Analysing Response-Times for Tasks with Offsets Tight Resluts at Fast Speed

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

MRTC report no. 161, May 2004

Abstract

We present an novel method to calculate approximate upper bounds on the worst-case response-times for tasks which may have offset relations amongst them. The response-times calculated are tighter (i.e. lower) than previous methods and the calculation time is significantly shorter. We have previously obtained both of these benefits in isolation, and in this paper the two methods are combined to form a tight *and* fast response-time analysis for tasks with offsets.

In a comprehensive simulation study we explore both the tightness and analysis speed of; exact analysis for tasks with offsets, traditional approximate analysis, our previous fast analysis, our previous tight analysis, and the novel combined tight and fast analysis. The results show that adding tightness to our fast analysis has a insignificant (albeit negative) impact on the analysis time. Since the tight and fast analysis always calculates the same response-times as those of the tight analysis, our tight and fast method is the preferred method to use in any situation high-quality response-time estimates are needed within reasonable analysis time.

1 Introduction

A powerful and well established schedulability analysis technique is the *Response-Time Analysis* (RTA) [1]. RTA is applicable to, e.g., systems where tasks are scheduled in priority order which is the predominant scheduling technique used in real-time operating systems today. In this paper, we present a novel method that enables a fast implementation of a tight approximation of the worst-case response-time for tasks which may have offset relations amongst them.

Traditionally, industrial use of RTA has been limited. However, with recent advancements in software development and synthesis tools, such as UML-based tools [2, 10, 13], e.g. schedulability tests using RTA, can be integrated in the normal workflow and tool-chains used by real-time engineers. This kind of tools can be used, for instance, to perform automatic task allocation in a distributed real-time system or to automatically derive task priorities (priority assignment) so that task deadlines are guaranteed to be met. Typically, such automatic allocation/assignment methods are based on optimisation or search techniques, during which numerous possible configurations are evaluated. (There can easily be tens or hundreds of thousands of possible configurations even for small systems.) For each configuration a schedulability test is performed in order to evaluate different solutions. Hence, schedulability tests must be fast in order to be suitable in practice for such systems. A system with even greater emphasis on analysis speed is dynamic real-time systems with on-line admission control of real-time tasks. These systems needs to be able to quickly evaluate whether a dynamically arriving task can be admitted to the system or not.

Most real-time systems are also embedded control-systems. Typically an embedded system is also a *resource constrained* system. This means that resources are scares and cannot easily be wasted. In this kind of systems its important that the analysis is *tight*, i.e., does not unnecessarily overestimate the resources that are needed.

The first RTA for tasks with offsets was presented by Tindell [14]. 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 also provided an approximation algorithm, with polynomial complexity, which gives pessimistic but safe results (i.e. the worst case response times are never underestimated) [14].

In this paper we focus on the approximate analysis presented by Palencia Gutierrez *et al.* [8] that formalised and generalaised the work of Tindell. Several researchers have further extended the approximative analysis [9, 11]. These methods focus on precedence-relations among tasks, and are orthogonal and complementary to the methods presented in this paper.

We will combine our two earlier, complementary, methods to achieve tight analysis results and fast analysis time. We have previously presented how the original offset analysis, by Tindell [14] and Palencia Gutierrez *et al.* [8], can be speed up by two orders of a magnitude for tasks sets of non-trivial size [6]. We have also shown how the original analysis can be tightened to calculate lower (but still safe) approximations of the worst case response-times [5, 7]. Both these improvements have been formally proven correct, safe, and never to calculate higher response-times than the original analysis. In this paper we show how these two, independent, improvements can be combined to form a tight and fast response-time analysis.

In a comprehensive simulation we study a large set of analysis techniques; the exact analysis for tasks with offsets, traditional approximate analysis, our previous fast analysis, our previous tight analysis, and the novel tight and fast analysis. We study both the calculated response-times and the analysis execution-time for each technique. This is done for a set of different simulation scenarios. Since the tight and fast analysis calculates the exact same response-times as those of the tight analysis (which can have a negative effect on analysis time), our tight and fast method is the preferred method to use in any situation high-quality response-time estimates are needed within reasonable analysis time.

Paper Outline: In section 2 we revisit and restate the original offset RTA [14, 8]. In section 3 we modify this RTA to calculate tighter response-times, and in section 4 we show how our fast analysis method works. Section 5 presents the combined tight and fast analysis and section 6 presents the evaluation study. Finally, in section 7 we conclude the paper.

2 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets [8, 14] and illustrates the intuition behind the analysis and the formulas.

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 interarrival time between two consecutive events) The activating events can be mutually independent, i.e., phasing between them is considered 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. We assume that the load of the system is less than 100%. (This can easily be tested, and if not the case some response-times may be infinite; rendering the system unschedulable.)

Parameters for an example transaction (Γ_i) with two tasks (τ_{ia}, τ_{ib}) is depicted in figure 1. The offset denotes the earliest release time of a task relative to the start of its transaction and jitter denotes the variability in the release of the task. (In figure 1 the jitter is not graphically visualised.)

