Efficient Response-Time Analysis for Tasks with Offsets

Jukka Mäki-Turja Mikael Nolin
Mälardalen Real-Time Research Centre (MRTC)
Västerås, Sweden

E-mail: jukka.maki-turja@mdh.se

Abstract

We present a method that enables an efficient implementation of the approximative response-time analysis (RTA) for tasks with offsets presented by Tindell [13] and Palencia Gutierrez et al. [8].

The method allows for significantly faster implementations of schedulability tools using RTA. Furthermore, reducing computation time, from tens of milliseconds to just a fraction of a millisecond, as we will show, is a step towards on-line RTA in for example admission control systems.

We formally prove that our reformulation of earlier presented equations is correct and allow us to statically represent parts of the equation, reducing the calculations during fix-point iteration. We show by simulations that the speed-up when using our method is substantial. When task sets grow beyond a trivial number of tasks and/or transactions a speed-up of more than 100 times (10 transactions and 10 tasks/transaction) compared to the original analysis can be obtained.

1 Introduction

A powerful and well established schedulability analysis technique is the *Response-Time Analysis* (RTA) [1]. RTA is applicable to systems where tasks are scheduled in strict priority order which is the predominant scheduling technique used in real-time operating systems today. In this paper, we present a method that enables an efficient implementation of the approximative RTA for tasks with offsets presented by Tindell [13] and Palencia Gutierrez *et al.* [8].

RTA is a method to calculate worst-case response-times for tasks in hard real-time systems. In essence RTA is used to perform a shedulability test, i.e., checking wheter or not tasks in the system will satisfy their deadlines. Traditionally, industrial use of schedulability tests has been limited. However, with recent advancements in software development and synthesis tools, such as UML-based tools [2, 10, 12], schedulability tests can be integrated in the normal workflow and tool-chains used by real-time engineers.

This kind of tools can be used, for instance, to perform automatic allocation of tasks to nodes in a distributed real-time system or to automatically derive task priorities (priority assignment) so that task deadlines are guaranteed to be met. To be able perform such allocation and/or assignment tasks, tools need to be able to perform schedulability tests. Typically, such automatic allocation/assignment methods are based on optimisation or search techniques, during which numerous possible configurations are evaluated. (There can easily be tens or hundreds of thousands of possible configurations even for small systems.) For each configuration a schedulability test is performed in order to evaluate different solutions. Hence, schedulability tests must be fast in order to be suitable for such systems.

Dynamic real-time systems, with on-line admission control of real-time tasks, needs to be able to quickly evaluate whether a dynamically arriving task can be admitted to the system. In these cases the tolerance for delays in the scheduling analysis is even less than in the case of software engineering tools.

Accounting for offsets between tasks gives significantly tighter analysis results than using the traditional notion of a critical instant where all tasks in the system are considered to be released simultaneously [4]. Hence, tools for automatic configuration (as well as on-line schedulability tests) would benefit from using this extension; it becomes easier to find feasible configurations. In fact, many systems that will be deemed infeasible by RTA without offsets will be feasible when taking offsets into account. However, the price of taking offsets into account is increased execution time of the analysis. Existing methods for RTA with offsets have all been focused on modelling capabilities while ignoring issues of computational complexity, e.g., [8, 9, 11, 13].

The first RTA for tasks with offsets was presented by Tindell [13]. He provided an exact algorithm for calculating response time for tasks with offsets. However, this algorithm becomes computationally intractable for anything but small task sets due to its exponential time complexity. In order to deal with this problem, Tindell also provided an approximation algorithm, polynomial in time, which gives

pessimistic but safe (worst case response time is never underestimated) results. Later, Palencia Gutierrez *et al.* [8] formalised, generalised and improved Tindell's work.

In this paper we present a method that enables an efficient implementation of the approximative offset analysis given by Tindell [13] and Palencia Gutierrez *et al.* [8]. The correctness of our method is formally proven by demonstrating algebraic equivalence with the original methods. The method significantly speeds up the calculation of response times, as we will show by simulations.

Paper Outline: In Sect. 2 we revisit and restate the original offset RTA [8, 13]. In Sect. 3 we present our new method. Section 4 presents evaluations of our method, and finally, Sect. 5 concludes the paper and outlines future work.

2 Existing offset RTA

This section revisits the existing response-time analysis for tasks with offsets [8, 13] and illustrates the intuition behind the analysis and the formulas.

