Real-Time Syst
DOI 10.1007/s11241-008-9050-9

Efficient implementation of tight response-times for tasks with offsets

Jukka Mäki-Turja · Mikael Nolin

Abstract Earlier approximate response time analysis (RTA) methods for tasks with offsets (transactional task model) exhibit two major deficiencies: (i) They overestimate the calculated response times resulting in an overly pessimistic result. (ii) They suffer from time complexity problems resulting in an RTA method that may not be applicable in practice. This paper shows how these two problems can be alleviated and combined in one single fast-and-tight RTA method that combines the best of worlds, high precision response times and a fast approximate RTA method.

Simulation studies, on randomly generated task sets, show that the response time improvement is significant, typically about 15% tighter response times in 50% of the cases, resulting in about 12% higher admission probability for low priority tasks subjected to admission control. Simulation studies also show that speedups of more than two orders of magnitude, for realistically sized tasks sets, compared to earlier RTA analysis techniques, can be obtained.

Other improvements such as Palencia Gutiérrez, González Harbour (Proceedings of the 20th IEEE real-time systems symposium (RTSS), pp. 328–339, 1999), Redell (Technical Report TRITA-MMK 2003:4, Dept. of Machine Design, KTH, 2003) are orthogonal and complementary which means that our method can easily be incorporated also in those methods. Hence, we conclude that the fast-and-tight RTA method presented is the preferred analysis technique when tight response-time estimates are needed, and that we do not need to sacrifice precision for analysis speed; both are obtained with one single method.

Keywords Response-time analysis · Fixed priority scheduling · Tasks with offsets

J. Mäki-Turja (✉) · M. Nolin
Mälardalen Real-Time Research Centre (MRTC), P.O. Box 883, 72123 Västerås, Sweden
e-mail: jukka.maki-turja@mdh.se

M. Nolin
e-mail: mikael.nolin@mdh.se

1 Introduction

Response-Time Analysis (RTA) (Audsley et al. 1995; Sha et al. 2004) is a powerful and well established schedulability analysis technique. RTA is a method to calculate upper bounds on response-times for tasks in 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. Furthermore, RTA is not only used as a schedulability analysis tool, but it is also used in a wider context. For example, schedulability analysis is performed in the inner loop of optimization or search techniques such as task attribute assignment and task allocation. These methods require that RTA methods provide tight response times and that implementations are efficient in order to be useful in engineering tools for 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, the transactional task model (Tindell 1992). Palencia Gutiérrez and González Harbour formalized and extended the work of Tindell in (Palencia Gutiérrez and González Harbour 1998).

In this paper we will show that the approximate RTA for task with offset presented in their work calculates unnecessarily pessimistic response-times. As a remedy, we will present our tight analysis. The main source for this improvement comes from more accurate modeling of inter-task interference. We also present an implementation technique that enables fast and efficient RTA calculations. The essence of this approach is to statically store higher priority task interference, and during equation solving (fix-point calculations) use a simple and fast table lookup. Furthermore, we combine these two improvements in a single RTA method resulting in a method that produces tight response times in a fast analysis time. Simulation results indicate that typically 15% tighter response times can be obtained, and in an analysis time that is two order of magnitudes faster, than with comparable methods. This gives an RTA method where one does not have to sacrifice accuracy for speed or vice versa, both are obtained with the presented *fast-and-tight* RTA method.

Paper outline We begin with presenting background and motivation together with related work in Sect. 2. In Sect. 3 we revisit and restate the original offset RTA method introduced by Tindell (1992), which was extended and formalized by Palencia Gutiérrez and González Harbour (1998), in more detail. We continue by discussing a misconception concerning high priority task interference in Sect. 4. Where we also present a remedy: "imposed" interference. In Sect. 5 we modify the existing offset RTA method to use "imposed" interference instead, and show some consequences and proofs of correctness. In Sect. 6 we present how this tight RTA can be efficiently implemented resulting in a *fast-and-tight* response-time analysis method. Section 7 presents evaluations quantifying our two improvements, and finally, Sect. 8 concludes the paper.

2 Background, motivation and related work

The transactional task model (Tindell 1992; Palencia Gutiérrez and González Harbour 1998), was introduced because in that model tasks have dependencies in their release times, so called offsets. With these offsets the critical instant assumption, where all tasks are to be released simultaneously, became too pessimistic. This task model is most widely known to model precedence relations among tasks in a distributed system. However, offsets only specify temporal dependencies among task release times in a transaction. The work in this paper originates from an industrial collaboration where hybrid, static and dynamic, scheduling is used (Mäki-Turja et al. 2005). In that work there are no precedence relations among tasks in one transaction, just temporal dependencies in task release times.

The exact RTA presented in (Tindell 1992; Palencia Gutiérrez and González Harbour 1998) is computationally intractable for anything else but small task sets. A more detailed discussion of this can be found in Sect. 3.2. Therefore, they also provided an approximate RTA that could be used in practice. It is this approximate RTA that produces unnecessarily pessimistic response-times. The objective of this paper is to improve upon this approximate RTA by more faithful modeling of inter-task interference.

Other improvements made to the RTA for tasks with offsets such as (Palencia Gutiérrez and González Harbour 1999; Redell 2003) are orthogonal and complementary (they perform better with large jitter) which means that our method can easily be incorporated to their method or vice versa. Rahni et al. (2007) show that in the special case of so called monotonic transactions the table lookup technique can be made even faster and always produces exact response times. However, the applicability of this approach is limited since most systems cannot be expected to consist of only monotonic transactions.

As discussed in (Mäki-Turja et al. 2005), RTA of tasks with offsets is useful to analyze hybrid scheduled systems with static and FPS (Fixed Priority Scheduling). Palencia and Harbour has also presented RTA for hybrid scheduled system using EDF (Earliest Deadline First) (Palencia Gutiérrez and González Harbour 2003b). Their work has also been extended to the transactional task model (Palencia and González Harbour 2003a). Other work on analysis of the transactional model include Pop et al. (2003), which focus on systems with time-triggered and event-triggered domains.

Other methods to tackle schedulability of hybrid scheduled system and systems with complex task-arrival patterns have been proposed, e.g. (Regher et al. 2003; Fersman 2003).

Concretely, this paper extends the method originally proposed by Palencia Gutiérrez and González Harbour (1998) in two dimensions: (1) We propose a tighter method to calculate response-times (the method never calculates higher response-times, and for a large portion of tasks the calculated response-time is lower), (2) we decrease the computation time for calculating the response-times.

3 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets (Tindell 1992; Palencia Gutiérrez and González Harbour 1998) and illustrates some intuition behind the analysis and the formulae.

3.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a periodic sequence of events with period T_i (for non-periodic events T_i denotes the minimum inter-arrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ tasks, and each task may not be activated (released for execution) until a 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.

The relation between event arrival, offset, jitter and task release is graphically visualized in Fig. 1. After the event arrival, task τ_{ij} is not released for execution until its offset (O_{ij}) has elapsed. The task release may be further delayed by jitter (maximally until $O_{ij} + J_{ij}$) making its exact release uncertain. For a more extensive explanation of task parameters see (Palencia Gutiérrez and González Harbour 1998). Parameters for an example transaction (Γ_i) with two tasks (τ_{i1}, τ_{i2}) are depicted in Fig. 2.

3.2 Response-time analysis

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

In the classical RTA (without offsets) the *critical instant* for τ_{ua} occurs when it is released at the same time as all higher priority tasks (Joseph and Pandya 1986; Liu and Layland 1973). In a task model with offsets this assumption yields pessimistic response-times since some tasks cannot be released simultaneously due to offset relations. Therefore, Tindell (1992) relaxed the notion of critical instant to be:

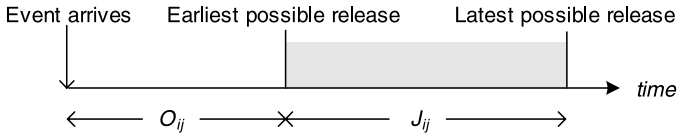


Fig. 1 Relation between an event arrival, offset, jitter and task release

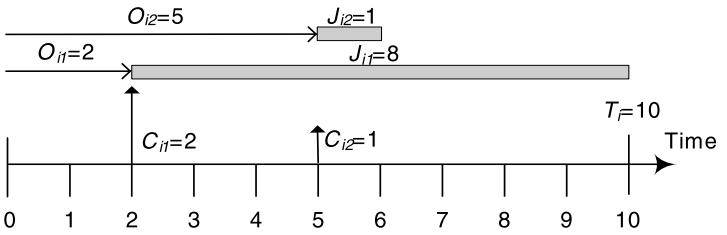


Fig. 2 An example transaction Γ_i

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 (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 Gutiérrez and González Harbour (1998) formalized and generalized Tindell's work. We will in this paper use the more general formalism of Palencia Gutiérrez and González Harbour, although our proposed method is equally applicable to Tindell's original algorithm.

3.3 Interference function

Central to RTA is to capture the worst case 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 coincides with the critical instant (denoted τ_{ic}), will do so after experiencing its worst case jitter delay.