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

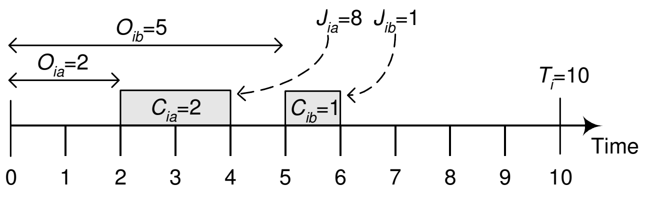


Figure 1: Example transaction 1

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, 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 [14] 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.* [8] 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.

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. 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.

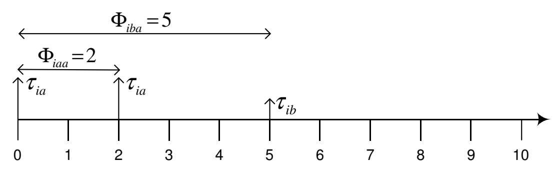
2. The task of Γ_i that coincide with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

The phasing between a task, τ_{ij}, and a critical instant candidate, τ_{ic}, becomes (slightly reformulated compared to [8], see Appendix A):

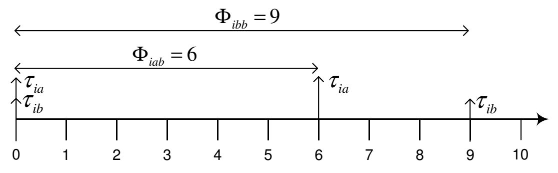
$$\Phi_{ijc} = (O_{ij} - (O_{ic} + J_{ic})) \bmod T_i \tag{1}$$

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 offset relations 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 2 illustrates the four different Φ_{ijc}-s that are possible for our example transaction of figure 1. The upward arrows denote task releases (the height of the corresponding arrow denotes amount of execution released, i.e., C_{ia} or C_{ib} respectively). Figure 2(a) shows the phasing between τ_{ia} (2) and τ_{ib} (5) when τ_{ia} acts as the candidate critical instant. One can see, for every task in the transaction, when the first invocation in $Set2$ is released, in the case that τ_{ia} coincides with the critical instant. Figure 2(b) shows the corresponding situation when τ_{ib} is the candidate to coincide with the critical instant.



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



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

Figure 2: 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}.

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:

$$I_{ijc}^{Set1} = \left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor C_{ij} \qquad I_{ijc}^{Set2}(t) = \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil C_{ij} \tag{2}$$

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

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) \tag{3}$$

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 [8, 14]. 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 [8, 14]:

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

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

As an example consider again transaction Γ_i depicted in figure 1. Figure 3 shows the interference function for the two candidates (W_{ia} and W_{ib}), 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. Appendix A shows how to perform these response-time calculations.

3 Tight Analysis

In our previous work we have shown how to decrease the pessimism of the analysis in section 2 [5, 7]. In this section we shortly present this *tight analysis*, and we begin with an illustrative example of how the original analysis overestimates the response-time. Consider a simple transaction Γ_i depicted in figure 4 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 $C_{ua} = 2$ and $D_{ua} = T_u = 12$. For this simple task set where $B_{ij} = J_{ij} = 0$, $D_{ua} \leq T_u$ only one instance of τ_{ua} is active at any point in time. This means that the response time formulas (presented in appendix A), for the low priority task, can be reduced and simplified to:

$$R_{ua} = C_{ua} + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, R_{ua}) \tag{5}$$

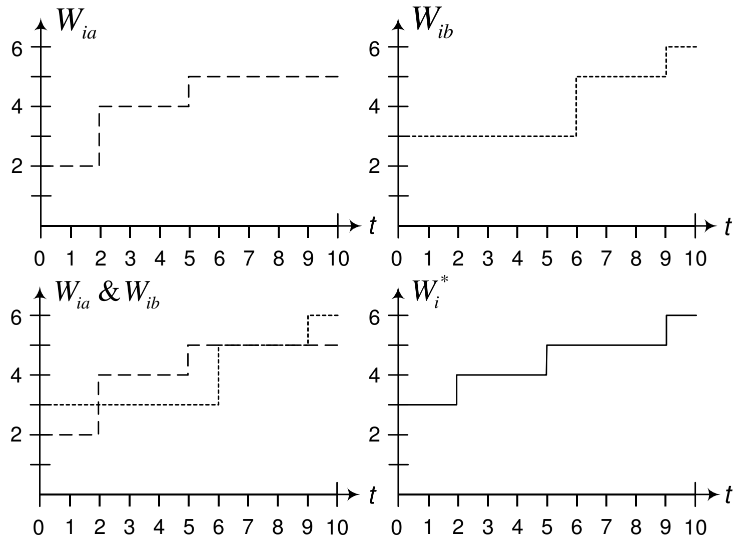


Figure 3: $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions for example 1

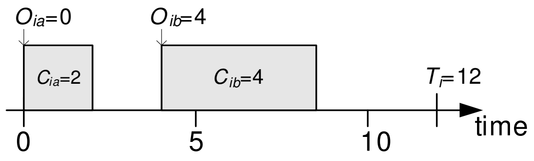


Figure 4: Example transaction 2

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

Iter#	t	W_{ia}	W_{ib}	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_{ia}" and "W_{ia}" denotes $W_{ic}(\tau_{ua}, t)$ for the two candidate tasks τ_{ia} and τ_{ib} respectively. "W_i^*" 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, 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_{ia} and τ_{ib} since both tasks are separated by at least 2 units of idle time. Hence the actual worst case response-time is $R_{ua} = 6$.

