Exact Best-Case Response Time Analysis of Fixed Priority Scheduled Tasks

Ola Redell and Martin Sanfridson
Mechatronics Laboratory, Dept. of Machine Design, KTH
{ola, mis}@md.kth.se

Abstract

We present a solution for exact calculation of the best-case response times of a periodic task set with fixed priorities. The solution is based on the identification of the best-case phasing of a low priority task compared to the higher priority tasks. This phasing occurs when the low priority task is released such that it finishes simultaneously with the releases of all the higher priority tasks, when these have experienced their maximum release jitter. A recurrence equation is applied to find the best-case response time. The dualism between worst-case and best-case response time calculation is characterized. The most important application of the solution is in the analysis of response jitter.

1. Introduction

In this paper we present the solution to the previously unsolved problem of finding the minimum response time of tasks in a preemptive fixed priority task set. The solution is based on the identification of the *best-case phasing* of a low priority task compared to its higher priority tasks. We show that a task has its best-case phasing when it finishes its execution at an instant when all higher priority tasks are released, after having experienced their maximum release jitter. The response time of a task that has executed in its best-case phasing is the shortest possible. Hence, the suggested method finds the exact best-case response time and not a lower bound, assuming that tasks can be phased arbitrarily.

Real-time system analysis has traditionally been focused on the analysis of worst-case behaviour in order to provide guarantees for tasks meeting their deadlines. There are however situations when the best-case behaviour of a task or a system is important as well. One such situation is in the calculation of response jitter, the maximum variation in the response time of a task or a sequence of tasks. Typically, response jitter is an important parameter in control systems. Control theory is dependant on the assumption that sampling and actuation is performed with precise periodicity and hence large variations in these periods can make an otherwise stable control system become unstable [8]. In order to guarantee control performance and stability it is therefore important to find tight bounds to the magnitudes of such variations. A good best-case analysis is a necessity for such tight bounds.

Analysis of fixed priority preemptive tasks known as RMA (Rate Monotonic Analysis), has been well developed since the paper by Liu and Layland in 1973 [5], from which it all originated. The set of analysis methods now handles a wide range of systems in which the original restricting assumptions of RMA have been successively removed. Systems that can be analysed include systems with task synchronisation (blocking), a mixture of periodic and aperiodic tasks, tasks with arbitrary deadlines etc. [1][3][4].

Even though RMA was initially formulated as a theory for analysis of tasks executing on a single processor, some extensions cover scheduling analysis in distributed systems. A method proposed by Tindell and Clark [7] produces worst-case response times of precedence related tasks in distributed systems. The method is based on local analysis of each node with the response jitter of one task or message being the release jitter of the next task in the sequence. Pessimistic bounds of response jitter are in this method a source for pessimism in the bounds of subsequent task response times. Hence, too pessimistic jitter bounds may make a schedulable system being considered unschedulable. Since response jitter is the difference between the worst- and best-case response times, finding tight bounds on these response times is important. While good methods exist for determining worst-case response times of tasks, the best-case has not received the same attention.

The method by Tindell and Clark was further developed by Palencia et al. [6] with an estimate of best-case response times. Palencia et al. find a lower bound on the response time of a task by assuming (optimistically) that all higher priority tasks finish at the instant the analysed lower priority task is released (after having experienced their best-case response times). This results in a correct lower bound for the best-case response times of tasks, but it is not exact. In a recent article, Henderson et al. [2] tried to further develop the ideas in [6] and find the exact best-case response times. However, their solution leads to a numerically intractable search through all possible orderings of higher priority task executions, prior to the release of the analysed low priority task.

This paper is organized as follows: In section 2 we define the task model and the problem to be solved. In section 3 we present an exact solution to the best-case analysis of task response times when the task deadlines are shorter than the periods. Analysis of task sets with longer deadlines is discussed in section section 4. Section 5 discusses the applicability of the method and gives suggestions for further work. Section 6 offers our conclusions.

2. Task model and problem formulation

A uniprocessor executes a set of independent tasks τ_k with $k = 1, 2,..., m$. Each task is periodic and characterized by the following parameters:

- a period time, T_k
- a fix and unique priority
- an execution time interval, $[C_k^{min}, C_k^{max}]$, limited by the task's minimum and maximum execution times. Each task instance may assume any execution time within the interval.
- a deadline, D_k, such that $D_k \leq T_k$
- a maximum release jitter, J_k.