In order to determine the amount of $Set2$ interference for a task, τ_{ij}, we need to know when the first activation of τ_{ij} occurs after the critical instant. This phasing between a task, τ_{ij}, and the critical instant, which according to Theorem 1 occurs at $O_{ic} + J_{ic}$, becomes:

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

Figure 3 illustrates the four (two transactions and two critical instant candidates) different Φ_{ijc}-s that are possible for our example transaction in Fig. 2. Note that the time of origin is set at the critical instant. The upward arrows denote task releases. The height of the upward arrows denotes the amount of execution released.

Figure 3(a) shows, for the case when τ_{i1} coincides with the critical instant, the invocations in $Set1$ (arriving at time 0) and the first invocation in $Set2$. Figure 3(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 interference caused by task τ_{ij} can be divided into two parts:

1. The part caused by instances in $Set1$ (which is independent of the time interval t), I_{ijc}^{Set1}, and
2. the part caused by instances in $Set2$ (which is a function of the time interval t), $I_{ijc}^{Set2}(t)$.

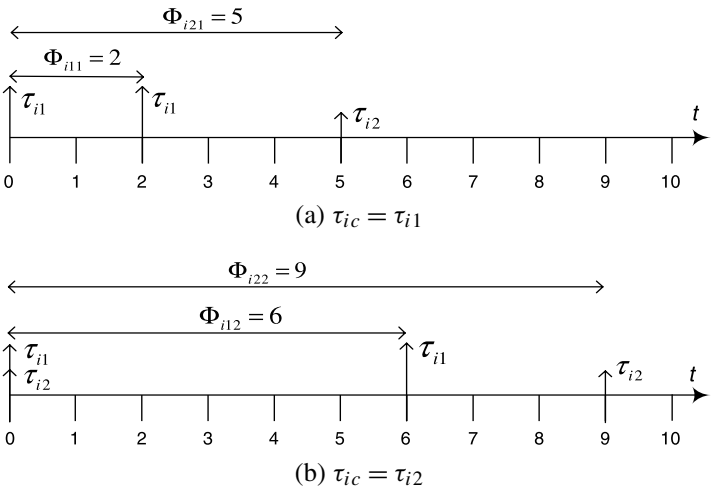


Fig. 3 Φ-s for the two candidates in Γ_i

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

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

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

Note that the definition of Φ_{ijc} in (1) is a redefinition of the original definition (see (17) in (Palencia Gutiérrez and González Harbour 1998), see Appendix A). However, $W_{ic}(\tau_{ua}, t)$ using these two definitions are equivalent as proved by the following theorem.

Theorem 1 *$W_{ic}(\tau_{ua}, t)$ using definition of Φ_{ijc} according to (1) is equivalent to using (17) in (Palencia Gutiérrez and González Harbour 1998).*

Proof reference The theorem is proved by algebraic equivalence in Appendix B. □

3.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination (Tindell 1992; Palencia Gutiérrez and González Harbour 1998). 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 (Tindell 1992; Palencia Gutiérrez and González Harbour 1998):

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

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

As an example, consider again transaction Γ_i depicted in Fig. 2. Figure 4 shows the interference functions (W_{i1} and W_{i2}) for the two candidates, and it shows how W_i^* is derived from them by taking the maximum of the two functions at every t. Given the interference (W_i^*) each transaction causes, during a time interval of length t, the response time of τ_{ua} (R_{ua}) can be calculated. Appendix A presents the complete set of formulae for these response-time calculations.

4 The concept of interference

Classical response time analysis for Liu and Layland's periodic task model (Liu and Layland 1973) (where a task τ_i has a period T_i and worst-case execution-time C_i),

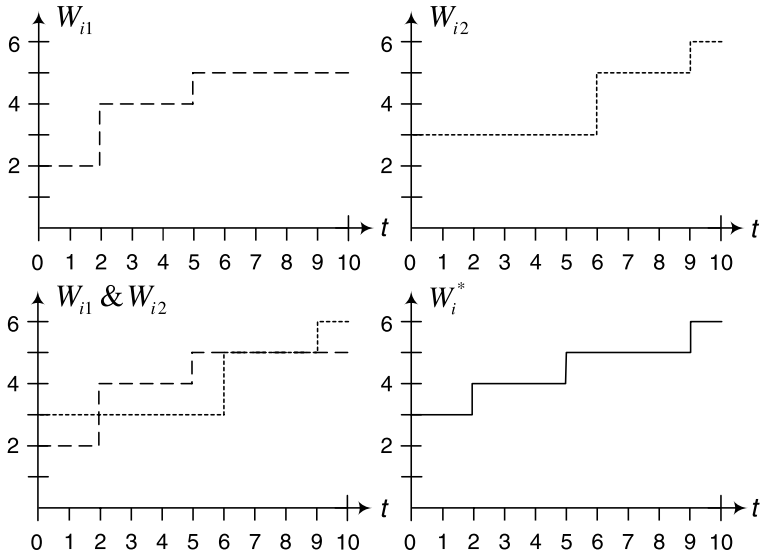


Fig. 4 $W_{ic}(\tau_{ua}, t)$ and $W_i^*(\tau_{ua}, t)$ functions

presented first by Joseph and Pandya (1986), states that the worst case response time, for a task under analysis (τ_i), occurs when it is released at the same time as all higher priority tasks. Under this assumption the worst case response time, R_i is:

$$R_i = C_i + \sum_{\forall j \in hp(i)} interference_j(R_i) \tag{5}$$

where C_i is the execution-time of task i, $hp(i)$ is the set of higher priority tasks, and $interference_j(t)$ is the amount of interference task j causes during time-interval t. The interference formula presented by Joseph and Pandya (1986) is:

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

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

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

$$\frac{\mathrm{d}interference_j(t)}{\mathrm{d}t} \leq \frac{\mathrm{d}t}{\mathrm{d}t} \quad \Rightarrow \quad \frac{\mathrm{d}interference_j(t)}{\mathrm{d}t} \leq 1 \tag{6}$$

Theorem 2 *Consider a task τ_j, activated at time 0 and subsequently with period T_j, having execution-time C_j $(0 < C_j \leq T_j)$. For a positive time-interval $t = kT_j + t'$ (where $k \in \mathbb{N}$ and $0 \leq t' < T_j$), $kC_j + \min(t', C_j)$ is an upper bound on the interference τ_j can impose on any lower priority task during t.*

Proof During kT_j, τ_j imposes an amount of interference of kC_j (task instances are activated periodically), one instance for every period. During the remaining time interval, t', τ_j can, according to (6), never impose more interference than the length of the interval itself. Hence, $kC_j + t'$ is an upper bound on the interference τ_j, can impose during t.

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

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

We denote the concept of interference which is bounded by $interference_j(t)$ and Theorem 2 with "imposed" interference. As an example, consider a task with $T_j = 10$ and $C_j = 4$. Figure 5 illustrates the difference between "released for execution" and "imposed" interference for $t \in 0 \ldots 20$. The released for execution interference increases in a *stepped stair* fashion, whereas the imposed interference increases in a *slanted stair* fashion (with a derivative of 1 in the slants).

In Fig. 5 the shaded areas represent the overestimation made by the released for execution concept. Note however, for classical response-time analysis this overestimation has no effect on the calculated response-time, and Joseph and Pandya's equation *does* yield exact worst case response-times. The reason for this is that the response-time analysis calculation (fix-point iteration) has no solutions in the shaded areas (as discussed further in Sect. 5.3). Also for exact RTA of task with offsets (Tindell 1992) this overestimation does not yield any pessimism in the calculated response-times.

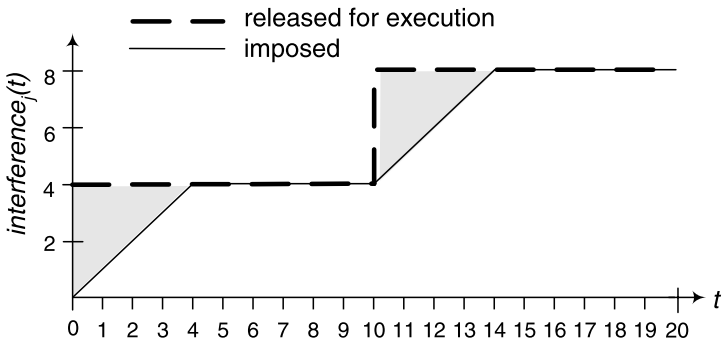


Fig. 5 Released for execution vs. imposed interference

5 Tight offset RTA

We begin this section with a simple and illustrative example of how the original analysis overestimates the response-time. Consider a simple transaction Γ_i depicted in Fig. 6 where jitter (J_{ij}) and blocking (B_{ij}) are zero. Also consider a lower priority task, τ_{ua}, which is the single task in transaction Γ_u, with $C_{ua} = 2$. For this simplified task model where $B_{ij} = J_{ij} = 0$, $D_{ua} \leq T_u$ only one instance of the task under analysis is active at any point in time. This means that the response time formulae, for the single lower priority task, presented in Appendix A, can be reduced and simplified to:

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

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

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

Where column "Iter#" denotes the iteration number, "t" the time interval, "W_{i1}" and "W_{i2}" denotes $W_{ic}(\tau_{ua}, t)$ for the two candidate tasks τ_{i1} and τ_{i2} respectively. "W_i^*" denotes the value of $W_i^*(\tau_{ua}, t)$, and "R_{ua}" the calculated response-time for the iteration. In iteration 4 the fix-point iteration terminates (R_{ua} has the same value as in the previous iteration), and the calculated response time is $R_{ua} = 8$. However, it can easily be seen that a task with $C_{ua} = 2$ can never be preempted by both τ_{i1} and τ_{i2} since both tasks are separated by at least 2 units of idle time. Hence, the actual worst case response-time is overestimated.