2.1 System model

The system model used is as follows: The system, Γ, consists of a set of k transactions $\Gamma_1, \ldots, \Gamma_k$. Each transaction Γ_i is activated by a (periodic) sequence of events with period T_i (for non-periodic events T_i denotes the minimum interarrival time between two consecutive events). The activating events are mutually independent, i.e., phasing between them is arbitrary. A transaction, Γ_i, contains $|\Gamma_i|$ tasks, and each task is activated (released for execution) when a time, *offset*, has elapsed 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 are allowed to be both smaller or greater than the period. Parameters for an example transaction (Γ_i) with two tasks (τ_{ia}, τ_{ib}) is visualised in Fig. 1. The offset denotes the earliest release time of a task relative to the start of its transaction and jitter denotes the variability in the release of the task. (In Fig. 1 the jitter is not graphically visualised.)

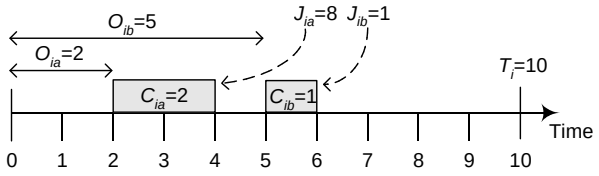

Figure 1. An example transaction Γ_i

2.2 Response-time analysis

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

In the classical RTA (without offsets) the *critical instant* for τ_{ua} is when it is released at the same time as all higher (or equal) priority tasks [3, 4]. In a task model with offsets this assumption yields pessimistic response-times since some tasks can not be released simultaneously due to offset relations. Therefore, Tindell [13] relaxed the notion of critical instant to be:

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

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

Tindell's exact RTA tries every possible combination of candidates among all transactions in the system. This, however, becomes computationally intractable for anything but small task sets (the number of possible combinations of candidates is m^n for a system with n transactions and with m tasks per transaction). Therefore Tindell provided an approximative RTA that still gives good results but uses one single approximation function for each transaction. Palencia Gutierrez *et al.* [8] formalised and generalised Tindells work. We will in this paper use the more general formalism of Palencia Gutierrez *et al.*, although our proposed method is equally applicable to Tindell's original algorithm.

2.3 Interference function

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

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

$Set2$ Activations that occur after the critical instant

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

RTA of tasks with offsets is based on two fundamental theorems [8, 13]:

1. The worst case interference a task τ_{ij} imposes on τ_{ua} is when $Set1$ activations are delayed by an amount of jitter such that they all occur at the critical instant and the activations in $Set2$ have zero jitter.

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

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

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

From the second theorem we get that τ_{ic} will coincide with the critical instant after having experienced its worst case jitter delay (i.e., the critical instant will occur at $(O_{ic} + J_{ic}) \mod T_i$, relative to the start of Γ_i). This implies that the first instance of a task τ_{ij} in $Set2$ will be released at Φ_{ijc} time units after the critical instant, and subsequent releases will occur periodically every T_i.

Figure 2 illustrates the four different Φ_{ijc}-s that are possible for our example transaction in Fig. 1. The upward arrows denote task releases (the height of the corresponding arrow denotes amount of execution released, i.e., C_{ia} and C_{ib} respectively). Figure 2(a) depicts the situation when τ_{ia} acts as the candidate critical instant. Shown is the phasing between τ_{ia} (2) and τ_{ib} (5) for this situation. Furthermore, Fig. 2(a) also shows activations for each task in the transaction. Task instances belonging to $Set1$ are released at time 0, and the first instance belonging to $Set2$ is also depicted (subsequent activation occur periodically). Figure 2(b) shows the corresponding situation if τ_{ib} happens 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 imposed by task τ_{ij} can be divided into two parts:

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

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

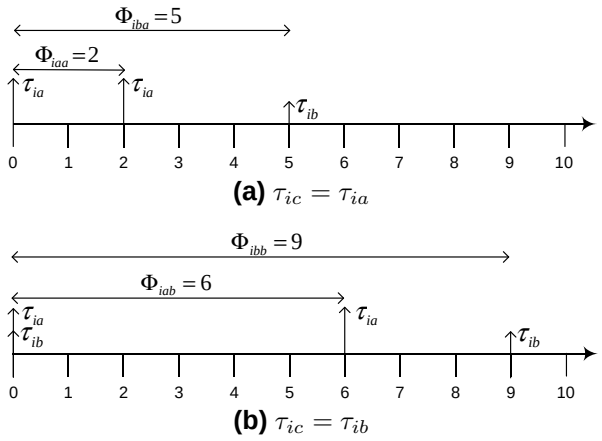

Figure 2. Φ-s for the two candidates in Γ_i

These are defined as follows:

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

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

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

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

2.4 Approximation function

Since we beforehand cannot know which task in each transaction coincides with the critical instant, the exact analysis tries every possible combination [8, 13]. However, since this is computationally intractable for anything but small task sets the approximative analysis, presented in [8, 13], defines one single, upward approximated, function for the interference caused by transaction Γ_i:

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

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

As an example consider again transaction Γ_i depicted in Fig. 1. Figure 3 shows the interference function for the two candidates (W_{ia} and W_{ib}), and it shows how W_i^* is derived from them by taking the maximum of the two functions at every t.

Given the interference (W_i^*) each transaction imposes on the task under analysis (τ_{ua}), during a time interval of

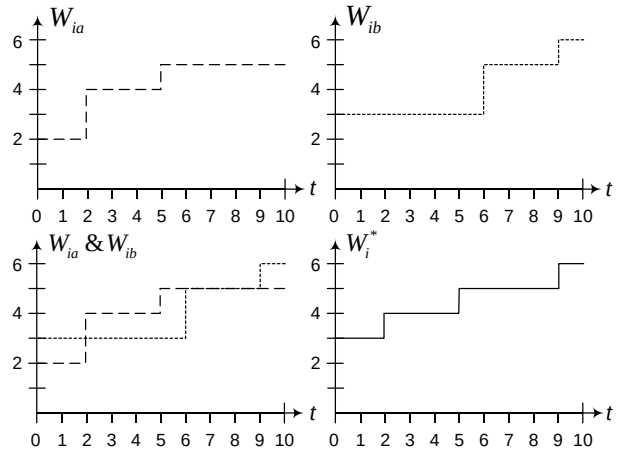

Figure 3. $W_{ic}(\tau_{ua}, t)$ **and** $W_i^*(\tau_{ua}, t)$ **functions**

length t, its response time (R_{ua}) can be calculated. Appendix A shows how to perform these response-time calculations.

3 Fast offset RTA

When calculating response times, the function $W_i^*(\tau_{ua}, t)$ (equation 4 on the preceding page) 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 pattern that is repeated every T_i time units (see theorem 2 in this section), 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 $W_i^*(\tau_{ua}, t)$ changes using such precomputed information and how to calculate and store that information.

3.1 Approximation function with lookup

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

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

We redefine Eq. 4 using our new notation as:

$$W_i^*(\tau_{ua}, t) = J_i^{ind}(\tau_{ua}) + T_i^{ind}(\tau_{ua}, t) \tag{5}$$

This partitioning of $W_i^*(\tau_{ua}, t)$ is visualised in Fig. 4. $J_i^{ind}(\tau_{ua})$ is the maximum starting value of each of the $W_{ic}(\tau_{ua}, t)$ functions (i.e. max of $W_{ic}(\tau_{ua}, 0)$, see Eq. 3) which is calculated by:

$$J_i^{ind}(\tau_{ua}) = \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} \tag{6}$$

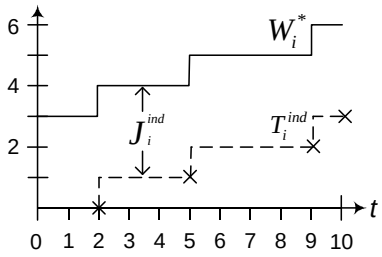

Figure 4. $W_i^*(\tau_{ua}, t)$**,** $J_i^{ind}(\tau_{ua})$**, and** $T_i^{ind}(\tau_{ua}, t)$

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

$$T_i^{ind}(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua}, t) \tag{7}$$

where

$$W_{ic}^{+}(\tau_{ua}, t) = \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t) \right) - J_i^{ind}(\tau_{ua}) \tag{8}$$

The correctness of our method requires that our new definition of $W_i^*(\tau_{ua}, t)$ in Eq. 5 is functionally equivalent to the definition in Eq. 4.