A task τ_k *arrives* periodically with period T_k. On arrival, the task is either *released* (put in the ready queue) immediately, or the release is delayed by an amount of time called *release jitter*. The magnitude of the jitter can change from one instance to the next, but is bounded by $[0, J_k]$. The arrival time of a task τ_k is denoted a_k, while the release time is r_k. After release, a task starts its execution (*start time*) as soon as it is the highest priority ready task in the system. The *finishing time* of a task is denoted f_k and the response time R_k is defined to be the time between arrival and finish: $R_k=f_k-a_k$. In the following, tasks are labelled by their numbers. The analysed task is denoted task i while task j denotes a task in $hp(i)$. $hp(i)$ is the set of tasks with higher priorities than task i.

There are no exclusion or precedence constraints that can make a lower priority task block the execution of a higher priority task. Hence, the only sources of variation in the response time of a task is the execution time, release jitter and interference from higher priority tasks. The problem is to calculate the best-case response times R^b of the tasks in the task set.

As a reference for the solution described in this paper, we state the equivalent result for worst-case calculations here [1]. The worst-case response time R^w of a task i is expressed by:

$$R_i^w = w_i + J_i \tag{1}$$

where

$$w_i = C_i^{max} + \sum_{j \in hp(i)} \left\lceil \frac{w_i + J_j}{T_j} \right\rceil \cdot C_j^{max} \tag{2}$$

Since equation (2) cannot be solved analytically, it has to be solved by iteration starting with e.g. $w_i = C_i^{max}$. When the iteration has converged, equation (1) is used to calculate the worst-case response of task i.

We will henceforth assume that any task set that is subject to best-case analysis has already been verified to be schedulable. Such a verification could be done by comparing the worst-case response times computed by (1) to the task deadlines.

The assumption that the task deadlines are smaller than the periods is a necessary requirement for the analysis derived in this paper to be exact. In section 4 we discuss how task sets that do not satisfy this requirement can be handled.

3. Best-case Analysis

The response time of a task is the duration between its arrival time and its corresponding finishing time. Because of interference, a low priority task can not always start executing immediately when it is released. We use Lemma 1 to state that a task that experiences its best-case response time must have arrival, release and starting times that coincide.

Lemma 1. A task that achieves its minimum response time must arrive and be released simultaneously, at an instant when no higher priority task is executing or is released.

Proof. Since any release jitter increases the response time, it is clear that the best-case response time corresponds to zero release jitter. Hence the arrival and release times must coincide.

Assume that a low priority task *i* arrives when a higher priority task *j* occupies the processor. The response time of *i* will always be shortened by delaying the arrival time to the instant when the higher priority task has stopped executing. This comes from the fact that the delay of *i*'s arrival time will not affect its start or finishing times and the response time of *i* is counted from its arrival to its finish.

Q.E.D.

In the following we will assume that the analysed task satisfies Lemma 1 and hence has arrival, release and start times that coincide.

To calculate the best-case response time, we first need to know when the low priority task *i* should be released. Obviously, the worst release time is the critical instant of that task. It is a very reasonable assumption that task *i* should be released before this instant such that it finishes exactly when all higher priority tasks *j* arrive simultaneously. For example, consider Figure 1 which shows three priority-ordered tasks with no release jitter. If the releases of the instances of task 2 are *advanced* in time (moved to the left) the release (and arrival) instant r_3 needs to be *delayed* (moved to the right) to keep the finishing time at f_3, rendering a shorter response time of task 3.

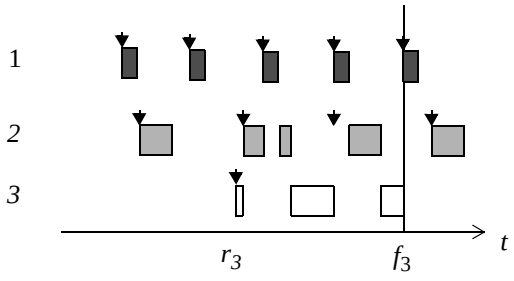

Figure 1. Example of phasing two higher priority tasks relative to task 3 which is released at r_3 and finishes at f_3. Vertical arrows indicate release instants. No release jitter is assumed in this example.