3.1 Using "Imposed" Interference

One property of the ceiling expression of $I_{ijc}^{Set2}(t)$ in equation 2 on page 6 is that it returns the amount of interference "released for execution" during t, resulting in a *stepped stair* interference function (as shown in e.g. figure 3). In [5, 7] we show that this is overly pessimistic and for the approximate offset analysis yields unnecessary pessimistic response-times. Instead, we proposed the concept on "imposed" interference, which more accurately captures the interference a lower priority task actually experiences during a time interval t.

When we modify $I_{ijc}^{Set2}(t)$ in equation 2 so that it returns interference "imposed" on τ_{ua} we get a *slanted stair* function. The two slanted stair functions for our second example transaction (figure 4 on the preceding page) are shown in figures 5(a) and 5(b).

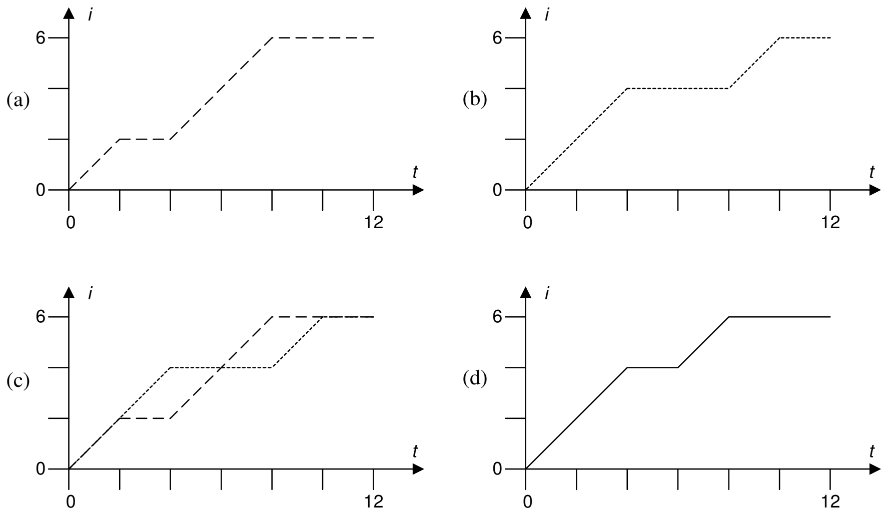


Figure 5: Interference *imposed* by our example transaction

The slanted stairs are obtained by modifying $I_{ijc}^{Set2}(t)$ defined in equation 2 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:

$$I_{ijc}^{Set2}(t) = \left\lceil \frac{t^*}{T_i} \right\rceil C_{ij} - x \qquad t^* = t - \Phi_{ijc} \qquad x = \begin{cases} 0 & \text{if } t^* < 0 \\ C_{ij} - (t^* \bmod T_i) & \text{if } 0 < t^* \bmod T_i < C_{ij} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where Φ_{ijc} is defined in equation 1 on page 5 and x is used to generate the slants of the "imposed" interference function.

For example 2 in figure 4 on the preceding page the slanted stairs generated by equation 6 are shown in figures 5(a) and 5(b), and figure 5(c) shows them overlayed. Using our new version of $I_{ijc}^{Set2}(t)$ in equation 3 on page 6 we get the maximated slanted stairs interference function, representing the approximation function W_i^*, shown in figure 5(d).

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

Iter#	t	W_{ia}	W_{ib}	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 $I_{ijc}^{Set2}(t)$ make the analysis able to "see" the empty slot between tasks τ_{ia} and τ_{ib} something the original analysis overlooked. The calculated response-time is 6, which is lower than that of the original analysis (which was 8).

4 Fast Analysis

In this section we present our method to speed up the analysis of section 2; our *fast analysis* [6]. The method is based on the insight that the function $W_i^*(\tau_{ua}, t)$ (equation 4 on page 6) has a pattern that is repeated every T_i time units (which is proved in [6]). A lot of computational effort is saved by representing the interference function statically, and during response-time calculation use a simple lookup function to obtain its value.

4.1 Approximation function with lookup

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

- A jitter induced part, denoted $J_i^{ind}(\tau_{ua})$. This part corresponds to the task instances belonging to $Set1$. Note that the amount of interference of these instances does not depend on t.
- A time induced part, denoted $T_i^{ind}(\tau_{ua}, t)$. This corresponds to task instances in $Set2$. The time induced part has a cyclic pattern that repeats itself every T_i units of time (see [6]).