Theorem 1 *$W_i^*(\tau_{ua}, t)$ as defined in Eq. 4 and $W_i^*(\tau_{ua}, t)$ as defined in Eq. 5 are equivalent.*

Proof 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 it is enough to store information for the first T_i time units:

Theorem 2 *Assume $t = k * T_i + t'$ (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 The theorem is proved by algebraic equivalence in Appendix B.

We represent $T_i^{ind}(\tau_{ua}, t)$ for the first T_i time units using the concave corners of the function $T_i^{ind}(\tau_{ua}, t)$ (marked with crosses in Fig. 4). The representation uses two arrays T_i^c and T_i^t. $T_i^c[x]$ represents the maximum amount of time induced interference Γ_i will pose on a lower priority task during interval lengths up to $T_i^t[x]$ ($x \in 1 \ldots |T_i^c|$). Using these two arrays we redefine $T_i^{ind}(\tau_{ua}, t)$ as follows:

$$\begin{aligned} T_i^{ind}(\tau_{ua}, t) =& k * T_i^c[|T_i^c|] + T_i^c[x] \\ k =& t \text{ div } T_i \\ t' =& t \text{ rem } T_i \\ x =& \min\{y \ : \ t' \leq T_i^t[y]\} \end{aligned} \tag{9}$$

For our example transaction, the time induced interference (represented in Fig. 4 by crosses) is stored in the arrays T_i^c and T_i^t as follows:

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

Using Eq. 5 and Eq. 9 instead of Eq. 4 to compute $W_i^*(\tau_{ua}, t)$ will significantly reduce the time to compute response times as we will show in Sect. 4.

3.2 Precomputing T_i^c and T_i^t

To compute T_i^c and T_i^t we will first calculate the pattern for each $W_{ic}^+(\tau_{ua}, t)$ from which we will later extract the maximum. Hence, we have to consider each task τ_{ic} in Γ_i as a candidate to coincide with the critical instant. For each candidate task, τ_{ic}, we define a set of points p_{ic}. Each point $p_{ic}[k]$ has an x and a y coordinate, describing how the time induced interference grows over time if the corresponding τ_{ic} coincides with the critical instant. The points in p_{ic} corresponds to the convex corners of $W_{ic}^+(\tau_{ua}, t)$ of Eq. 8. W_{ia}^+ and W_{ib}^+, for our example transaction, are depicted in Fig. 5 and the corresponding p_{ia} and p_{ib} are illustrated by black and white circles respectively.

To calculate the set p_{ic}, we (without loss of generality) assume that tasks are enumerated according to their first activation after the critical instant, i.e., according to Φ_{ijc} values. The following equations define the array p_{ic}:

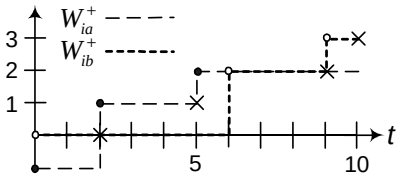

Figure 5. Visual representation of p_{ic} sets

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

Each p_{ic} set represents how the time induced interference grows, for critical instant candidate τ_{ic}, during one period (T_i). For our example transaction of Fig. 1, we get the following two p_{ic}-s (corresponding to the black and white circles in Fig. 5):

$$\begin{array}{lll} p_{ia} & = [\langle 0, -1\rangle, \langle 2, 1\rangle, \langle 5, 2\rangle] & \text{black circles} \\ p_{ib} & = [\langle 0, \ \ 0\rangle, \langle 6, 2\rangle, \langle 9, 3\rangle] & \text{white circles} \end{array}$$

Now, we have the information generated by all $W_{ic}^+(\tau_{ua}, t)$-functions, stored in the p_{ic}-sets. These stepwise functions are represented by one point per step. In order to get a representation of $T_i^{ind}(\tau_{ua}, t)$ in Eq. 7, we extract the points that represents the maximum of all $W_{ic}^+(\tau_{ua}, t)$-s. Thus, we will obtain the convex corners of $T_i^{ind}(\tau_{ua}, t)$.

