f(t) = K_1 + W_{uc}(\tau_{ua}, t) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, t) \qquad (28')$$

where K_1 is some constant value. We note that a solution to eq. 28' exists, and fix-point convergence is reached, when $f(t) = t$, for some t. Since both exact (W_{uc}) and approximate (W_i^*) interference functions are monotonically increasing, we conclude that $f(t)$ is also monotonically increasing.

Lemma 1 *The smallest solution to eq. 28', denoted s, cannot exist where $f(t)$ has a derivative greater than or equal to 1 (i.e. where $f'(t) \geq 1$).*

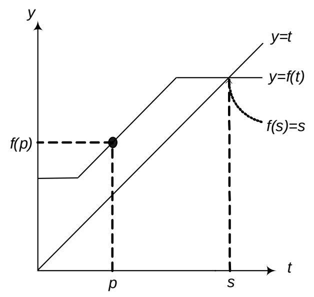


Figure 10. Fix-Point Iteration when $f'(t) \geq 1$

Proof of Lemma 1 *From [10] we know that:*

1. *For any monotonically increasing response-time equation, for any $p < s$, $f(p) > p$ holds.*
2. *We can start fix-point iteration from any point $p < s$ and still find the smallest fix-point s.*
3. *At a point $p < s$ where $f'(p) \geq 1$, consider Fig. 10, the line $y = f(p)$ cannot be converging with line $y = p$ (which has a derivative of 1).*

Assume that s is a point where $f'(t) \geq 1$ then (by the continuousness of $f(t)$) there exists a point $p = s - \epsilon$ (for some small ϵ) where $f'(p) \geq 1$. Then by 1 $f(p) > p$, and by 3 the lines will not be converging. However, by 2 it should be possible to start fix-point iteration at p and converge into s.

A contradiction has been reached and the assumption does not hold. Hence the lemma holds. □

Theorem 3 *Equation 28' cannot have a solution at a time t where any approximate interference function has a derivative greater than or equal to one.*

Proof of Theorem 1 None of the terms in $f(t)$ has a negative derivative. Hence, if for time t any of the approximate interference functions $W_i^*(\tau_{ua}, t)$ has a derivative of one[6], then the function $f(t)$ has a derivative greater than or equal to one. Then, by lemma 1, the theorem holds. □

[6]The derivative of an approximation function $W_i^*(\tau_{ua}, t)$ is either one (for a slant) or zero (for a stair).