We redefine equation 4 using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t) \tag{7}$$

For our example in figure 1 on page 4, this partitioning of $W_i^*(\tau_{ua}, t)$ is visualised in figure 6. $J_i^{ind}(\tau_{ua})$ 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 equation 3 on page 6) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} \tag{8}$$

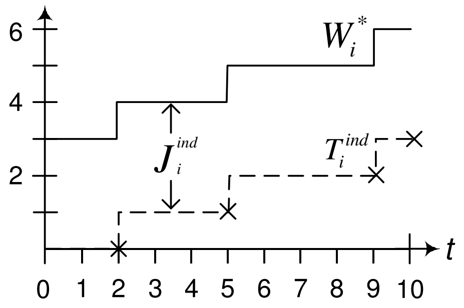


Figure 6: Relation between $W_i^*(\tau_{ua}, t)$, $J_i^{ind}(\tau_{ua})$, and $T_i^{ind}(\tau_{ua}, t)$

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:

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua}, t) \tag{9}$$

where

$$W_{ic}^{+}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) - J_i^{ind}(\tau_{ua}) \tag{10}$$

We represent $T_i^{ind}(\tau_{ua}, t)$ for the first T_i time units using the concave corners of the function $T_i^{ind}(\tau_{ua}, t)$ (marked with crosses in figure 6). The representation uses two arrays T_i^c and T_i^t, where $T_i^c[x]$ represents the maximum amount of time induced interference Γ_i will cause a lower priority task during interval lengths up to $T_i^t[x]$ $(x \in 1 \ldots |T_i^c|)$. Since the interference-pattern is periodic it is enough to represent interference for one period (from time 0 to time T_i).

$T_i^{ind}(\tau_{ua}, t)$ can now use T_i^c and T_i^t and perform a simple lookup to obtain the time induced interfere during t. Using this new definition of $T_i^{ind}(\tau_{ua}, t)$ together with equations 7 and 8 instead of equation 4 to compute $W_i^*(\tau_{ua}, t)$, will reduce the time to compute response times by an order (or two) of a magnitude [6]. In this paper we do not go into details of how to compute and use the arrays T_i^c and T_i^t for the *fast analysis*, the interested reader is referred to [6].

5 Tight and Fast Analysis

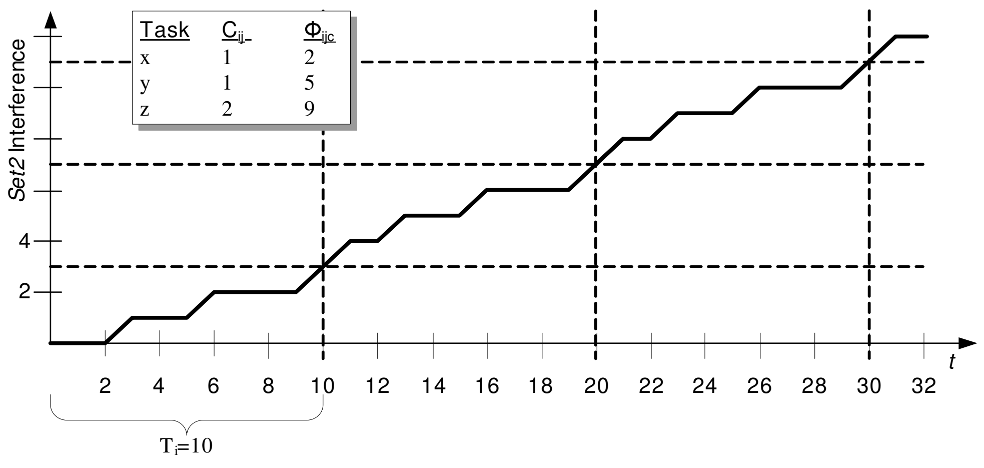
In this section we present our novel *tight and fast* analysis for tasks with offsets. The method will calculate the same values as our tight analysis and it will do so using approximately the same computational effort as our fast analysis. In essence, we will apply a similar techniques to the tight analysis as we did to the original analysis in order to speed up the calculations. However, as we will show, there are some new considerations when making the tight analysis fast.

We will reuse the framework of the fast analysis is section 4. Since the tight analysis does not affect the $Set1$ task instances, equations 7 and 8 on the page before will be the base of also the tight and fast analysis. Hence, in this section we will show how to make an efficient implementation of $T_i^{ind}(\tau_{ua}, t)$ and how to calculate and use the arrays T_i^c and T_i^t to represent the tight analysis.

5.1 The Periodicity of the Interference

The fundamental pre-requisite to statically represent the interference for a transaction, is that a repetetive pattern can be found (such that it suffices to store that pattern and use it calculate the amount of interference for any time interval t). In our previous fast analysis 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 makes some of the interference to occur in the subsequent period. Figure 7 shows an example critical instant candidate where the interference from task z *spills* over between periods.



Task	C_{ij}	Φ_{ijc}
x	1	2
y	1	5
z	2	9

Figure 7: Interference spilling into the next period

As seen in figure 7, the interference for the first period $I_{ijc}^{Set2}(t)$ 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 repetetive (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.

5.2 Preliminaries

To prepare for subsequent calculations, we define three operations (merge, split, and order) 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-behaviour of the interference, they only help us structure the information within a transaction. The operations will be performed in the order given.

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

Operation: Split We define the *spill* of a task j, belonging to transaction 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:

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

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

5.3 Precomputing T_i^c and T_i^t

We will start out by calculating the interference for the first period (for $t \leq T_i$) and store that information in T_i^c and T_i^t, where $T_i^c[x]$ represents the maximum amount of time induced interference Γ_i will cause a lower priority task during interval lengths up to $T_i^t[x]$ ($x \in 1 \ldots |T_i^c|$). This method is essentially the same as for the fast analsys [6], but adapted for "imposed" interference.

