The Non-Optimality of the Monotonic Priority Assignments for Hard Real-Time Offset Free Systems

J. GOOSSENS AND R. DEVILLERS
{jgoosens, rdevil}@ulb.ac.be
Université Libre de Bruxelles, Département d'Informatique CP 212, Boulevard du Triomphe, 1050 Brussels, Belgium.

Abstract. In this paper, we study the problem of scheduling hard real-time periodic tasks with static priority preemptive algorithms. We consider tasks which are characterized by a period, a hard deadline, a computation time and an offset (the time of the first request), where the offsets may be chosen by the scheduling algorithm, hence the denomination offset free systems. We study the rate monotonic and the deadline monotonic priority assignments for this kind of system and we compare the offset free systems and the asynchronous systems in terms of priority assignment. Hence, we show that the rate and the deadline monotonic priority assignments are not optimal for offset free systems.

Keywords: hard real-time scheduling, periodic task set, static priority preemptive algorithm, synchronous system, asynchronous system, rate monotonic scheduler, deadline monotonic scheduler, feasibility interval.

1. Introduction

We shall consider the scheduling of a hard real-time periodic task set. The set is composed of $n$ periodic tasks $T_1, T_2, \ldots, T_n$, each task $T_i$ being characterized by a period $T_i$, a deadline $D_i$, an execution time $C_i$ and an offset $O_i$, which represents the time at which the first request for $T_i$ occurs. The $T_i$'s requests are separated by $T_i$; the execution must finish before the deadline $D_i$ ($C_i \leq D_i \leq T_i$). All those characteristics are generally assumed to be commensurable, so that, from a judicious choice of the time scale, they may always be taken merely as natural integers. Particular interest has been devoted in the literature to special subclasses of such systems, both for their potential simplification and for their practical interest: synchronous systems correspond to the case where $O_i = 0$ for all $1 \leq i \leq n$, and systems with late deadlines correspond to the case where $D_i = T_i$ for all $1 \leq i \leq n$ (each task must be completed before the next request for it occurs). We study the problem of scheduling a task set for a mono-processor system with a static priority preemptive algorithm. In addition we will assume that the switching times (including scheduling) may be neglected (hence the special interest devoted to the less time consuming fixed priority strategies); we will assume to have independent tasks.
The most important results about synchronous systems with late deadlines are due to Liu and Layland [6] and Serlin [7]. In particular they showed that the rate monotonic scheduler which assigns a static priority to each task inversely proportional to its period, is optimal for this kind of system.

**Theorem 1** ([6]) *If a feasible static priority assignment exists for some task set where all tasks are started at the same time and all deadlines equal their period, the rate monotonic assignment is feasible for that task set.*

In this theorem, a feasible static priority assignment means that with these priorities the static scheduler will always meet the deadlines. Theorem 1 implies that if a task set is unschedulable for the rate monotonic priority assignment, the task set is unschedulable for all fixed priority assignments, i.e. the rate monotonic assignment is *optimal* for synchronous systems with late deadlines. Liu and Layland have also proved that the worst case in terms of response time for a request of task \( \tau_i \) occurs when all requests of higher priority tasks are synchronized with the request of \( \tau_i \); we define the *response time* to be the time between the arrival of the request and the completion of its processing. Hence, if a request of a task \( \tau_i \) is schedulable in the synchronous situation, it is also the case in all asynchronous situations; from a schedulability point of view, the synchronous case is the worst case. These results have been extended to any synchronous system by Leung and Whitehead [5], who showed the optimality of the deadline monotonic scheduling which assigns a static priority to each task inversely proportional to its deadline.

**Theorem 2** ([5]) *If a feasible static priority assignment exists for a synchronous task set, the deadline monotonic assignment is also feasible for it.*

It may be observed that the deadline monotonic scheduling reduces to the rate monotonic one when each \( D_i = T_i \), and that Theorem 2 encompasses Theorem 1.

We can see that, with Theorems 1 and 2, the schedulability of a task set is considered in the worst case, when all tasks are started at the same time (\( O_i = O_j \quad \forall i, j \)). As a consequence, if the real-time system for which the scheduling is computed does not have definite requirements about the task start times (offsets), then Theorems 1 and 2 are too pessimistic. Indeed, a task set can be unschedulable with respect to Theorem 1 (or Theorem 2) while being schedulable if we consider different task start times for each task.

**Example:** Consider two tasks \( \tau_1 \) and \( \tau_2 \) with \( T_1 = D_1 = 6 \), \( C_1 = 2 \) and \( T_2 = D_2 = 8 \), \( C_2 = 5 \). The priorities of tasks \( \tau_1 \) and \( \tau_2 \) are given by the rate monotonic scheduler, hence \( \tau_1 \) has a higher priority than \( \tau_2 \) (\( \tau_1 > \tau_2 \)). If all tasks are started at the same time, this task set is not schedulable (see figure 1). But, the task set is schedulable with \( O_1 = 1 \) and \( O_2 = 0 \) (see figure 2).