Next, 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 what points in p_i that corresponds the the convex corners of $T_i^{ind}(\tau_{ua}, t)$, we define the relation *subsumes* that says: 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 6 illustrates the subsumes relation graphically, and the formal definition is:

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

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

$$\text{From } p_i \text{ remove } p_i[b] \text{ if } \exists a \neq b : p_i[a] \succ p_i[b] \tag{10}$$

Now, p_i contains the convex corners of the function $T_i^{ind}(\tau_{ua}, t)$. For our example transaction we now have:

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

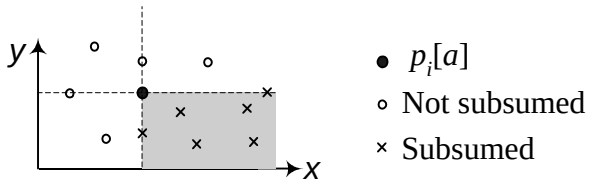

Figure 6. Removing points from p_i

All we have to do now is to find the concave corners (illustrated by crosses in Fig. 5) and store them in the arrays T_i^c and T_i^t. This is done by the following algorithm:

```
for k := 1 to |p_i| do
    T_i^c[k] := p_i[k].y
    if k < |p_i| then
        T_i^t[k] := p_i[k+1].x
    else
        T_i^t[k] := T_i
done
```

For our example transaction this gives the following T_i^c and T_i^t (corresponding to crosses in Fig. 5):

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

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

3.3 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 ($|\Gamma_i|$ points for each candidate task). Thus, storing T_i^c and T_i^t results in a quadratic space complexity since, in the worst-case, no points from p_i will be removed.

The method presented in this paper divides the calculation of W_i^* into a pre-calculation and a fix-point iteration phase. A naive implementation of the removal procedure in Eq. 10 requires comparison of each pair of points; resulting in cubic time-complexity ($O(|\Gamma_i|^3)$) for pre-calculating T_i^c and T_i^t.[1] During the fix-point iteration phase, a binary search through a quadratically sized array is performed (Eq. 9), resulting in $O(\log |\Gamma_i|^2)$ time complexity for calculating W_i^* according to Eq. 5. The original complexity for calculating W_i^* according to Eq. 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 formulas in appendix A). Assume X denotes number of fix-point iterations needed, then the overall complexity for the original approach (Eq. 4) is ($O(X|\Gamma_i|^2)$), whereas our method (Eq. 5 & Eq. 9) yields ($O(|\Gamma_i|^3 + X \log |\Gamma_i|^2)$).

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

4 Evaluation

In order to evaluate the effectiveness of our method we have implemented the response-time equations in appendix A, using both the original defintion of W_i^* from Sect. 2 (Old RTA) and our faster version of W_i^* from Sect. 3 (Fast RTA). Using these implementations and a synthetic task-generator we have performed an evaluation, by simulations, of both approaches by calculatig the response times for all tasks in the system.

4.1 Description of Simulation

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

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

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

- The total system load is proportionally distributed over all transactions in the system.
- Transaction periods (T_i) are randomly distributed in the range 1.000 to 1.000.000 (uniform distribution).
- Each offset (O_{ij}) is randomly distributed within the transaction period (uniform distribution).
- The execution times (C_{ij}) are chosen as a fraction of the time between two consecutive offsets in the transaction. The fraction is the same throughout one transaction. The fraction is selected so that the the transaction load (as defined by the first property) is obtained.
- The jitter (J_{ij}) is randomly distributed between zero and 1.2 times the transaction period ($0..1.2T_i$, uniform distribution).
- Blocking (B_{ij}) is set to zero.
- The priorities are assigned in rate monotonic order [4].

We have measured execution times for performing RTA (for all tasks in the system) using both methods (Old RTA and Fast RTA). The execution times are obtained from a laptop with a Pentium III CPU. For Fast RTA the execution

times include the time to calculate T_i^c and T_i^t. The results in Sect. 4.2 have been obtained by taking the mean values of 50 simulated task-sets for each point in each graph. The 95% confidence intervals are shown for all execution times (although difficult to see due to their small size).

4.2 Simulation Results

Figure 7(a) shows the execution times for Fast RTA and Old RTA when the number of tasks/transaction is varied from 1 to 10 (while keeping the system load at 9/10 (90%) and the number of transactions at 10). When the number of tasks/transaction is 10, the execution time is less than 0.40 seconds for Fast RTA, and about 20 seconds for Old RTA. This amounts to a speedup of 50 times. Similar execution times are obtained both when varying the number of transactions between 1 and 10 and when varying load between 1/10 (10%) and 9/10.