To compute T_i^c and T_i^t we will first calculate the interference pattern for each critical instant candidate. That is we calculate the pattern generated by $W_{ic}^{+}(\tau_{ua}, t)$ (equation 10 on page 10) while using the $I_{ijc}^{Set2}(t)$ of the tight analysis (equation 6 on page 8).

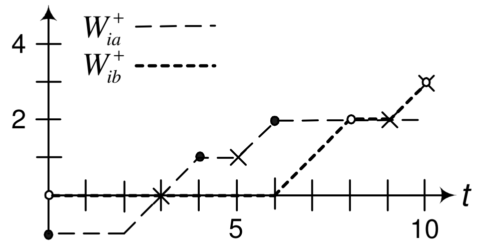


Figure 8: Visual representation of p_{ic} sets

For each critical instant candidate, τ_{ic}, tasks are merged, splitted, spill tasks removed (remeber that spill tasks does not affect the first period), and ordered according to section 5.2. We define a set of points p_{ic}, where each point $p_{ic}[k]$ has an x and a y coordinate, describing how the time induced interference grows over time when τ_{ic} is the critical instant candidate. The points in p_{ic} corresponds to the convex corners of $W_{ic}^{+}(\tau_{ua}, t)$ of equation 10 with $I_{ijc}^{Set2}(t)$ definition of equation 6. The following equation define the array p_{ic}:

$$\begin{aligned} p_{ic}[1].x &= 0 \\ p_{ic}[1].y &= \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} - J_i^{ind}(\tau_{ua}) \end{aligned} \qquad k \in 2 \ldots |\Gamma_i| \begin{cases} p_{ic}[k].x = \Phi_{ikc} + C_{ik} \\ p_{ic}[k].y = p_{ic}[k-1].y + C_{ik} \end{cases} \tag{11}$$

The initial relation (i.e. vertical distance at time 0) between different critical instant candidates is given by the difference in jitter-induced interference (the sum of all I_{ijc}^{Set1} instances). Furthermore, we want the time-induced interference to start with 0 (at time 0), see figure 6 in section 4. This is achived by subtracting the maximum of all jitter-induced interference (stored in $J_i^{ind}(\tau_{ua})$) when initialising $p_{ic}[1].y$ in equation 11.

The slanted stair variant of W_{ia}^{+} and W_{ib}^{+}, for our example transaction in figure 1 on page 4, are depicted in figure 8 and the corresponding p_{ia} and p_{ib} sets are illustrated by black and white circles respectively. For this example transaction we get the following two p_{ic}-s:

$$\begin{aligned} p_{ia} &= [\langle 0, -1\rangle, \langle 4, 1\rangle, \langle 6, 2\rangle] && \text{black circles} \\ p_{ib} &= [\langle 0, \ \ 0\rangle, \langle 8, 2\rangle, \langle 10, 3\rangle] && \text{white circles} \end{aligned}$$

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 represents 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 that corresponds to the convex corners of $T_i^{ind}(\tau_{ua}, t)$, we define the *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. Figure 9 illustrates the subsumes relation graphically, and the formal definition is:

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

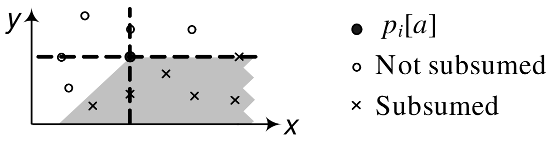


Figure 9: The subsumes relation

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 figure 1 we have:

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

All we have to do now is to find the concave corners (illustrated by crosses in figure 8 on the page before) and store them in the arrays T_i^c and T_i^t. This is done by the following algorithm:

```
for k := 1 to $|p_i|$ do
    $T_i^c[k]$ := $p_i[k].y$                                  // The interference is the y-value
    if k < $|p_i|$ then
        $T_i^t[k]$ := $p_i[k+1].x - (p_i[k+1].y - p_i[k].y)$  // Adjust the time for the slanted shape
    else
        $T_i^t[k]$ := $T_i$
done
```

For our figure 1 example transaction, T_i^c and T_i^t (correspons to crosses in figure 8) are as follows:

$$\begin{array}{ll} T_i^c & = [\ 0,\ 1,\ 2,\ \ 3] \\ T_i^t & = [\ 3,\ 5,\ 9,\ 10] \end{array}$$

In the special case that some task τ_{ij} has $\Phi_{ijc} = 0$, the first element of T_i^c will not be zero. However, since $T_i^{ind}(0) = 0$, we need to have at least one element in T_i^c that is zero. In such cases we prepend both the arrays T_i^c and T_i^t with a zero (stating that there will be 0 time induced interference for any time interval of length up to 0).

5.4 Precomputing P_i^c and P_i^t

T_i^c and T_i^t represents the interference for the first period ($t \leq T_i$). This a special case and does not represent the repetetive pattern. Interference in the following period ($T_i < t \leq 2T_i$) does, and we store this information in P_i^c and P_i^t.