5.1 Using imposed interference

One property of the ceiling expression of $I_{ijc}^{Set2}(t)$ in (2) is that it returns the amount of interference "released for execution" at time t. This results in a *stepped stair* interference function. If we modify $I_{ijc}^{Set2}(t)$ in (2) so that it returns the interference "imposed" on τ_{ua} we get a *slanted stair* function (as shown in Sect. 4). Our redefined

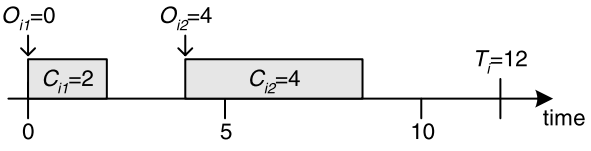


Fig. 6 A simple example transaction

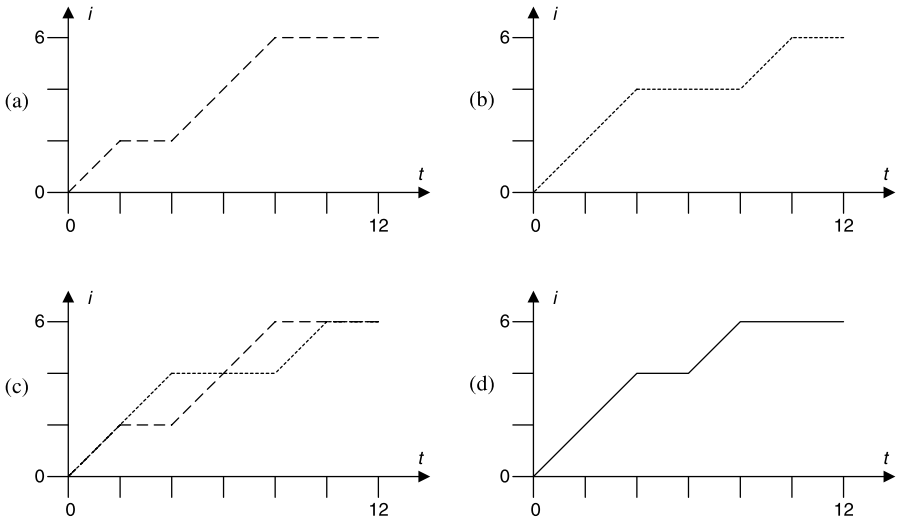


Fig. 7 Interference *imposed* on τ_{ua} by our example transaction

version of $I_{ijc}^{Set2}(t)$ then becomes:

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

where Φ_{ijc} is defined in (1) and x is used to generate the slants of the "imposed" interference function.

The slanted stairs, for the example of Fig. 6, generated by (8) are shown in Figs. 7(a) and 7(b), and Fig. 7(c) shows them superimposed. Using our new version of $I_{ijc}^{Set2}(t)$ in (3) we get the maximized slanted stairs interference function, representing the approximation function W_i^*, shown in Fig. 7(d).

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

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

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

Hence, the calculated response-time ($R_{ua} = 6$) is lower than that of the original analysis ($R_{ua} = 8$), and in Sect. 7 we will quantify this improvement in a simulation study. Section 5.3 also gives an intuitive explanation of why the calculated response times are lower.

5.2 Correctness criteria

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

- The new definition of $I_{ijc}^{Set2}(t)$ is not allowed to be greater than the old definition (for any t). If this condition holds, the analysis performed with the new definition is guaranteed not to yield larger response-times than the old definition does.
- The new definition of $I_{ijc}^{Set2}(t)$ must not underestimate the interference caused by $Set2$-tasks. If the interference is underestimated, analysis performed with the new definition could yield unsafe response-time estimates.
- The new definition of $I_{ijc}^{Set2}(t)$ must yield a monotonically increasing interference function $W_i^*(\tau_{ua}, t)$. Monotonicity is required to guarantee that at least one solution to the response-time formula exists and that fix-point iteration finds the smallest existing solution (Sjödin and Hansson 1998).

Note that the time of origin is set at the critical instant.

Theorem 3 *For a given task under analysis, τ_{ua}, and one candidate task, $\tau_{ic} \in \Gamma_i$, our new definition of I_{ijc}^{Set2} (see (8)) is never greater than the old definition (see (2)).*

Proof x, as defined in (8), is used to decrease the calculated value of I_{ijc}^{Set2}. Since x, by definition, is never negative, it can never contribute to making (8) greater than (2). □

Theorem 4 *For any time interval $t \geq 0$ our new definition of $I_{ijc}^{Set2}(t)$ (8) never underestimates the interference caused by $Set2$ task instances.*

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

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

For times at or after the first invocation, i.e. $t \geq \Phi_{ijc}$, we have $t^* \geq 0$. Now, assume $t^* = kT_i + t'$, where $k \in \mathbb{N}$ and $0 \leq t' < T_i$ (the relation between t, t^* and t' is graphically visualized in Fig. 8). If the interference calculated by (8) is not below the *safe upper bound* defined by Theorem 2:

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

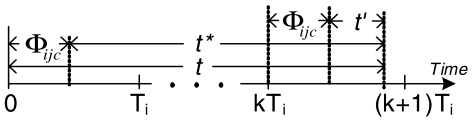


Fig. 8 Relation between t, t^* and t'

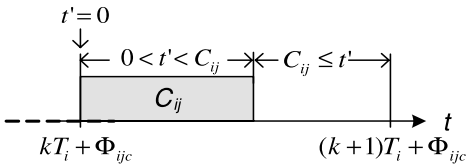


Fig. 9 Three proof cases for t'

the interference is not underestimated.

We divide the proof into three cases depending on the value of t' for a time-interval t (the three different cases are depicted graphically in Fig. 9):

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

□

Theorem 5 *The definition of* $I_{ijc}^{Set2}(t)$ *in* (8) *is* (*non-strictly*) *monotonically increasing with the time interval* t.

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

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

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

□

5.3 Discussion

At first glance, it is not obvious that lowering the interference function $W_{ic}(\tau_{ua}, t)$ should automatically give lower response-times. In fact, the stepped-stair interference function has been used for many years to represent the interference in RTA (Audsley et al. 1995; Audsley et al. 1993), without introducing any pessimism.

The underlying reason why stepped stairs (in analysis without offsets) does not introduce pessimism can be found in (Sjödin and Hansson 1998). In short, the fix-point iteration will terminate when the sum of all interference functions (demand) meets the line from origin with slope 1 (supply). Hence, replacing stepped stairs with slanted stairs (with slope 1) will not contribute to earlier fix-point convergence.

However, in approximate response-time analysis with offsets, the interference functions, W_{ic}-s, are not used directly in the fix-point iterations. Instead they are first subjected to a maximization function (see (4)). This situation can be compared to floating point addition: if you round up the floating point numbers at each calculation step, instead of just in the end, you will lose precision. This corresponds to passing released for execution interference, instead of more precise imposed interference, to the maximization function. Another view of this is that by using slanted-stair functions as input to the maximization function, one essentially "delays" the time it takes for one low-interference scenario to overtake a high-interference scenario, as we show below.

Figure 10(a) shows our simple example transaction from Fig. 6 with two arrows denoting the two possible scenarios for the critical instant (one "dashed" scenario

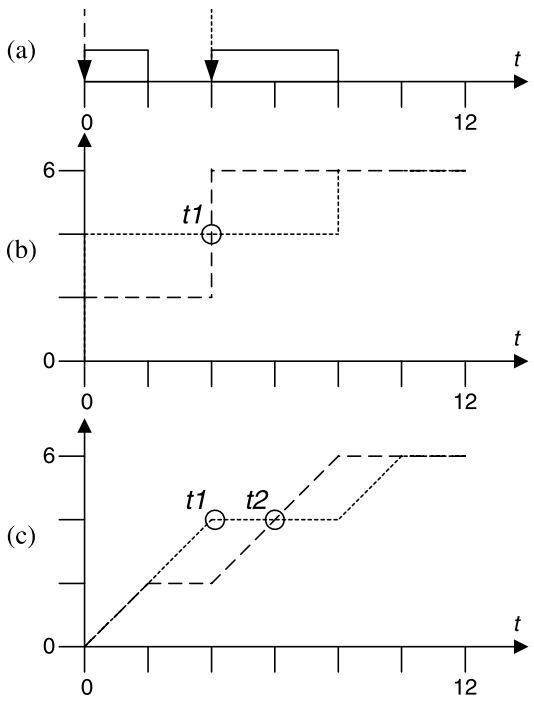


Fig. 10 Stepped stairs vs. slanted stairs

and one "dotted" scenario). Figures 10(b) and 10(c) shows the stepped stairs and slanted stairs interference functions, respectively, for both scenarios. For times $t < t1$, the dotted scenario is the one with highest interference. Time $t1$ corresponds to the release of the second task in the dashed scenario. For the stepped stairs case, this means immediately adding another 4 units of interference to the dashed scenario, hence immediately making it the scenario with the highest interference. However, for the slanted stairs case, the time $t1$ means that the dashed line starts to increase, but not until time $t2$ it catches up with the dotted scenario. Hence, the interval between $t1$ and $t2$ represents the time by which the slanted stairs "delay" the dashed scenario to catch up with the dotted scenario. If fix-point convergence can be achieved during this time interval, then approximate RTA with imposed interference will calculate a lower response time than does RTA with released for execution interference.