In Fig. 7(b) the complexity of Fast RTA is shown, and by comparison with Fig. 7(a) it can be seen that Fast RTA has a less steep curve than does Old RTA. Also, in Fig. 7(b) the amount of time spent pre-calculating the arrays T_i^c and T_i^t is plotted, and it is apparent that the overhead is negligible. For the larger task sets, about 0.3% of the total time of Fast RTA, is spent on precomputing T_i^c and T_i^t.

Figure 7(c) shows the execution-time of the pre-calculation only. Since we use a $O(N^2 \log N)$ implementation of the pre-calculation the slope is slightly less than what could be expected from a naive implementation ($O(N^3)$).

In Fig. 8 we show the relative execution time of Fast RTA compared to Old RTA, calculated by t_{Fast}/t_{Old}, where t_{Fast} is the execution time for Fast RTA and t_{Old} for Old RTA. The first plot (+) shows how the relative execution time changes when the number of tasks/transaction is varied from 3 to 10. When the number of tasks/transaction is 1 the relative execution time is 0.58 and it rapidly decreases to the values visible in the graph.

The second plot ($\times$) in Fig. 8 illustrates when the number of transactions is varied between 2 and 10. When the number of transactions is 1, the relative execution time is 1.01, which means that Fast RTA is *slower* than Old RTA. When performing RTA for a single transaction, the overhead of precomputing T_i^c and T_i^t outweighs the benefits obtained during the RTA (the pre-computed W_i^* is never used). However, as seen in the plot, when the number of transactions is higher than 1, the overhead is well justified since the total RTA is significantly faster.

The third plot ($*$) in Fig. 8 illustrates when the load is varied between 1/10 (10%) and 9/10 (90%). In this plot we see that the relative execution time is not highly dependent on the system load, only a small decrease in relative execution time is obtained as the system load grows.