In fact, Theorem 1 states what phase every high priority task should have relative to the finishing time of the low priority task *i* in order for task *i* to experience its best-case response time. There may be other choices of phases that will give an equally short response time, but none will be shorter. Note that contrary to the case of critical instant, it is the finishing time of *i* that is considered, not the release time.

Theorem 1. Best-case phasing.
The best-case phasing of a task *i* occurs whenever it finishes simultaneously with the release of all its higher priority tasks and these have experienced their maximum release jitter. All instances of tasks in *hp*(*i*) released prior to the finish of *i* should have zero release jitter.

Proof. Let τ_j, $j = 1, 2, 3,..., i$ denote a set of priority-ordered tasks with task *i* being the task with the lowest priority. The task set is successfully scheduled on a uniprocessor. Consider a particular execution of task *i* which finishes at time f_i. Instances of a task *j*, $j < i$, arrive periodically at times $\{..., t_j-2T_j, t_j-T_j, t_j, ...\}$ where the instance of *j* that arrives at t_j is the last instance of task *j* to execute before f_i.

We assume, without loss of generality, that task *i* will always finish at f_i. Hence, task *j* is not allowed to be phased such that any of its instances executes at f_i. In the following, we will use the fact that the release time of task *i* is given implicitly by f_i, the phasing of higher priority tasks, their release jitter and Lemma 1.

The release time of *i* is either unchanged or delayed by reducing the release jitter of an instance of task *j* that arrives at t_j or earlier. Increasing the release jitter of one of these instances will not delay the release time of *i*. Hence the instances of task *j* up to and including the one that arrives at t_j should have zero release jitter for task *i* to have its best-case phasing.

Delaying the arrival time t_j will not delay the release time of *i*. Advancing the arrival time t_j will not advance the release time of *i*. The release time of *i* is either unchanged or delayed by an advance of t_j. As a consequence, the release of *i* is the latest when t_j is advanced as much as possible – up to a point when a succeeding instance of *j* gets released at f_i. The succeeding instance arrives at $t_j + T_j$. Release jitter will delay its release enabling a further advance of t_j. Thus, the instance of *j* that is released at f_i should experience its maximum release jitter, J_j. Repeating the argument successively for all τ_j, $j = 1, 2, 3,..., i-1$ proves the theorem.

Q.E.D.

Remark. The instant f_i associated with the finishing time of task *i* is identical to the *critical instant* of task *i*-1 as defined in [5]. We call the finishing time f_i in the situation of best-case phasing the *favourable instant* to underline the dualism between worst-case and best-case analysis.

As a direct result of Theorem 1, we formulate the following corollary which will be useful in the discussion about task sets with deadlines longer than the periods in section 4.

Corollary 1. Given a schedule of tasks 1,...,*i*-1, there cannot exist an interval with arbitrary length *L*, in that schedule, that includes more idle time than the interval $[f_i\text{-}L, f_i]$. Where f_i is a favourable instant of some lower priority task *i* as defined in Theorem 1.

Proof. Assume that an interval $[a, a+L]$ in the schedule exists that does include more idle time than $[f_i\text{-}L, f_i]$. Let the total amount of idle time in $[a, a+L]$ be *l*. Assume that a low priority task *i* with $C_i^{min} = l$ and $T_i = \text{LCM}(T_1, \ldots, T_{i-1})$, is added to the task set. This task would achieve its minimum response time if it was executed in $[a, a+L]$ and not in $[f_i\text{-}L, f_i]$. This contradicts Theorem 1 and hence such an interval $[a, a+L]$ cannot exist.

Q.E.D.

We next continue by deriving an expression for the response time of a task *i*, when it is phased as described in Theorem 1 and when it satisfies Lemma 1. This is the best-case response time of task *i*.

Theorem 2. Best-case response time.
Given a schedulable set of independent fixed priority scheduled tasks with deadlines equal to or smaller than their periods, the best-case response time R_i^b of a task *i* in the set satisfies the following equality:

$$R_i^b = C_i^{min} + \sum_{j \in hp(i)} \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^{min} \qquad (3)$$

where $\lceil x \rceil_0 = \max(0, \lceil x \rceil)$.