6 Fast and tight analysis

When calculating response times, the function $W_i^*(\tau_{ua}, t)$ (see 4) 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. However, since $W_i^*(\tau_{ua}, t)$ has a repetitive pattern (as we will show in Theorem 7), a lot of computational effort could be saved by representing the interference function statically, and during response-time calculation use a simple lookup function to obtain its value. This section shows how the function $W_i^*(\tau_{ua}, t)$ could be changed to use such pre-computed information and how to calculate and store that information.

6.1 Periodicity of 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 previous fast analysis for the original RTA with offsets (Mäki-Turja and Nolin 2004), the full interference of each task occurs within the first period. Hence, one 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. Figure 11 shows an example critical instant candidate where the interference from task z *spills* into next period.

As seen in Fig. 11, the interference for the first period differs from that of later periods. Obviously, there can be no spill into 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.

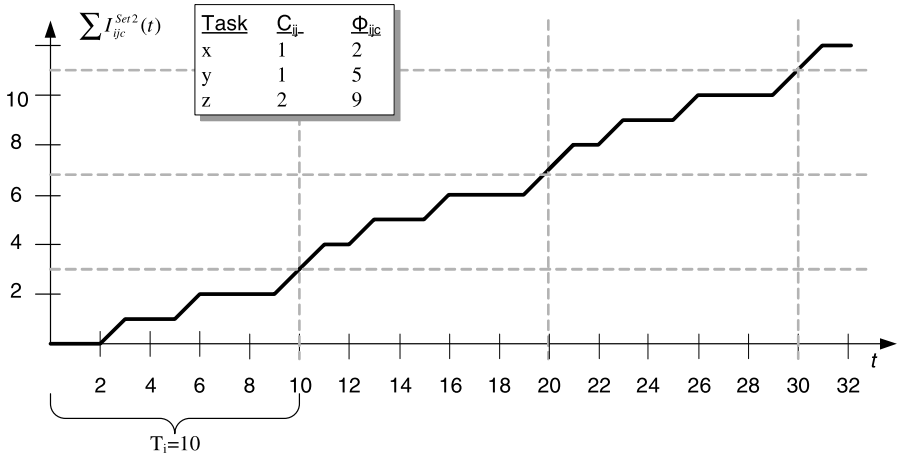


Fig. 11 Interference spilling over periods

6.2 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 For each pair of tasks, j' and j, where j' is released before j and does not have a chance to finish its execution before the release of j, 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 pair of tasks has 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

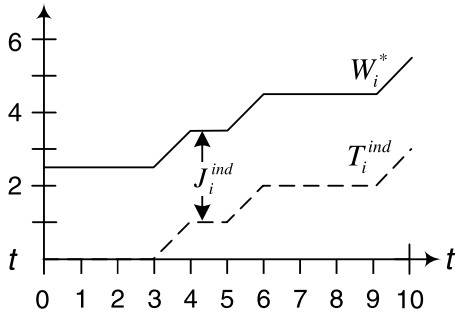


Fig. 12 Relation between $W_i^*(\tau_{ua}, t)$, $J_i^{ind}(\tau_{ua})$, and $T_i^{ind}(\tau_{ua}, t)$

amount of interference that occurs at the beginning of the subsequent period. The definitions are:

$$C_{ij'} = C_{ij} - S_{ijc}, \qquad C_{ij''} = S_{ijc},$$
$$\Phi_{ij'c} = \Phi_{ijc}, \qquad \Phi_{ij''c} = 0.$$

6.3 Jitter and time induced interference

The key to make a static representation of $W_i^*(\tau_{ua}, t)$ is to recognize 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 dependent 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 we prove below).

We redefine (4) using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t). \tag{9}$$

This partitioning of $W_i^*(\tau_{ua}, t)$ is visualized in Fig. 12. $J_i^{ind}(\tau_{ua})$ 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 (3)) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1}. \tag{10}$$

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:

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^+(\tau_{ua}, t) \tag{11}$$

where

$$W_{ic}^+(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t)) - J_i^{ind}(\tau_{ua}). \tag{12}$$

The correctness of our method requires that our new definition of $W_i^*(\tau_{ua}, t)$ in (9) is functionally equivalent to the definition in (4).

Theorem 6 *$W_i^*(\tau_{ua}, t)$ as defined in (4) and $W_i^*(\tau_{ua}, t)$ as defined in (9) are equivalent.*

Proof reference The theorem is proved by algebraic equivalence in Appendix B. □

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 7 *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 Appendix B. □

6.4 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.

6.4.1 Spill task not accounted for

For each critical instant candidate, τ_{ic}, tasks are ordered, merged, and split according to Sect. 6.2. 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 (12). The following equations define the array p_{ic}:

$$\begin{aligned}
p_{ic}[1].x &= 0, \\
p_{ic}[1].y &= \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} - J_i^{ind}(\tau_{ua}), \\
p_{ic}[k].x &= \Phi_{ikc} + C_{ik}, \quad k \in 2 \ldots |\Gamma_i|, \\
p_{ic}[k].y &= p_{ic}[k-1].y + C_{ik}, \quad k \in 2 \ldots |\Gamma_i|.
\end{aligned} \tag{13}$$

$p_{ic}[1].y$ 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

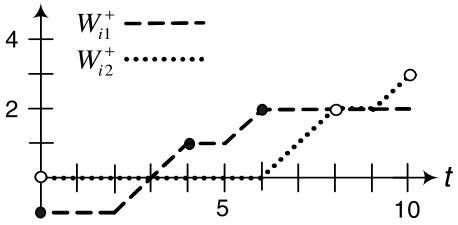


Fig. 13 Visual representation of p_{ic} sets

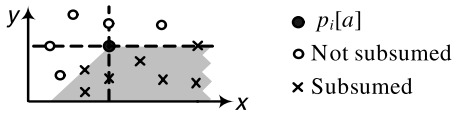


Fig. 14 The subsumes relation

Fig. 12) 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 (13).

The W_{i1}^+ and W_{i2}^+, for our example transaction of Fig. 2, are depicted in Fig. 13 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:

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

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 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. Figure 14 illustrates this relation graphically with a shaded region, and the formal definition is:

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

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. 2 we have:

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

6.4.2 *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 additional merge and order operations.
- In (13) $p_{ic}[1].y$ 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 (13):[1]

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

6.5 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 by black and white circles in Fig. 15. Note that in the absence of spill tasks, the sets p_i and p'_i are identical.

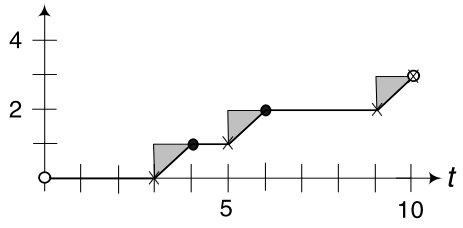
It can be proven that the fix-point iterative solution to (28′) in Appendix B, which is the equation where the interference function is used, cannot have any solution during the slants.

Theorem 8 *Equation* (28′), *in Appendix B, 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 proven in Appendix B. □

No solutions to the response-time equation can exist during the slant of any interference function. Hence, we can remove the slants and replace them with a stepped stair function, as illustrated by the grey areas of Fig. 15, without introducing any

Fig. 15 Removing the slants



[1] Analogous to (13), we normalize the points to start at 0, hence we subtract the maximum of all $p_{ix}[|\Gamma_i|].y$.

pessimism in the resulting response times. 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. 15 we will speed up the fix-point convergence without modifying the calculated response-times.

Removing the slants is equivalent to transforming the convex corners to concave corners (illustrated by crosses in Fig. 15[2]). The rules for finding the concave corners, v_i, from a set of convex corners, p_i, is as follows:

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

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. 2, v_i becomes (indicated by crosses in Fig. 15):

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

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 of spill tasks), $v_i[1].x$ will not be zero. However, since $T_i^{ind}(0) = 0$ (follows from (11)), 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).

6.6 $T_i^{ind}(\tau_{ua}, t)$ 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 Sect. 6.4. Next we will remove the slants for both these point sets as described in Sect. 6.5 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

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

reduced to a fast lookup function:

$$
\begin{aligned}
T_i^{ind}(\tau_{ua}, t) &= \begin{cases} v[n].y & \text{if } k < 1, \\ v_i[|v_i|].y + (k-1) * v_i'[|v_i'|].y + v_i'[n'].y & \text{if } k \geq 1, \end{cases} \\
k &= t \operatorname{div} T_i, \\
t^* &= t \operatorname{rem} T_i, \\
n &= \min\{m : t^* \leq v_i[m].x\}, \\
n' &= \min\{m \ : \ t^* \leq v_i'[m].x\}
\end{aligned}
\tag{14}
$$

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.