Computing P_i^c and P_i^t is analogous to computing T_i^c and T_i^t, with the following differencies:

- Spill tasks from the split operation are not ignored, instead they are included in the calculations.
- In equation 11 on page 13 $p_{ic}[1].y$ defines the initial relation between different critical instant candidates. In P_i^c and P_i^t, however the relation should reflect this relation at the end of the first period. Using p'_{ic} to denote the points used when calculating P_i^c and P_i^t, we get the following modification to equation 11:

$$p'_{ic}[1].y = p_{ic}[|\Gamma_i|].y - \max_{x \in \Gamma_i} p_{ix}[|\Gamma_i|].y \tag{12}$$

Analogous to section 5.3 we get that the relation at time T_i for each critical instant candidate is stored in $p_{ic}[|\Gamma_i|]$. And in order to normalise the points to start at 0, we subtract the maximum of all $p_{ic}[|\Gamma_i|].y$.

5.5 Calculating the Interference

Once we have calculated the interference patterns and stored them in the arrays T_i^c and T_i^t (for interference when $t \leq T_i$), and P_i^c and P_i^t (for interference when $t > T_i$), we can modify the fast implementation of $T_i^{ind}(\tau_{ua}, t)$ (equation 9 on page 10) to use these arrays to lookup the interference for an arbitrary t:

$$\begin{aligned}
T_i^{ind}(\tau_{ua}, t) = &\begin{cases} T_i^c[n'] & \text{if } k < 1 \\ T_i^c[|T_i^c|] + (k-1) * P_i^c[|P_i^c|] + P_i^c[n''] & \text{if } k \geq 1 \end{cases} \\
k =& t \text{ div } T_i \\
t^* =& t \text{ rem } T_i \\
n' =& \min\{m \ : \ t^* \leq T_i^t[m]\} \\
n'' =& \min\{m \ : \ t^* \leq P_i^t[m]\}
\end{aligned} \tag{13}$$

where k represents the number of whole periods (T_i) in t, and t^* is the part of t that extends into the last period. It could be noted that $T_i^c[|T_i^c|]$ contains the sum of all interference during the first period, and $P_i^c[|P_i^c|]$ contains the sum of all interference during subsequent periods.

It should be noted that we in equation 13 calculate a *stepped stair* interference function. The arrays T_i^c, T_i^t, P_i^c, and P_i^t represents the discrete steps when interference increases and to which level it increases. The actual slant is not represented. It is of course possible to design equation 13 to with the slants. However, following the reasoning in our previous work [12] we can conclude that fix-point converge cannot occur during such slants. Hence, adding slants is needles, it would not change the calculated response-times, just increase the number of fix-point iterations needed.

Using equations 7, 8, and 13 (instead of the equation 4 on page 6), we obtain the tight and fast analysis. The analysis calculates the same values as the tight analysis (since the tight version of $I_{ijc}^{Set2}(t)$, equation 6 on page 8 is used). The computational effort is similar to that of the fast analysis. In the next section we demonstrate these properties of the tight and fast analysis.

6 Evaluation

In order to evaluate and quantify our proposed improvements, we have implemented the approximate response-time equations in appendix A, using: the original definition of $I_{ijc}^{Set2}(t)$ from section 2 (Orig), our tight analysis from section 3 (Tight), our fast analysis from section 4 (Fast), and our novel fast and tight analsysis from section 5 (Tight+Fast). Furthermore, we have also, as a comparison, implemented the exact analysis (Exact).

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

6.1 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, and
- The number of tasks per transaction to generate.
- 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 in the system.
- Transaction 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. The fraction is selected so that the 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 [4].

The task submitted for admission control has execution time 10.000 and deadline 500.000.

6.2 Description of Simulation Setup

We have implemented the complete response-time equations of appendix A which will show the effects of our improvements in a realistic scenario. The setup of the simulation is as follows: a task set is generated according to input parameters (system load, number of tasks, number of transactions, jitter). And to simulate an admission control situation we calculate the response time for a low priority task subjected to admission control.

The results in section 6.3 have been obtained by taking the mean value from 500 generated task-sets for each point in each graph. The admission probabilities and execution times are plotted with 95% confidence interval for the mean values.

We have measured three metrics from the simulations:

- "Admission probability (%)" — This metric measures the fraction of times (out of the 500 generated task sets) that the admission control task passes the admisson test, i.e., its response time is lower than its deadline.
- "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 (and the equivalent fast analysis). (Orig) and (Fast) are used as a baseline, hence no curves are plotted for those methods. Note that this metric says nothing about the size of improvements. In our previous work we have studied this and similar metrics for the tight analysis more thoroughly [7].
- "Execution time (s)" — The average execution time for performing the analysis. Note, that since this is an admission control scenario the time to pre-calculate the arrays T_i^c and T_i^t (which presumably has been made for all tasks already admitted to the system) is not included in the execution time. In our previous work we have shown that this time is insignificant for the fast analysis [6]. Our experiments show that this is also the case for the tight and fast analysis, where the execution time is approximately the double of the fast analysis (which is expected, since we calculate two sets of arrays).

6.3 Simulation Results

Figure 10 on the following page shows a typical subset of the simulation results. Note, that the exact analysis can only be run on small task sets, hence it is not present for larger tasks sets. Also, since the execution-time of the exact analysis is not relevant, we have not set the scale of y-axis to show all of its execution times, instead we have zoomed in on the approximate methods.