Proof. Assume a best-case phasing of task *i* according to Theorem 1 and that the arrival, release and start times of task *i* coincide as stated in Lemma 1. We define the phase from the release of task *i* to the first following release of a higher priority task *j* to be ϕ_j. Figure 2 shows the best-case phasing of a low priority task 3 when executed together with the two higher priority tasks 1 and 2. Tasks 1 and 2 are released at $\{..., f_3\text{-}2T_1\text{-}J_1, f_3\text{-}T_1\text{-}J_1, f_3, ...\}$ and $\{..., f_3\text{-}2T_2\text{-}J_2, f_3\text{-}T_2\text{-}J_2, f_3, ...\}$ respectively. Task 3 arrives at at times $\{..., r_3\text{-}T_3, r_3, ...\}$. Task instances released at and after f_3, do not need to be considered since they

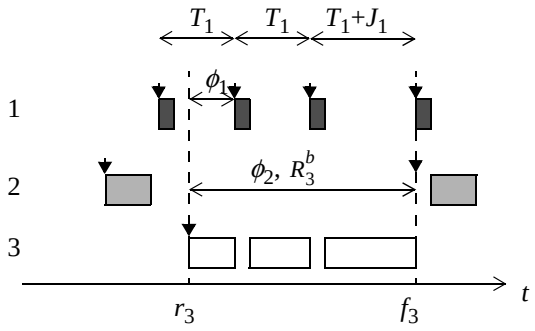

Figure 2. The best-case phasing of task 3 with release time r_3 and finishing time f_3.

Task *i*'s response time will be minimal only if its execution time is minimal. Hence, the minimum response time of task *i* can be expressed as the sum of the minimum execution time and interference by all higher priority tasks:

$$R_i^b = C_i^{min} + \sum_{j \in hp(i)} I_i^b(j) \qquad (4)$$

Here $I_i^b(j)$ is the interference of task *i* by task *j* when the tasks are phased according to the best-case phasing for task *i*. Hence, in order to find the best-case response time of task *i*, we need to find the amount of interference due to all higher priority tasks. By studying Figure 2 we find that there are two different cases to be considered for the interference of a higher priority task *j* to the execution of task *i*:

Case 1: $R_i^b > T_j + J_j$

and

Case 2: $R_i^b < T_j + J_j$

where $R_i^b = f_i - r_i$. Since R_i^b is the best-case response time of *i* it cannot be equal to $T_j + J_j$ as a consequence of Lemma 1. A more general inequality, based on Lemma 1, that holds for all higher priority tasks *j* is:

$$R_i^b \neq k \cdot T_j + J_j \text{ for } k = 1, 2, \ldots \qquad (5)$$

Case 2 is exemplified by task 2 which has T_2+J_2 large enough not to interfere with task 3's execution at all. This case is obviously simple to handle. Case 1 is exemplified by task 1, which does interfere with task 3. It follows from Figure 2 that R_3^b can be expressed as

$$R_3^b = f_3 - r_3 = \phi_1 + 2 \cdot T_1 + J_1$$

while a general relation between the best case response time of a low priority task i and any $j \in hp(i)$ that belongs to case 1 is

$$R_i^b = f_i - r_i = \phi_j + n_j \cdot T_j + J_j \tag{6}$$

where $n_j \in \{1, 2, \ldots\}$ is the number of instances of task *j* that interfere with task *i*. As a result of Lemma 1 we have

$$0 < \phi_j < T_j \tag{7}$$

and by rewriting (6) as

$$\phi_j = R_i^b - n_j \cdot T_j - J_j \tag{8}$$

we get

$$0 < R_i^b - n_j \cdot T_j - J_j < T_j \tag{9}$$

Rewriting this expression results in

$$\frac{R_i^b - J_j}{T_j} > n_j > \frac{R_i^b - J_j}{T_j} - 1 \tag{10}$$

Now, since $(R_i^b - J_j)/T_j$ cannot be an integer (by equation 5), there are two equally valid solutions for n_j in (10):

$$n_j = \left\lfloor \frac{R_i^b - J_j}{T_j} \right\rfloor \tag{11}$$

and

$$n_j = \left\lceil \frac{R_i^b - J_j}{T_j} \right\rceil - 1 = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil \tag{12}$$

Any of the two expressions (11) and (12) can be used to express the number of instances of task *j* that interfere with *i* during its best-case execution. We will henceforth use (12) since it has characteristics better suited for finding the best-case response time of task *i*. This will be explained later.