6.7 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, in the worst theoretical case, 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 (14) requires comparison of each pair of points; resulting in cubic time-complexity ($O(|\Gamma_i|^3)$) for pre-calculating v_i and v_i'.[3] During the fix-point iteration phase, a binary search through a quadratic sized array is performed (either v_i or v_i' in (14)), resulting in $O(\log|\Gamma_i|^2)$ time complexity for calculating W_i^* according to (9). The original complexity for calculating W_i^* according to (4) 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 formulae in Appendix A). Assume X denotes number of fix-point iterations needed, then the overall complexity for the original approach (see 4) is ($O(X|\Gamma_i|^2)$), whereas our method (see (9) and (14)) 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 time complexity.

7 Evaluation

We have, by simulation studies, evaluated both improvements presented in this paper. The main purpose of the simulation is to quantify our improvements and to investigate what parameters affects the improvements in different ways. We use a random task generator which does not necessarily correspond to real systems. Furthermore, the

[3] In Sect. 7 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.

transactional task model can be used for different system models such as distributed systems (Palencia Gutiérrez and González Harbour 1998) or combined statically and dynamically scheduled systems (Mäki-Turja et al. 2005), we have therefore kept the simulation setup general as possible.

7.1 Evaluating response-time precision

In order to evaluate and quantify our proposed improvement of Sect. 5, we have implemented the approximate response-time equations of Appendix A, using both the original definition of $I_{ijc}^{Set2}(t)$ from Sect. 3 and our tighter version of $I_{ijc}^{Set2}(t)$ from Sect. 5. Furthermore, we have also, as a comparison, implemented the exact analysis.

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

7.1.1 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 utilization),
- The number of transactions to generate,
- The number of tasks per transaction to generate, and
- Jitter fraction (in % of the transaction periods).

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

- The total system load is proportionally distributed over all transactions.
- 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, and is selected so that the transaction load (as defined by the first property) is obtained.
- The jitter is set to the jitter fraction of the period ($J_{ij} = f * T_i$).[4]
- Blocking (B_{ij}) is set to zero.
- The priorities are assigned in rate monotonic order (Liu and Layland 1973).

[4] In the distributed system model jitter should be calculated as the response time of a preceding task. However, our purpose of the simulation is to be as general as possible (remember that the task model can be used to model, e.g., static schedules) and to see how the amount of jitter affects the improvements presented in this paper.

7.1.2 Description of simulation setup

The heart of the improvement made to the approximate response time analysis is a new definition of $I_{ijc}^{Set2}(t)$. We have implemented the response-time equations of Appendix A which will show the effects of our improvements in a realistic scenario.

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

We have calculated and compared response times for our tighter analysis (Tight), Palencia Gutiérrez and González Harbour's original analysis (Orig) and the exact analysis (Exact). The results in Sect. 7.1.3 have been obtained by taking the mean value from 1000 generated task-sets for each point in each graph. The graphs in the left and in the right columns also show the 95% confidence interval for these mean values.

We have measured three metrics from the simulations:

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

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

7.1.3 Simulation results

In the simulations we have varied our four task-generator parameters in different ways. Figure 16 shows a subset of the simulation results. The exact analysis can only be run on small task sets; hence it is not present for larger tasks sets. For every parameter that is varied we show all three metrics described in the previous section, corresponding to column one, two and three respectively in Fig. 16. (In Fig. 16, note that "Tasks $= x$" denotes "x tasks/transaction".)