We consider in this paper hard real-time systems which have no definite requirements about the task start times. In such systems the offsets will be chosen beforehand by the scheduling algorithm, together with the priorities. We call this
kind of system: offset free system. In this context, it is not necessarily true that the rate/deadline monotonic scheduler still gives the optimal static priority assignment (Theorem 1 or 2). The optimality of the rate/deadline monotonic scheduler was already reconsidered for systems where the tasks are not started at the same times, but where the offsets are fixed beforehand (by the user, from the application constraints) contrary to our offset free systems where the offsets can be chosen by the scheduling algorithm itself. Systems with fixed offsets were studied basically by Leung and Whitehead [5] under the denomination of asynchronous systems. The synchronous systems studied by Liu and Layland are special cases of asynchronous systems (the case where $O_i = 0$ for all $1 \leq i \leq n$), but the asynchronous systems and the offset free systems are of different natures.

The remainder of the paper is organized as follows: section 2 provides a background to the schedulability problem of asynchronous systems. In section 3 we present the optimality results for asynchronous systems; in section 4 we state new results about the optimality of offset free systems and in section 5 we consider the practical interest of offset free systems.

2. Feasibility interval

Before studying the optimality of the rate/deadline monotonic priority assignment, we shall first consider the problem of deciding if a system is schedulable (or not), for a priority and offset assignment. In some circumstances, it is possible to devise simple necessary and/or sufficient conditions for the schedulability of a given task.
set. For instance, for late deadline synchronous systems with the rate monotonic scheduling, Liu and Layland [6] devised the rule that a task set is schedulable if (but not if) \( \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(\sqrt{2} - 1) \). In all generality, we also have a task set is certainly not schedulable if \( \sum_{i=1}^{n} \frac{C_i}{T_i} > 1 \). If no such conditions are known, or if they do not give a definite answer for a given task set, of course it is not possible to simulate the evolution of the system till the end of ages in order to check if something goes wrong. However, it is generally possible to determine an interval such that if nothing goes wrong in it then nothing will ever go wrong.

**Definition.** For a given scheduling algorithm and a task set, a feasibility interval is a finite interval such that it is sure that no deadline will ever be missed if, when we only keep the requests made in this interval, all deadlines for them in this interval are met.

For instance, we have:

**Theorem 3** For a synchronous system with a static priority scheduler, \([0, \max_{i=1 \ldots n} \{D_i\}]\) is a feasibility interval.

**Proof:** Leung and Whitehead have shown that the largest response time of task requests occurs in the synchronous case. More exactly, Liu and Layland first showed the property (with a slightly different setting) in the late deadline case, but Leung and Whitehead noticed that it was valid for any deadline configuration. Hence, for each task \( \tau_i \) we only have to check if it meets its first deadline \( D_i \); for the whole task set, we only have to consider all deadlines in the interval \([0, \max_{i=1 \ldots n} \{D_i\}]\).

In the general asynchronous case, we have:

**Theorem 4** ([5]) For an asynchronous system with a static priority scheduler, \([O_{max}, O_{max} + 2P]\) is a feasibility interval, where \( O_{max} \) is the maximal task offset and \( P \) is the least common multiple of all the task periods.

It may be observed that:

- \([0, O_{max} + 2P]\) is also a feasibility interval, meaning that it is only necessary to simulate the evolution of the system up to \( O_{max} + 2P \). But Theorem 4 says that it is not necessary to start the system (i.e., the requests) before \( O_{max} \);
- even so, Theorem 4 is certainly not optimal since it does not reduce to Theorem 3 in the synchronous case \((2P > \max_{i=1 \ldots n} \{D_i\})\).

From [2], it is possible however to improve this test, in the following way. First, from the results of [6], extended in [5], we get:

**Theorem 5** If an asynchronous system has a common release time, i.e., an instant \( S \) such that \( \forall i : S = O_i + n_iT_i \) for some \( n_i \), then \([S, S + \max_{i=1 \ldots n} \{D_i\}]\) is a feasibility interval, and the deadline/rate monotonic scheduling is optimal.
**Proof:** Immediate from Theorem 3, and the fact that synchronous releases is the worst case. 