Figure 10 shows a base configuration where the number of tasks per transaction is 6, the number of transactions is 3, system load is 90% and the jitter is 0. From this base configuration we vary either (1) the number of tasks/transaction (the left column), (2) the number of transactions (the centre column), or (3) the jitter (the right column).

In the two first rows of figure 10 the effects of tightening the response times are shown. We notice that the two fast analysis techniques coincide with their respective slower analysis (as expected).

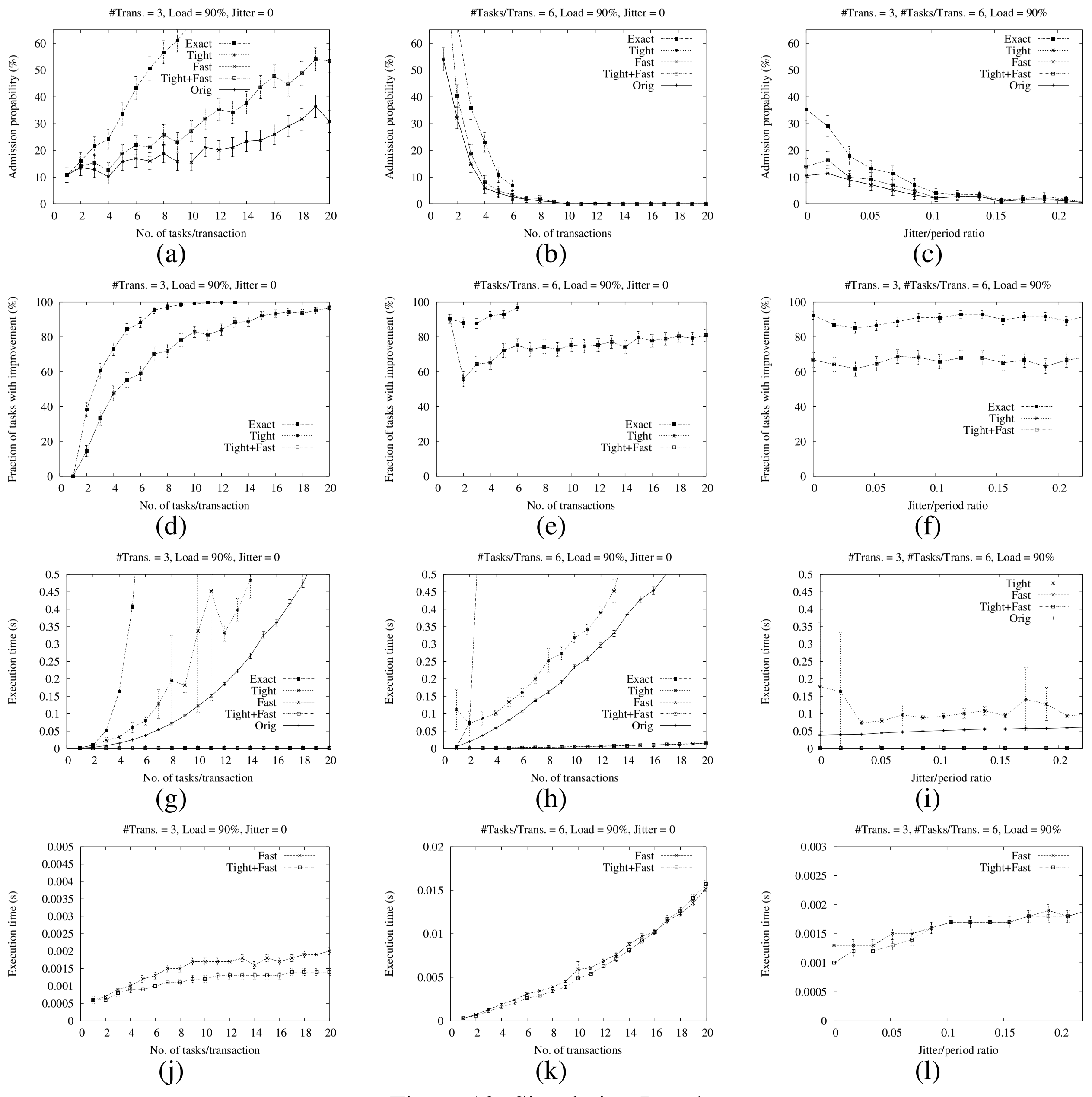


Figure 10: Simulation Results

We also see that the tight analysis gives a significant improvement over the original analysis. When the number of transactions or the jitter increases all methods drops in performance but their relative performance remains roughly the same (the second row shows the relative performance). We notice (e.g. in (f)) that the tight analysis methods makes almost as many improvements as does the exact analysis.

The two last rows of figure 10 shows the execution times. In row three we see that (Tight) has a serious penalty compared to (Orig) and that its execution time has very high variance (hence the large confidence intervals). The reason for this penalty is *not* increased complexity in each fix-point iteration. Rather; by not overestimating the interference, (Tight) progresses slower towards the fix-point (the progress made in each iteration is inversely proportional to the overestimation in each point). Hence, the penalty is due to the more faithful modelling of interference, not due to complexity of calculating the interference.