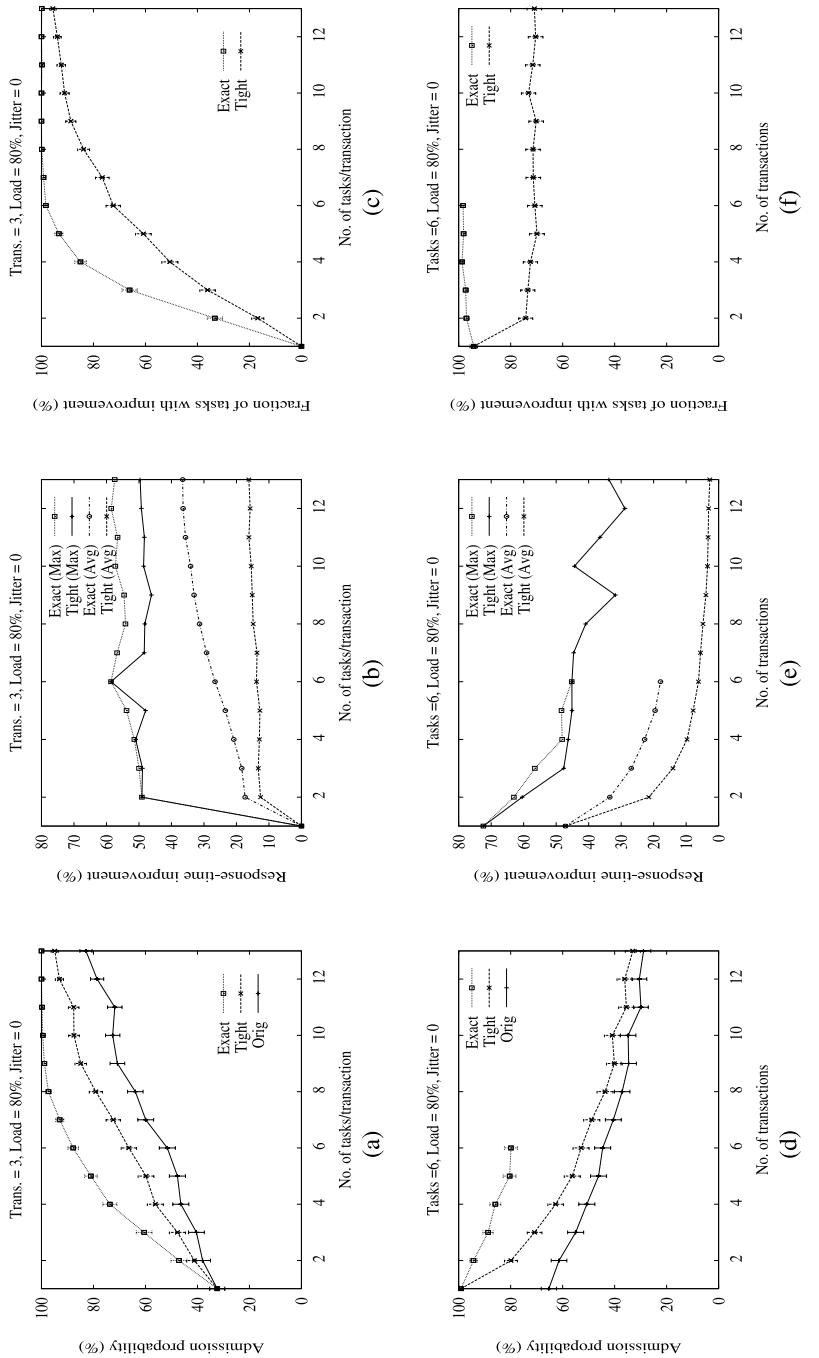


Fig. 16 Results when evaluating the tightness

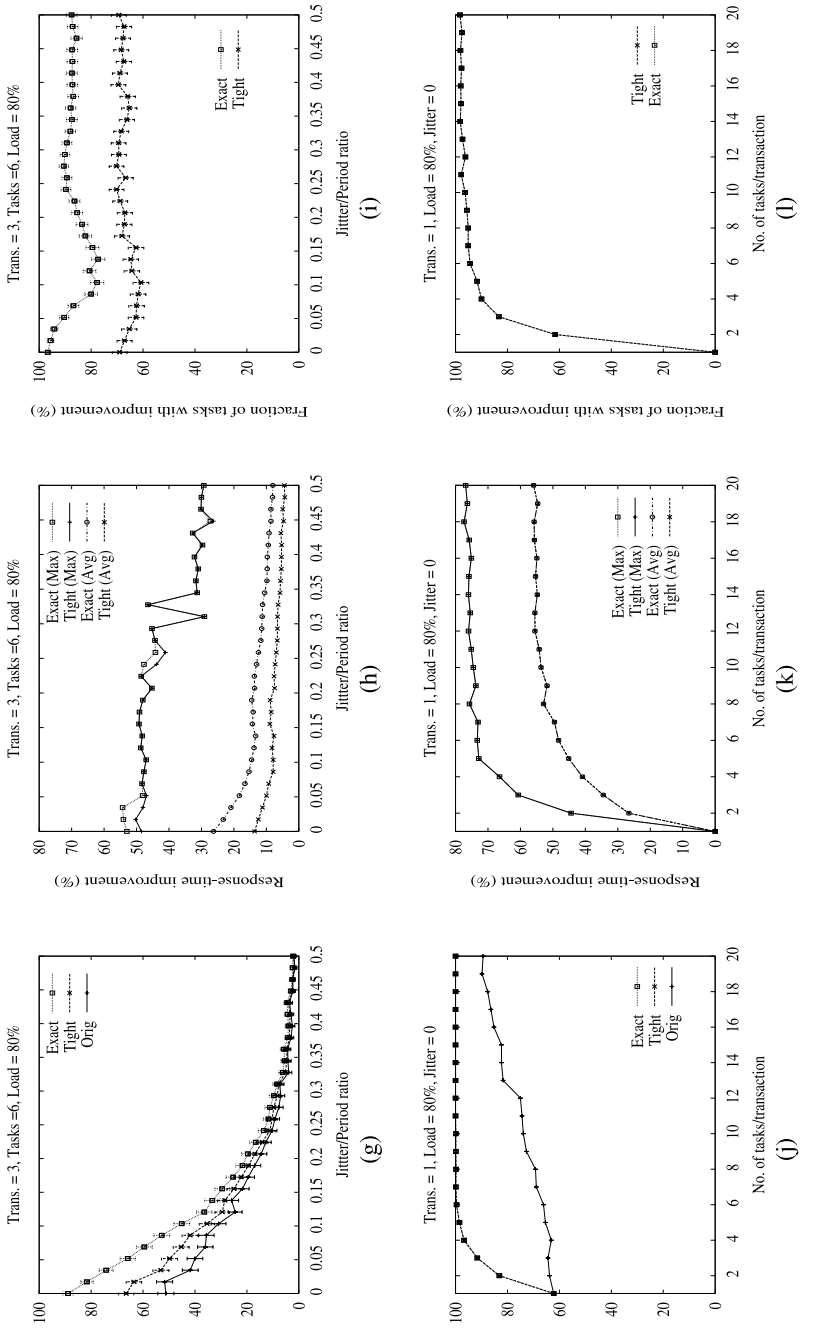


Fig. 16 (*continued*)

Figures 16(a–i) corresponds to a base configuration where the number of tasks per transaction is 6, the number of transactions is 3, system load is 80% and the load of task under admission control is 2%.[5] From this base configuration we vary the number of tasks/transaction (a–c), number of transactions (d–f), jitter (g–i), while keeping the other parameters constant.

Figures 16(a–c) shows the results when the number of tasks is varied between 1 and 13. For more than 5 tasks we can see in (a), that the admission probability for (Tight) is around 12% higher than for (Orig). In (b) we see that the average response time improvement of (Tight) is for 10 tasks over 15%, and that there are task sets (although rare) where improvement of more than 50% can be obtained. In (c) we see that when the number of tasks/transaction grows, so does the probability of a response time improvement.

For Figs. 16(d–f), where the number of transactions is varied, a quite different picture emerges. The difference between (Orig) and (Tight) gets smaller as the number of transactions grows. This is not surprising, since in the case where the tasks/transaction ratio approaches 1, there are very few offset relations among tasks and the analysis approaches the analysis for tasks without offsets.

Figures 16(g–i) show what happens when jitter is varied. Not only does the admission probability decrease drastically, but also the relative improvement of (Tight) over (Orig). This is mainly due to the fact that jitter contributes to I_{ijc}^{Set1}, whereas our improvement only affects $I_{ijc}^{Set2}(t)$. As I_{ijc}^{Set1} accounts for an increasingly larger fraction of the total response-time, the relative improvement of (Tight) decreases. However, the absolute response-time improvement (not shown) and the number of improvements (figure (i)) is not noticeably affected by the jitter. As the jitter grows larger than the period (or larger than several periods) the effects of our improvements diminish further. For system with such large jitters (such as multimedia applications), other methods (Palencia Gutiérrez and González Harbour 1999; Redell 2003) to reduce the estimated response-time can be combined with our presented method.

Finally, Figs. 16(j–l) correspond to a configuration where the number of transactions is 1, system load 80%, and load of the task under admission control is 2%. This type of scenario would occur in a system using a hybrid scheduling method, supporting both static cyclic scheduled tasks (corresponding to the single high priority transaction) and priority scheduled tasks running in the background of the static schedule (Mäki-Turja et al. 2005). This situation shows where our method excels. All tasks have offset relations among them, resulting in well over 30% better admission probability (4–9 tasks) over (Orig) and an average improvement of over 50% when the number of tasks/transaction is more than 8. Another interesting thing is that (Exact) and (Tight) always yield exact response-times. This comes from the fact that when considering a transaction in isolation (no interference among several transactions) the slanted stair interference function captures the worst case interference exactly.

[5] The execution time (C) is randomly generated and the deadline for the task under admission control is set to C/utilization.

7.2 Evaluating implementation efficiency

In order to evaluate and quantify the efficiency (with respect to execution time of RTA) of the method presented in Sect. 6, we have implemented a set of approximate response-time techniques, using the complete set of response-time equations of Appendix A. We compare five RTA methods:

- *fast-tight*, presented in this paper, is the method that is optimized the furthest 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 Sect. 6.5). 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 Sect. 5. It is only optimized towards tightness. These three first methods all produce the same tight response times.
- *orig*, presented by Palencia Gutiérrez and González Harbour (1998) and outlined in Sect. 3, which is not optimized either for tightness nor for analysis speed. It is included in the evaluation a baseline to compare to an existing technique and to see if the relative performance degradation of *tight*, compared to *orig*, remains in *fast-tight* when compared to *fast-orig*.
- *fast-orig*, a speed-up method of *orig* presented in (Mäki-Turja and Nolin 2004). 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*.

7.2.1 Description of simulation setup

In our simulations we generate task sets, by the task generator presented in Sect. 7.1.1, that are used as input to the different RTA implementations. The generated task sets have the following different characteristics compared to Sect. 7.1.1:

- Total system load is 90%.
- The number of transactions is 10.
- Jitter (J_{ij}) for each task is 20% of its transaction period.

The execution time for performing the RTA in Sect. 7.2.2 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. 6.4 and 6.5.

7.2.2 Simulation results

Figure 17(a) shows how the execution time of the five (although the 3 fast methods are indistinguishable) RTA analysis varies with varying tasks/transaction. 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.

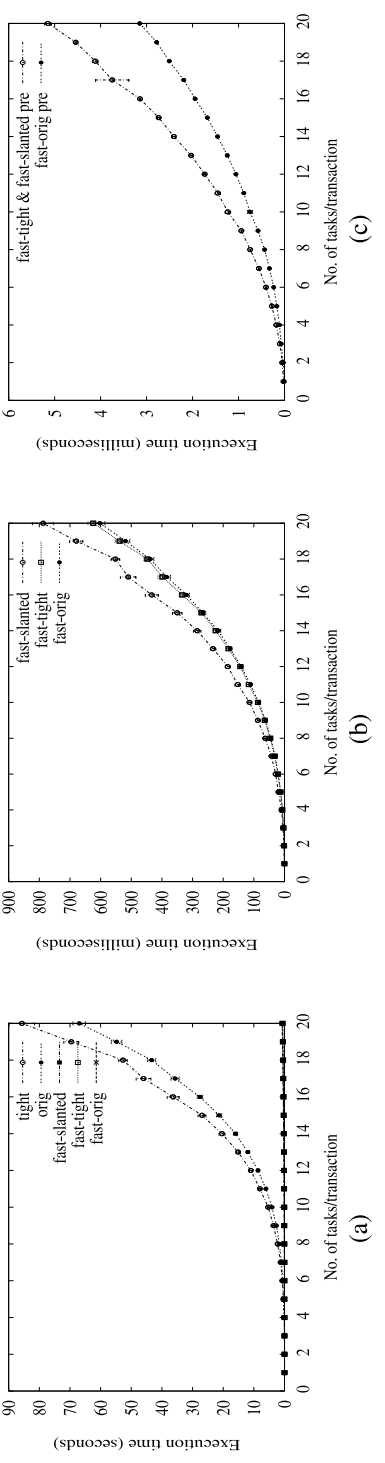


Fig. 17 Results when evaluating analysis speed

Note also that, *tight* has a slight penalty to pay, compared to *orig*, due to more accurate interference modeling.

Zooming in on the three fast analysis methods in Fig. 17(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. 17(b) we see that *fast-tight* has consistently slightly longer execution time.

In Fig. 17(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. 17(a) and 17(b) that the main contribution of speeding up the response times comes from static representation and lookup, but that reverting back to a stepped stair function gives an additional speedup of just above 20%.

In Fig. 17(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* are 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. 17(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 instance 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.

These evaluations show that combining our presented fast and tight methods for response time analysis, one gets the better of two worlds; a response time analysis method that is both fast and tight, significantly outperforming previous methods.

8 Conclusions

We have in this paper presented two improvements to existing methods to calculate approximate response times for tasks with offsets. One provides tighter response times alleviating the problem of too pessimistic results. The second improvement introduces a lookup technique that reduces the calculations done at each fix-point iteration step which considerably shortens the time to calculate the response times. We have also combined these two improvements in one single fast-and-tight RTA method.

For the tight RTA method we prove that it never calculates greater response-times than the method in (Palencia Gutiérrez and González Harbour 1998). Furthermore, we prove that our method never underestimates the interference caused by higher priority tasks. Hence, it calculates a safe and tight approximation of the actual worst-case response-time.