By combining the results for the two cases of higher priority task interference, we find that the number of interfering instances of task *j* can be expressed as:

$$n_j = \begin{cases} 0 & \text{for } R_i^b < T_j + J_j \\ \left\lceil \dfrac{R_i^b - J_j - T_j}{T_j} \right\rceil & \text{for } R_i^b > T_j + J_j \end{cases} \tag{13}$$

which is equivalent to

$$n_j = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil_0 \tag{14}$$

It is now possible to derive an expression for the interference of a task *i* by a higher priority task *j*, when task *i* is phased according to the best-case phasing defined in Theorem 1. By taking into account that, in the best case for task *i*, all interfering instances of task *j* execute with their minimum execution times, we find the interference to be

$$I_i^b(j) = \left\lceil \frac{R_i^b - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^{min} \tag{15}$$

We now add up the interference of all tasks in $hp(i)$ and the minimum execution time of task *i* itself, as given by (4). This results in the expression for the best-case response time of *i* as shown in (3).

Q.E.D.

At this point we have an expression, (3), that the best-case response time of *i* must satisfy. Just like in worst-case response time calculations however, this equation cannot be solved analytically. Therefore, the minimum response time has to be found through iteration. We now need to show that this is possible. We restate equation (3) as a recurrence equation,

$$R^{n+1} = C_i^{min} + \sum_{j \in hp(i)} \left\lceil \frac{R^n - J_j - T_j}{T_j} \right\rceil_0 \cdot C_j^{min} \tag{16}$$

and note that the right hand side corresponds to the

complete high priority workload released in an open interval $(f_i\text{-}R^n, f_i)$ plus the workload of task i itself. Figure 3 shows how R^n should be interpreted.

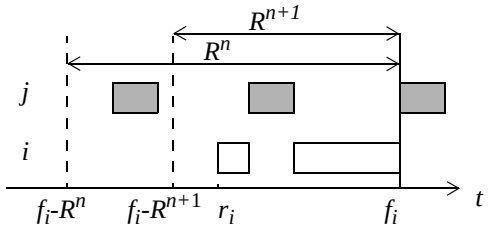

Figure 3. Interpretation of R^n during recursion.

We first need to show that recursion over (16) will converge to a stable value. Then we need to show that this value is the correct minimum response time and not one of the other possible solutions to (16) (there is usually more than one, see e.g. Figure 4).

We note that there must exist a finite R^0 such that the interval $(f_i\text{-}R^0, f_i)$ includes some interval during which the processor is either idle, executes tasks with a priority lower than i, or executes a previous instance of task i (whose workload is not included in (16)). If the iteration is started with such an R^0 we know that $R^1<R^0$. We will call an R^0 of this type a *valid* R^0.

Lemma 2. Iteration over R^n in (16) starting with a valid R^0, will converge to the largest possible solution to (16).

Proof. By observing that the right hand side of (16) is monotonically decreasing with decreasing R, and since the number of possible values produced by (16) is finite, the iteration will converge. It is obvious that this solution is the largest solution to (16).

Q.E.D.

We will henceforth use R^* to denote the largest solution of (16). We also note that any R^0 larger than or equal to R^* will make the iteration converge to R^*. It now remains to be shown that R^* is in fact the best-case response time of task i. The two following lemmas take care of that:

Lemma 3. The instant $f_i\text{-}R^*$ must correspond to the release (and start) of task i.

Proof. We show this by making two statements about R^* that can easily be verified through observation of (16).

First, R^* corresponds to an interval $[f_i\text{-}R^*, f_i]$ during which the processor is completely occupied executing task i or higher priority tasks. Furthermore all these tasks are released within the interval $[f_i\text{-}R^*, f_i)$.

Second, R^* cannot correspond to an instant $f_i\text{-}R^*$ at which a task in $hp(i)$ is released. This is easiest understood by noting that the workload due to tasks in $hp(i)$ that is included in (16) must be released within the open interval $(f_i\text{-}R^*, f_i)$. This behaviour is a result of choosing expression (12) instead of (11) for the number of interfering instances of a higher priority task.

By combining the two observations, we find that task i must be released and start its execution at $f_i\text{-}R^*$.

Q.E.D.

Lemma 4. The best-case response time of a task i is equal to R^*.

Proof. Assume that R is a solution to (16) such that $R^*>R$. Then there must be a release of at least one task j in $hp(i)$ within the interval $(f_i\text{-}R^*, f_i\text{-}R]$ (interval open to the left by Lemma 3) that has a total execution time of $R^*\text{-}R$ and therefore interferes with the execution of i. Hence, R cannot be the best-case response time of i. This argument can be applied to any R that is smaller than R^*, and therefore R^* must be the best-case response time of i.