In the bottom most row we have zoomed in on the two fast analysis methods to expose their relative behaviour. Here we see that (Tight+Fast) has no penalty towards (Fast) and that either one of the methods could be the faster. We obtain this comparative improvement of (Tight+Fast) mainly since we revert to representing the interference with stepped stairs once we have calculated the (Tight) interference function. Hence, we gain this computational benefit by adding overestimation at places where we are *guaranteed* not to have fix-point solutions. It is still the case the (Tight+Fast) models the interference more faithfully than (Fast), hence the occasional cases where (Fast) is faster than (Tight+Fast). The fact the (Tight+Fast) can calculate lower response times may further help to improve its relative performance towards (Fast). The conclusion we draw from our simulations is that the disadvantage of faithful modelling of interference is compensated by the tighter solutions; hence the two methods show comparable execution times.

7 Conclusions

In this paper we have presented a novel method that allows for efficient and tight calculations of approximate worst-case response-times for tasks with offsets. We have successfully merged our two previous improvements [6, 7] to the original response-time analysis (RTA) presented by Tindell [14] and Palencia Gutierrez *et al.* [8]. Our improvements are orthogonal and complementary to other proposed extensions to the original offset analysis [9, 11]

The main effort in performing RTA for tasks with offsets is to calculate how higher (or equal) priority tasks interfere with a task under analysis. The essence of our fast method is to calculate and store this information statically and during response time calculations (fix-point iteration), use a simple table lookup. Our tight analysis exploits the fact that the interference imposed by higher priority tasks are overestimated in traditional RTA. By removing this overestimation, significantly tighter response-times can be calculated. The tight and fast analysis presented in this paper successfully combines our two methods to improve the RTA by identifying the periodic and non-periodic parts of the interference, and representing them separately.

Faster RTA have 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 utilisation. 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 tight and fast analysis is a very attractive choice to include in engineering tools and/or admission control software.

In a comprehensive simulation study we see that our novel analysis have very similar computational requirements to that of the fast analysis (sometimes slightly slower due to smaller steps in the fix-point iteration, and sometimes faster due to earlier fix-point convergence). Especially we notice that the computational disadvantage of the tight analysis (compared to the original analysis) is completely removed when comparing the tight and fast with the fast analysis. Example benchmarks include that the tight and fast analysis allows for over 20% more tasks admitted to the system (in an admission control situation), and that 80% of the tasks receives a lower response-time estimate, while the execution time is decreased by over two orders of a magnitude (compared to the original analysis).

References

[1] N.C. Audsley, A. Burns, R.I. Davis, K. Tindell, and A.J. Wellings.
Fixed Priority Pre-Emptive Scheduling: An Historical Perspective.
Real-Time Systems, 8(2/3):129–154, 1995.

[2] I-Logix.
Rhapsody.
http://www.ilogix.com/products/rhapsody.

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

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

[5] J. Mäki-Turja and M. Sjödin.
Improved Analysis for Real-Time Tasks With Offsets – Advanced Model.
Technical Report MRTC no. 101, Mälardalen Real-Time Research Centre (MRTC), May 2003.

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

[7] Jukka Mäki-Turja and Mikael Nolin.
Tighter Response-Times for Tasks with Offsets.
Submitted for publication, April 2004.

[8] J.C. 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.

[9] J.C. 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.

[10] Rational.
Rational Rose RealTime.
http://www.rational.com/products/rosert.

[11] 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.

[12] 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.

[13] TeleLogic.
Telelogic tau.
http://www.telelogic.com/products/tau.

[14] 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.

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.* [8].

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 3 in this paper):

$$W_{ic}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \Big(\left\lfloor \frac{J_{ij} + \Phi_{ijc}}{T_i} \right\rfloor + \left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil \Big) * C_{ij} \qquad \text{(26 in [8])}$$

where the phase between task τ_{ij} and the candidate critical instant task τ_{ic} is defined as (see equation 1 in this paper):

$$\Phi_{ijc} = T_i - (O_{ic} + J_{ic} - O_{ij}) \bmod T_i \qquad \text{(17 in [8])}$$

The approximation function for transaction Γ_i which considers all candidate τ_{ic}-s simultaneously, is defined by (see equation 4 in this paper):

$$W_i^*(\tau_{ua}, w) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, w) \qquad \text{(27 in [8])}$$

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):

$$\begin{aligned} L_{uac} =& B_{ua} + (p - p_{0,uac} + 1)C_{ua} + \\ & W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac}) \end{aligned} \qquad \text{(30 in [8])}$$

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:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1 \qquad \text{(29 in [8])}$$

and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil \qquad \text{(31 in [8])}$$

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

$$\begin{aligned} w_{uac}(p) =& B_{ua} + (p - p_{0,uac} + 1)C_{ua} \\ & + W_{uc}(\tau_{ua}, w_{uac}(p) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p)) \end{aligned} \qquad \text{(28 in [8])}$$

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

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua} \qquad \text{(32 in [8])}$$

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}:

$$R_{ua} = \max_{\forall c \in hp_u(\tau_{ua}) \cup a} \Big[\max_{p = p_{0,uac}, \dots, p_{L,uac}} (R_{uac}(p)) \Big] \qquad \text{(33 in [8])}$$