In order to compare Old RTA and Fast RTA, in the con-

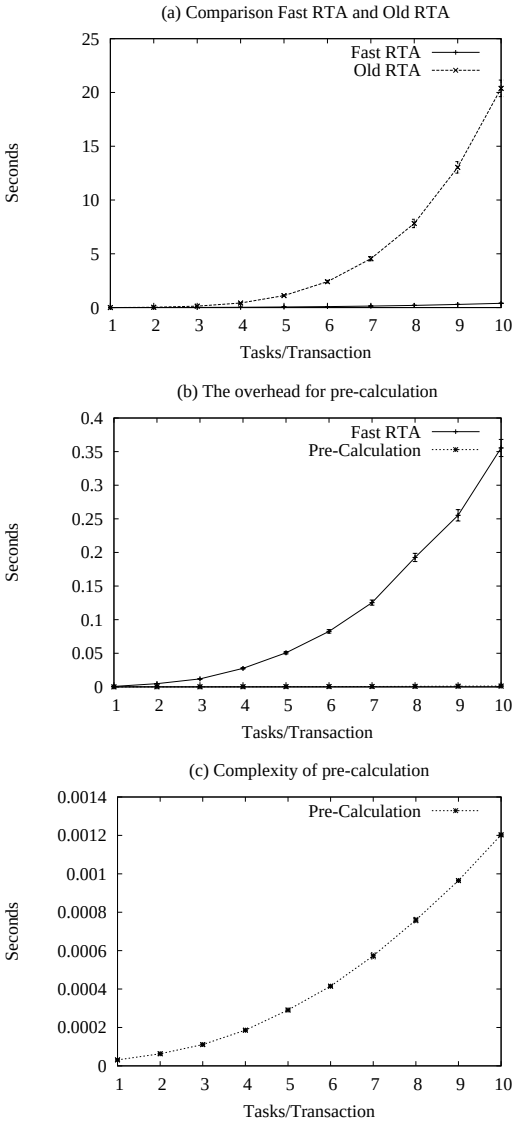

Figure 7. Execution time

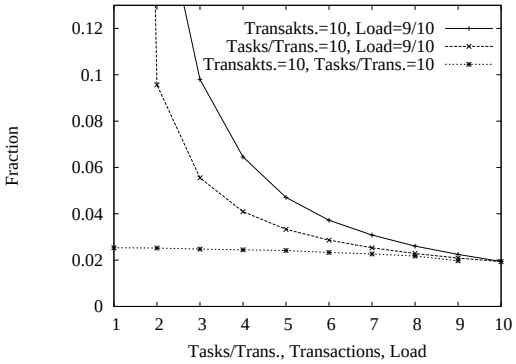

Figure 8. Relative execution time

text of on-line admission control, we generated task sets with 9/10 load, 10 transactions with 10 tasks/transaction and performed the RTA for a single task (corresponding to a dynamically arriving task to the system) at lowest priority. We generated 100 different tasks sets using execution times for the single task between 1000 and 6000. The result was that the average execution time for Fast RTA was 0.33ms and 44ms for Old RTA. The speedup for admission control is about 130 times, which is noticeable greater than in the previous simulations. The reason is that the new task is the only task in its transaction, which means that W_i^* is used for all interference computations and W_{ic} is never used (see Appendix A) and hence our improvement to W_i^* is isolated. In fact, in the preliminary work for this paper [5] a speedup of over 600 times was observed for a simplified task-model where fix-point iteration only required W_i^* to be computed.

Our conclusions from this simulation study are that: (1) Fast RTA performs significantly better than Old RTA. For anything but trivially small task sets the speedup is at least in the order of a magnitude, (2) Fast RTA brings down execution times for whole scenarios from the order of seconds to fractions of seconds, and (3) Fast RTA brings down execution times for single tasks from the order of some 100ms to the microsecond range. This decrease is important in order to make RTA a feasible technique to include in, e.g., on-line scheduling algorithms performing RTA on-line (admission control being an example) and optimizing allocation or configuration tools.

5 Conclusions and Future Work

In this paper we have presented a novel method that allows for an efficient implementation of the approximative Response-Time Analysis (RTA) for tasks with offsets presented by Tindell [13] and Palencia Gutierrez *et al.* [8].

The main effort in performing RTA for tasks with offsets is to calculate how higher (or equal) priority tasks interfere with a task under analysis. The essence of our method is to calculate and store this information statically and during response time calculations (fix-point iteration), use a simple table lookup. We have formally proved that the RTA-equations can be reformulated to allow such static representation of task interference.

We have, by simulations, shown that the speedup for our method compared to [8] is substantial. For realistically sized task sets (100 tasks), performing schedulability analysis gives a speedup of about 50 times. And from our evaluation we can conjecture that the relative improvement will be even higher for larger task sets. In an on-line RTA context, e.g., on-line admission control systems, our method outperforms previous methods by at over a factor of 100 and reducing the actual time to the micro second range.

Faster RTA have several positive practical implications: (1) Engineering tools (such as those for task allocation and priority assignment) can feasible 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.

We have earlier provided a tighter version of the RTA for tasks with offsets [7]. Our next step is to extend our method of static representation of task interference to our tighter RTA, yielding a RTA that is both significantly faster and provides less pessimistic response times than previous techniques. Further, we are currently starting a project where RTA for tasks with offsets will be used in software engineering tools. The RTA will be used both to perform schedulability tests and for automatic allocation of software to nodes in a distributed system.

References

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

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

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

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

[5] J. Mäki-Turja and M. Nolin. Faster Response Time Analysis of Tasks With Offsets. In *24^{th} IEEE Real-Time Systems Symposium (RTSS) Work In Progress Proc.*, December 2003.

[6] J. Mäki-Turja and M. Nolin. Speeding Up the Response-Time Analysis of Tasks with Offsets. Technical Report ISSN 1404-3041 ISRN MDH-MRTC-154/2004-1-SE, MRTC Report, Mälardalen Real-Time Research Centre, Mälardalen University, February 2004.

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

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

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

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

[11] O. Redell. *Response time analysis for implementation of distributed control systems*. PhD thesis, KTH, Department of Machine Design, 2003. Series: TRITA-MMK 2003:17.

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

[13] K. Tindell. Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets. Technical Report YCS-182, Dept. of Computer Science, University of York, England, 1992.

A Complete RTA formulas

In this appendix we provide the complete set of formulas to calculate the worst case response time, R_{ua}, for a task under analysis, τ_{ua}, as presented in Palencia Gutierrez *et al.* [8].

The interference transaction Γ_i poses on a lower priority task, τ_{ua}, if τ_{ic} coincides with the critical instant, is defined by (see Eq. 3 in this paper):

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

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

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

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

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

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

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

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

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

and

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

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

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

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

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

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

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

B Proofs of Theorems

In this appendix we provide proofs of theorems 1 and 2. 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. These proofs are also available in [6].