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

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

The main effort in performing RTA for tasks with offsets is to calculate how higher priority tasks interfere with a task under analysis. The key 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. In a simulation study we see that our combined fast-and-tight analysis gives speedups of over two orders of magnitude 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.

Furthermore, our improvements are orthogonal and complementary to other proposed extensions to the original offset analysis such as (Palencia Gutiérrez and González Harbour 1999; Redell 2003) which means they can easily be incorporated into eachother. Also, RTA for hybrid scheduled EDF and FPS systems (Palencia and González Harbour 2003a) uses heavy calculations of task interference (using similar interference functions as in this paper) and should benefit from the methods described in this paper to speed up the analysis.

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.

Appendix A: Complete set of RTA formulae

In this appendix we provide the complete set of formulae to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutiérrez and González Harbour (1998).

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 (3) in this paper):

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

((26) in Palencia Gutiérrez and González Harbour 1998) where the phase between task τ_{ij} and the candidate critical instant task τ_{ic} is defined as:

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

((17) in Palencia Gutiérrez and González Harbour 1998).

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

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

((27) in Palencia Gutiérrez and González Harbour 1998).

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):

$$L_{uac} = B_{ua} + (p - p_{0,uac} + 1)C_{ua} + W_{uc}(\tau_{ua}, L_{uac}) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, L_{uac})$$

((30) in Palencia Gutiérrez and González Harbour 1998) 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:

$$p_{0,uac} = -\left\lfloor \frac{J_{ua} + \Phi_{uac}}{T_u} \right\rfloor + 1$$

((29) in Palencia Gutiérrez and González Harbour 1998) and

$$p_{L,uac} = \left\lceil \frac{L_{uac} - \Phi_{uac}}{T_u} \right\rceil$$

((31) in Palencia Gutiérrez and González Harbour 1998).

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

$$w_{uac}(p) = B_{ua} + (p - p_{0,uac} + 1)C_{ua} + W_{uc}(\tau_{ua}, w_{uac}(p) + \sum_{\forall i \neq u} W_i^*(\tau_{ua}, w_{uac}(p))$$

((28) in Palencia Gutiérrez and González Harbour 1998).

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

$$R_{uac}(p) = w_{uac}(p) - \Phi_{uac} - (p-1)T_u + O_{ua}$$

((32) in Palencia Gutiérrez and González Harbour 1998).

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

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

((33) in Palencia Gutiérrez and González Harbour 1998).

Appendix B: Proofs of theorems

In this appendix we provide proofs of Theorems 1, 6, 7 and 8. We will perform all proofs by algebraic manipulation and use braces to highlight the expression that is manipulated in each step. We also annotate braces with the equations, properties, lemmas, or assumptions referred to when performing some manipulations.

When performing the manipulations we will, e.g., rely on the following properties:

(max) The $\max_v$ operator allows terms that are constant with respect to the maximization variable (v) to be moved outside the maximization operation:

$$\max_v(X_v + Y) = \max_v(X_v) + Y.$$

(sum) Summation over a set of terms can be divided into two separate summations:

$$\sum_v (X_v + Y_v) = \sum_v X_v + \sum_v Y_v$$

(ceil) When taking the ceiling ($\lceil\ \rceil$) of a set of terms, terms that are known to be integers can be moved outside of the ceiling/floor expression:

$$X \in \mathbb{Z} \Rightarrow \lceil X + Y \rceil = X + \lceil Y \rceil$$

(floor) When taking the floor ($\lfloor\ \rfloor$) of a set of terms, terms that are known to be integers can be moved outside of the ceiling expression:

$$X \in \mathbb{Z} \Rightarrow \lfloor Y + X \rfloor = X + \lfloor Y \rfloor$$

(mod) Adding a positive term subjected to a modulus Y operation, $X \bmod Y$, with is negative counterpart, $(-X) \bmod Y$, will result in Y:

$$X \bmod Y + (-X) \bmod Y = Y$$

Theorem 2 *$W_{ic}(\tau_{ua}, t)$ using definition of Φ_{ijc} according to* (1) *is algebraically equivalent to using* (17) *in* (Palencia Gutiérrez and González Harbour 1998).

Proof Let $x = O_{ij}$ and $y = J_{ic} + O_{ic}$. With the definitions of x and y the two different definitions of Φ_{ijc} (denoted Φ^{17}_{ijc} and Φ^{3}_{ijc} respectively) can be reformulated as:

$$\Phi^{3}_{ijc} = (x - y) \bmod T_i$$

(reformulation of (1)),

$$\Phi^{17}_{ijc} = T_i - (y - x) \bmod T_i,$$

(reformulation of (17) in Palencia Gutiérrez and González Harbour 1998). We divide the proof into two cases, $(x - y) \bmod T_i \neq 0$ and $(x - y) \bmod T_i = 0$: $(x - y) \bmod T_i \neq 0$: Assume:

$$\begin{aligned} & \underbrace{(x - y) \bmod T_i = k \text{ and } (y - x) \bmod T_i = k'}_{\text{(mod)}} \\ \Rightarrow \quad & k + k' = T_i \\ \Rightarrow \quad & \underbrace{T_i - k' = k}_{\text{def. of } \Phi^{17}_{ijc}} \\ \Rightarrow \quad & T_i - (y - x) \bmod T_i = k. \end{aligned}$$

Thus the two definitions of Φ_{ijc} are equivalent. $(x - y) \bmod T_i = 0$: Note that Φ_{ijc} with the two different definitions are

$$\begin{aligned} & \Phi^{3}_{ijc} = 0 \quad \text{by definition,} \\ & \Phi^{17}_{ijc} = T_i \quad \text{since } \underbrace{(x - y) \bmod T_i = 0}_{\text{(mod)}} \Rightarrow T_i - 0 = (y - x) \bmod T_i. \end{aligned}$$

So using the two values of Φ_{ijc} in $W_{ic}(\tau_{ua}, t)$ (see (26) in Palencia Gutiérrez and González Harbour 1998) it suffices to investigate:

$$\underbrace{\left\lfloor \frac{J_{ij} + T_i}{T_i} \right\rfloor} + \underbrace{\left\lceil \frac{t - T_i}{T_i} \right\rceil} \stackrel{?}{=} \left\lfloor \frac{J_{ij} + 0}{T_i} \right\rfloor + \left\lceil \frac{t - 0}{T_i} \right\rceil,$$

$$\underbrace{\left\lfloor \frac{J_{ij}}{T_i} + \frac{T_i}{T_i} \right\rfloor}_{\text{(floor)}} + \underbrace{\left\lceil \frac{t}{T_i} - \frac{T_i}{T_i} \right\rceil}_{\text{(ceil)}} \stackrel{?}{=} \left\lfloor \frac{J_{ij}}{T_i} \right\rfloor + \left\lceil \frac{t}{T_i} \right\rceil,$$

$$\underbrace{\left\lfloor \frac{J_{ij}}{T_i} \right\rfloor + 1 + \left\lceil \frac{t}{T_i} \right\rceil - 1}_{} \stackrel{?}{=} \left\lfloor \frac{J_{ij}}{T_i} \right\rfloor + \left\lceil \frac{t}{T_i} \right\rceil,$$

$$\left\lfloor \frac{J_{ij}}{T_i} \right\rfloor + \left\lceil \frac{t}{T_i} \right\rceil = \left\lfloor \frac{J_{ij}}{T_i} \right\rfloor + \left\lceil \frac{t}{T_i} \right\rceil. \qquad \square$$

Theorem 6 *$W_i^*(\tau_{ua}, t)$ as defined in (4) and $W_i^*(\tau_{ua}, t)$ as defined in (9) are equivalent.*

Proof

$$\underbrace{W_i^*(\tau_{ua}, t)}_{(9)} =$$

$$J_i^{ind}(\tau_{ua}) + \underbrace{T_i^{ind}(\tau_{ua}, t)}_{(11)} =$$

$$J_i^{ind}(\tau_{ua}) + \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^+(\tau_{ua}, t)}_{(12)} =$$

$$\underbrace{J_i^{ind}(\tau_{ua}) + \max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t)) - J_i^{ind}(\tau_{ua}) \right)}_{(\max)} =$$

$$\underbrace{J_i^{ind}(\tau_{ua})}_{} + \max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t)) \right) - \underbrace{J_i^{ind}(\tau_{ua})}_{} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t))}_{(3)} =$$

$$\underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t)}_{(4)} = W_i^*(\tau_{ua}, t). \qquad \square$$

Before proving Theorem 7 we need to establish some lemmas.

Lemma 1 *Regardless of candidate critical instant c: $I_{ijc}^{Set2}(T_i) = C_{ij}$.*

Proof

$$\underbrace{I_{ijc}^{Set2}(T_i)}_{(2)}$$

$$(1) \;=\; \underbrace{\left\lceil \frac{T_i - \Phi_{ijc}}{T_i} \right\rceil}_{0 \le \Phi_{ijc} < T_i \text{ (see (1))}} \quad C_{ij}$$

$$(2) \quad = \underbrace{1 C_{ij}}$$

$$(3) \quad = C_{ij}. \qquad \square$$

Lemma 2 *Assume* $t = k * T_i + t'$ *(where* $k \in \mathbb{N}$ *and* $0 \le t' < T_i$*), then* $I_{ijc}^{Set2}(t) = k * I_{ijc}^{Set2}(T_i) + I_{ijc}^{Set2}(t')$.