Q.E.D.

Figure 4 shows the difference between the two solutions R and R^* of (16), for a task set with two tasks.

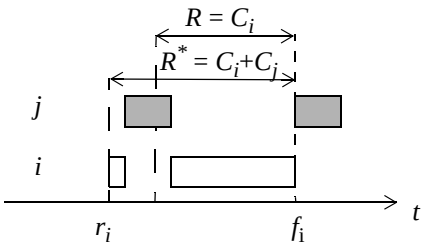

Figure 4. Two solutions to (16): R and R^*. R cannot be the best-case response time of i.

Theorem 3. Best-case response time calculation. The best-case response time of a task i, R_i^b, can be found by iteration over (16) starting with a value $R^0 = R_i^w$ as derived from (1). When the iteration has converged, the corresponding R^n is equal to the best-case response time of task i.

Proof. The worst-case response time of i, R_i^w, is

always larger than or equal to R_i^b and hence R^*. Therefore, by Theorem 2 and Lemmas 3 and 4, the iterations will converge to R_i^b.

Q.E.D.

Remark. Another possible initial value R^0 that is always larger than or equal to R_i^b is T_i.

4. Analysis of Task Sets with D > T

In the above analysis, one important assumption has been that a task's deadline has to be smaller than or equal to its period. In this section we discuss how the analysis is affected if this assumption is removed. It will be evident that the analysis derived in section 3 can be used to compute correct lower bounds on the best-case response times of tasks with deadlines longer than the periods.

We first note that even though deadlines may be larger than the periods, the interval (of arbitrary length) with the most amount of idle time is still correctly specified by Corollary 1. This result does not depend on the deadline of the imaginary lower priority task. Hence, if an analysed task *i* can be phased to arrive, be released, and start its execution simultaneously and then finish at the favourable instant as defined in Theorem 1, the analysis derived in section 3 would still be exact. Unfortunately however, when tasks are allowed to have deadlines larger than their periods, the exact analysis falls. This is due to the fact that earlier instances of task *i* may interfere with the execution of the analysed instance. Therefore the implication of Lemma 1, that a task that experiences its best case response time must start its execution at the instant it arrives, does not hold.

To illustrate this, consider the following simple example involving a task set with two tasks. Task *j* which has the highest priority has a constant execution time of $C_j^{min} = C_j^{max} = 4$ and a period $T_j = 8$. Task *i* has a period $T_i = 5$ and an execution time $C_i^{min} = C_i^{max} = 2.5$. Both tasks have zero release jitter ($J_i = J_j = 0$) and their deadlines are $D_j = 8$ and $D_i = 10$. The total utilisation of the task set is 1 and the tasks are feasibly schedulable on a uniprocessor (this can be verified by, for example, a simple schedule simulation). Figure 5 shows the execution of two instances of task *j* together with the execution of two instances of task *i*. The instances of task *i*, numbered 1 and 2, are released at 0.5 and 5.5 respectively. The release times have been chosen such that instance 2 would finish at a favourable instant ($t = 8$) unless instance 1 would interfere with its execution. However, due to interference from instance 1, instance

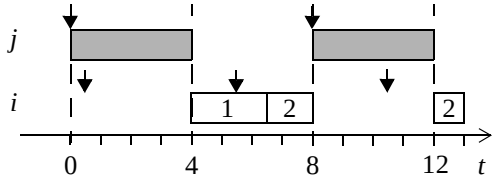

Figure 5. Instance 2 of task *i* cannot start executing when it is released and hence fails to finish at 8 due to interference from an earlier instance. Vertical arrows indicate release times.

2 finishes at 13. It is obvious that, if instance 2 is to finish at 8, task *i* has to be phased to be released earlier. Hence in this case, task *i* cannot be released such that it starts executing immediately, if it is to finish in a favourable instant as defined by Theorem 1.

As a direct consequence of the above discussion, we find that allowing deadlines larger than task periods can not make the best case response times get smaller. Hence, if we assume (optimistically) that the analysed task *i* will not experience any interference by preceding instances, the analysis method described in section 3 can be used to compute a lower bound to the best case response time of task *i*. However, the value produced will not be exact, as is the case when $D_i \le T_i$.