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 maximisation variable (v) to be moved outside the maximisation 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 expression:

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

Theorem 1 *$W_i^*(\tau_{ua}, t)$ as defined in Eq. 4 and $W_i^*(\tau_{ua}, t)$ as defined in Eq. 5 are equivalent.*

Proof 1

$$\underbrace{W_i^*(\tau_{ua}, t)}_{Eq.5} = J_i^{ind}(\tau_{ua}) + \underbrace{I_i^{ind}(\tau_{ua}, t)}_{Eq.7} =$$

$$J_i^{ind}(\tau_{ua}) + \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^+(\tau_{ua}, t)}_{Eq.8} =$$

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

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

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

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

In prooving theorem 2 we will use some lemmas.

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

Proof of Lemma 1

$$\underbrace{I_{ijc}^{Set2}(T_i)}_{Eq.2} = \underbrace{\left\lceil \frac{T_i - \Phi_{ijc}}{T_i} \right\rceil}_{0 \le \Phi_{ijc} < T_i\ (Eq.1)} C_{ij} = C_{ij} \qquad \square$$

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 of Lemma 2

$$\underbrace{I_{ijc}^{Set2}(t)}_{Eq.2} = \underbrace{\left\lceil \frac{t - \Phi_{ijc}}{T_i} \right\rceil}_{Assumption} C_{ij} = \underbrace{\left\lceil \frac{k*T_i + t' - \Phi_{ijc}}{T_i} \right\rceil}_{} C_{ij} =$$

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

$$k \underbrace{C_{ij}}_{Lem.1} + \underbrace{\left\lceil \frac{t' - \Phi_{ijc}}{T_i} \right\rceil}_{Eq.2} C_{ij} = k * I_{ijc}^{Set2}(T_i) + I_{ijc}^{Set2}(t') \quad \square$$

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

Proof of Lemma 3

$$\underbrace{T_i^{ind}(\tau_{ua}, T_i)}_{Eq.7} = \max_{\forall c \in hp_i(\tau_{ua})} \underbrace{W_{ic}^{+}(\tau_{ua}, T_i)}_{Eq.8} =$$

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

$$\max_{\forall c \in hp_i(\tau_{ua})} \left(\sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1} + \sum_{\forall j \in hp_i(\tau_{ua})} \underbrace{I_{ijc}^{Set2}(T_i)}_{Lem.1} - J_i^{ind}(\tau_{ua}) \right) =$$

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

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} I_{ijc}^{Set1}}_{Eq.6} - J_i^{ind}(\tau_{ua}) =$$

$$\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} + \underbrace{J_i^{ind}(\tau_{ua}) - J_i^{ind}(\tau_{ua})}_{} = \sum_{\forall j \in hp_i(\tau_{ua})} C_{ij} \quad \square$$

Theorem 2 *Assume* $t = k * T_i + t'$ *(where* $k \in \mathbb{N}$ *and* $0 \le t' < T_i$*), then*

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

Proof 2

$$\underbrace{T_i^{ind}(\tau_{ua}, t)}_{Eq.7} = \underbrace{W_{ic}^{+}(\tau_{ua}, t)}_{Eq.8} =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + \underbrace{I_{ijc}^{Set2}(t)}_{Lem.2}\right) - J_i^{ind}(\tau_{ua}) =$$

$$\max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + k * \underbrace{I_{ijc}^{Set2}(T_i)}_{Lem.1} + I_{ijc}^{Set2}(t')\right) -$$

$$J_i^{ind}(\tau_{ua}) =$$

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

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

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

$$k * \underbrace{\sum_{\forall j \in hp_i(\tau_{ua})} C_{ij}}_{Lem.3} + \max_{\forall c \in hp_i(\tau_{ua})} \sum_{\forall j \in hp_i(\tau_{ua})} \left(I_{ijc}^{Set1} + I_{ijc}^{Set2}(t')\right) -$$

$$J_i^{ind}(\tau_{ua}) =$$

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

$$k * T_i^{ind}(\tau_{ua}, T_i) + \underbrace{\max_{\forall c \in hp_i(\tau_{ua})} W_{ic}^{+}(\tau_{ua}, t')}_{Eq.7} =$$

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