Proof

$$\underbrace{I_{ijc}^{Set2}(t)}_{(2)} \overset{?}{=} k * \underbrace{I_{ijc}^{Set2}(T_i)}_{\text{Lemma 1}} + \underbrace{I_{ijc}^{Set2}(t')}_{(2)}$$

$$(1) \quad \underbrace{\left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil}_{\text{Assumption}} C_{ij} \overset{?}{=} k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

$$(2) \quad \underbrace{\left\lceil \frac{k * T_i + t' - \Phi_{ijc}}{T_i} \right\rceil}_{} C_{ij} \overset{?}{=} k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

$$(3) \quad \underbrace{\left\lceil \frac{k * T_i}{T_i} + \frac{t' - \Phi_{ijc}}{T_i} \right\rceil}_{\text{(ceil)} \, \wedge k \in \mathbb{N}} C_{ij} \overset{?}{=} k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

$$(4) \quad \underbrace{\left(k + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil\right) C_{ij}}_{} \overset{?}{=} k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}$$

$$(5) \quad k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij} = k C_{ij} + \left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil C_{ij}. \qquad \square$$

Lemma 3 $T_i^{ind}(\tau_{ua}, T_i) = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}$.

Proof

$$\underbrace{T_i^{ind}(\tau_{ua}, T_i)}_{(11)}$$

$$(1) \quad = \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^{+}(\tau_{ua}, T_i)}_{(12)}$$

$$(2) \quad = \max_{\forall c \in hp_i(\tau_{ua})} \Bigg(\underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(T_i))}_{\text{(sum)}} - J_i^{ind}(\tau_{ua}) \Bigg)$$

$$
\begin{aligned}
(3) &= \max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} + \sum_{\forall j \in hp_i(\tau_{ua})} \underbrace{I_{ijc}^{Set2}(T_i)}_{\text{Lemma 1}} - J_i^{ind}(\tau_{ua}) \right) \\
(4) &= \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} + \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} - J_i^{ind}(\tau_{ua}) \right)}_{\text{(max)}} \\
(5) &= \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} - J_i^{ind}(\tau_{ua}) \right)}_{\text{(max)}} \\
(6) &= \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1}}_{(10)} - J_i^{ind}(\tau_{ua}) \\
(7) &= \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{J_i^{ind}(\tau_{ua}) - J_i^{ind}(\tau_{ua})} \\
(8) &= \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}.
\end{aligned}
$$

□

Theorem 7 *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

$$
\begin{aligned}
&\underbrace{T_i^{ind}(\tau_{ua}, t)}_{(11)} \\
(1) &= \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^{+}(\tau_{ua}, t)}_{(12)} \\
(2) &= \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + \underbrace{I_{ijc}^{Set2}(t)}_{\text{Lemma 2}}) - J_i^{ind}(\tau_{ua}) \\
(3) &= \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + k * \underbrace{I_{ijc}^{Set2}(T_i)}_{\text{Lemma 1}} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua}) \\
(4) &= \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + kC_{ij} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua})}_{\text{(sum)}}
\end{aligned}
$$

$$(5) = \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij} + \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua}) \right)}_{(\max)}$$

$$(6) = \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} kC_{ij}}_{} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua})$$

$$(7) = k * \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}}_{\text{Lemma 3}} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua})$$

$$(8) = k * T_i^{ind}(\tau_{ua}, T_i) + \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} (I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')) - J_i^{ind}(\tau_{ua})}_{(12)}$$

$$(9) = k * T_i^{ind}(\tau_{ua}, T_i) + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua}, t')}_{(11)}$$

$$(10) = k * T_i^{ind}(\tau_{ua}, T_i) + T_i^{ind}(\tau_{ua}, t').$$

□

In proving Theorem 8 we will use the definition of $w_{uac}(p)$ ((28) in Palencia Gutiérrez and González Harbour 1998, see Appendix A), 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, $f(t)$:

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

where K_1 is some constant value. We note that a solution to (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 4 *The smallest solution to* (28′), *denoted* s, *cannot exist where* $f(t)$ *has a derivative greater than or equal to* 1 (*i.e. where* $f'(t) \geq 1$).

Proof From (Sjödin and Hansson 1998) 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. 18, the line $y = f(p)$ cannot be converging with line $y = p$ (which has a derivative of 1).

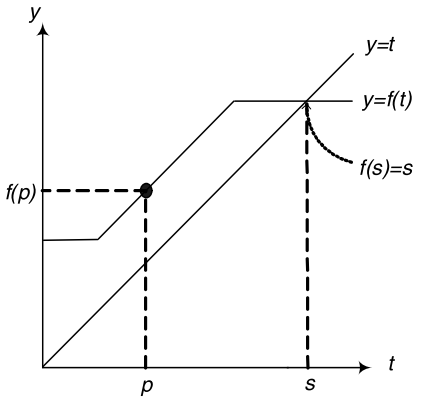


Fig. 18 Fix-Point Iteration when $f'(t) \geq 1$

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 8 *Equation* (28′) *in Appendix B, cannot have a solution at a time t where any approximate interference function has a derivative greater than or equal to one.*

Proof 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 4, the theorem holds. □

References

Audsley NC, Burns A, Tindell K, Richardson MF, Wellings AJ (1993) Applying new scheduling theory to static priority pre-emptive scheduling. Softw Eng J 8(5):284–292

Audsley NC, Burns A, Davis RI, Tindell K, Wellings AJ (1995) Fixed priority pre-emptive scheduling: an historical perspective. Real-Time Syst 8(2–3):173–198

Fersman E (2003) A generic approach to schedulability analysis of real-time systems. PhD thesis, Uppsala University

Joseph M, Pandya P (1986) Finding response times in a real-time system. Comput J 29(5):390–395

Liu C, Layland J (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM 20(1):46–61

Mäki-Turja J, Nolin M (2004) Faster response time analysis of tasks with offsets. In: Proceedings of the 10th IEEE real-time technology and applications symposium (RTAS), May 2004

Mäki-Turja J, Hänninen K, Nolin M (2005) Efficient development of real-time systems using hybrid scheduling. In: International conference on embedded systems and applications (ESA), June 2005

Palencia Gutiérrez JC, González Harbour M (1998) Schedulability analysis for tasks with static and dynamic offsets. In: Proceedings of the 19th IEEE real-time systems symposium (RTSS), December 1998

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

Palencia Gutiérrez JC, González Harbour M (1999) Exploiting precedence relations in the schedulability analysis of distributed real-time systems. In: Proceedings of the 20th IEEE real-time systems symposium (RTSS), December 1999, pp 328–339

Palencia JC, González Harbour M (2003a) Offset-based response time analysis of distributed systems scheduled under EDF. In: Proceedings of the 15th Euromicro conference on teal-time systems, June 2003

Palencia Gutiérrez JC, González Harbour M (2003b) Response time analysis for tasks scheduled under EDF within fixed priorities. In: Proceedings of the 24th IEEE real-time systems symposium (RTSS), December 2003

Pop T, Eles P, Peng Z (2003) Schedulability analysis for distributed heterogeneous time/event triggered real-time systems In: Proceedings of the 15th Euromicro conference on real-time systems, July 2003

Rahni A, Traore K, Grolleau E, Richard M (2007) Comparison of two worst-case response time analysis methods for real-time transactions. In: Junior researchers workshop on real-time computing, March 2007

Redell O (2003) 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

Regher J, Reid A, Webb K, Parker M, Lepreau J (2003) Evolving real-time systems using hierarchical scheduling and concurrency analysis. In: Proceedings of the 24th IEEE real-time systems symposium (RTSS), December 2003. IEEE Computer Society, Los Alamitos

Sha L, Abdelzaher T, Årzén K-E, Cervin A, Baker TP, Burns A, Buttazzo G, Caccamo M, Lehoczky JP, Mok AK (2004) Real time scheduling theory: a historical perspective. Real-Time Syst 28(2/3):101–155

Sjödin M, Hansson H (1998) Improved response-time calculations. In: Proceedings of the 19th IEEE real-time systems symposium (RTSS), December 1998. http://www.docs.uu.se/~mic/papers.html

Tindell K (1992) Using offset information to analyse static priority pre-emptively scheduled task sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England

Jukka Mäki-Turja is senior lecturer and researcher at Mälardalen Real-Time Research Centre. His research interest lies in design and analysis of predictable real-time systems. Jukka received his PhD in computer science from Mälardalen University in 2005 with response time analysis for tasks with offsets as focus.

Mikael Nolin is a professor in real-time system at Mälardalen Real-Time Research Centre. Since 2002 he has been active at MRTC, focusing his research towards techniques and tools for development of predictable distributed embedded control systems. He also holds an industrial position at the company CC Systems, where he provides expertise in development of software different vehicle applications and systems. Mikael received his PhD and MSc in computer science from Uppsala University in 2000 and 1995 respectively.