5. Discussion

The best-case response time analysis derived in section 3 is similar to the corresponding worst-case analysis. One major difference is that in best-case analysis, the finishing time of the analysis task is fixed to the release of all the higher priority tasks, and the algorithm searches the latest possible release time of the task. Furthermore, the iteration performed to find the best-case response time must be started with a value larger than or equal to the final solution, which is the largest value satisfying the recurrence equation. If a task model with no release jitter is used, equation (3) becomes:

$$R_i^b = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^b - T_j}{T_j} \right\rceil \cdot C_j \qquad (17)$$

Hence, the maximisation over zero can be skipped since R_i^b can never be smaller than or equal to zero. Obviously equation (17) is very similar to the corresponding equation for worst-case analysis.

The inclusion of release jitter in the task model used in the derivation of (3) makes it applicable to methods for response time analysis in distributed

systems with precedence related tasks, as discussed in section 1. However, when using any best-case analysis in such methods, one must be aware that the precedence relations between tasks may make the best-case analysis incorrect. This is a result of the fact that the tasks in such a system are not independent and low priority tasks in one task sequence may block higher priority tasks in another.

The exact best-case analysis is based on the assumption that tasks can be arbitrarily phased. This means that the periodic arrival times of different tasks can assume any relative phasing to each other. If a task model with offsets is used instead, fixing the phasing between task arrivals, the analysis described here can be used to produce lower bounds on the best-case response times. This is similar to the upper bounds on worst-case response times found by corresponding worst-case analysis methods when phases are not arbitrary.

An obvious target for future work will be to find exact best-case response times for tasks with deadlines larger than their periods.

6. Conclusions

We have derived an algorithm for the calculation of *best-case* response times for independent tasks with fixed priorities. The method produces exact best-case response times for all tasks with $D_i \leq T_i$, and lower bounds for tasks with deadlines longer than the periods. If offsets are used in the task model such that the tasks are not arbitrarily phased, the analysis produces lower bounds on the best-case response times.

The presented algorithm is similar to the one frequently used in analysis of *worst-case* response times, and the dualism between best-case and worst-case is quite clear. A difference in the recurrence equation defined here is that the iteration starts from an initially high value and iterates down to the best-case response time. The solution is found at the release time of the low priority task; whereas in worst-case response time analysis, the solution is found at the finishing time.

We have also identified the best time for a low priority task to finish, rendering the shortest possible response time. The *favourable instant* for a low priority task to finish execution is the *critical instant* of the next higher priority task.

7. Acknowledgements

The authors wish to acknowledge Christer Norström, Martin Törngren and Jan Wikander for valuable comments to early versions of this paper.

This work was partially supported by SSF, the Swedish strategic research initiative, through ARTES project 0005-18: PICADOR, and partially supported by VINNOVA, the Swedish agency for innovation systems, through the DICOSMOS project.

References

[1] N. Audsley, A. Burns, K. Tindell, M. Richardson and A. Wellings, “Applying New Scheduling Theory To Static Priority Pre-emptive Scheduling”, Software Engineering Journal, 8(5):284-292, September 1993.

[2] W. Henderson, D. Kendall and A. Robson, “Improving the Accuracy of Scheduling Analysis Applied to Distributed Systems”, Intl. Journal of Real-Time Systems, Kluwer, 20(1):5-25, January 2001.

[3] M. Klein, T. Ralya, B. Pollack, R. Obenza and M. González Harbour, “A Practitioners Handbook for Real-Time Systems Analysis”, Kluwer Academic Publishers, 1993.

[4] J.P. Lehoczky, “Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines”, Proc. of the 11th IEEE Real-Time Systems Symposium, pp. 201-209, December 1990.

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

[6] J.C. Palencia Gutiérrez, J.J. Gutiérrez García and M. González Harbour, “Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems”, Proc. of tenth Euromicro Workshop on Real-Time Systems, Berlin, pp. 35-44, June 1998.

[7] K. Tindell and J. Clark, “Holistic Schedulability Analysis for Distributed Hard Real-Time Systems”, Microprocessing & Microprogramming, 50(2-3):117-134, April 1994.

[8] M. Törngren, “Fundamentals of implementing Real-time Control applications in Distributed Computer Systems”, J. of Real-time Systems, Kluwer, 14:219-250, 1998.