If this is not the case, let us first observe that in any case:

**Theorem 6** Any feasible schedule of an asynchronous system is finally periodic, i.e. periodic from some point.

**Proof:** For any integer instant time $t$ after $O_{\text{max}}$, let us call the configuration of the schedule the list of pairs $\{ (\varepsilon_i, \delta_i) \mid 1 \leq i \leq n \}$ where $\varepsilon_i$ is the amount of processor time used by task $\tau_i$ from its last request, and $\delta_i$ is the real time elapsed from this last request. For static priority pre-emptive schedulers, the configuration at time $t + 1$ is univocally determined by the configuration at time $t$. Since $\forall i : \varepsilon_i \leq C_i$, $\delta_i < T_i$ and they are both natural integers, there are finitely many possible configurations and we may find two instants $t_1, t_2 \quad (t_1 < t_2)$ with the same configuration. From there on, the schedule will repeat periodically (with a period dividing $t_2 - t_1$).

For unfeasible schedules, we cannot say anything since we did not even describe what would happen then. If extra delays are accumulated, then the schedule is certainly not periodic, but it is if requests with missed deadlines are simply dropped.

It may be noticed that this proof is valid not only for any static priority schedule, but more generally for any schedule where the priorities only depend on the configuration. For instance, this is also true for the (dynamic) earliest deadline schedule also considered in [6] (here the priority of task $\tau_i$ is $1/(D_i - \delta_i)$, if $\varepsilon_i < C_i$, otherwise the task is not active).

For static priority schedules, the Theorem 6 may even be refined:

**Theorem 7** Let $S_i$ be inductively defined by $S_1 = O_1$, $S_i = \max \{ O_i, O_i + \lfloor \frac{\varepsilon_i - \delta_i}{T_i} \rfloor \} \quad (i = 2, 3, \ldots, n)$, then if the task set is ordered by decreasing priorities and has a feasible schedule, it is periodic from $S_1$ with period $P = \text{lcm}[T_i \mid i = 1, \ldots, n]$.

**Proof:** This immediately results by induction on $n$. The property is true in the trivial case where $n = 1$: the schedule for $\tau_1$ is periodic of period $T_1$ from the first release of $\tau_1$. By hypothesis induction, the schedule for the task subset $\{ \tau_1, \ldots, \tau_{i-1} \}$ is periodic from $S_{i-1}$ of period $P_{i-1} = \text{lcm}[T_j \mid j = 1, \ldots, i - 1]$. $S_i$ is the first release of task $\tau_i$ after (or at) $S_{i-1}$. Since the tasks are ordered by priority, the periodicity of the first ones is unchanged by the requests of task $\tau_i$ and since the schedule is assumed to be feasible, the schedule repeats at time $S_i + \text{lcm}[P_{i-1}, T_i]$. Hence, for the task set $\{ \tau_1, \ldots, \tau_i \}$ the schedule repeats from $S_i$ with period $P_i = \text{lcm}[P_{i-1}, T_i]$.

It may be noticed that, while $P$ is indeed the true period of the periodic part of the schedule (it is also the period for the relative phasing of successive requests of the various tasks), it is not said that $S_1$ is the earliest point from which the
periodicity occurs; even for the first level, i.e., task \( \tau_1 \) alone, we could have started from \( S'_1 = O_1 - (T_1 - C_1) \), since the idle phase from \( S'_1 \) to \( S_1 \) corresponds to the one from \( S_1 + C_1 \) to \( S_1 + T_1 \); and similarly, if the schedule for \( \tau_2 \) leaves an idle period for it of length \( \delta_2 \) before the instant \( S_2 + \lcm\{T_1, T_2\} \), we could replace \( S_2 \) by \( S' = \max\{S_1, S_2 - \delta_2\} \).

From a schedulability point of view, the first part of a schedule constructed by a given priority assignment may be neglected; we may only consider the schedule from its periodic part, in other words, in 'regime' situation. In fact, from a schedulability point of view, the regime situation is worse than the initial situation:

**Lemma 1** ([5]) At any instant \( t \geq O_1 \), let \( \varepsilon_i(t) \) be the amount of processor time used by \( \tau_i \) since its last request; we have that \( \forall i \in \{1, \ldots, n\}, \forall t \in \mathbb{N}, t - k \cdot P \geq O_i \implies \varepsilon_i(t - k \cdot P) \geq \varepsilon_i(t) \).

**Proof:** We prove the lemma by contradiction: suppose there is some task \( \tau_j \) and instant \( t_1 \) such that \( \varepsilon_j(t_1) < \varepsilon_j(t_2) \) where \( t_2 = t_1 + k \cdot P \). In this case, there must exist some time \( t'_1 < t_1 \) such that \( \tau_j \) is active at both \( t'_1 \) and \( t'_2 = t'_1 + k \cdot P \) (i.e., \( \varepsilon_j(t'_1) < C_j \) and \( \varepsilon_j(t'_2) < C_j \)), and \( \tau_j \) is executing at \( t'_1 \) but not at \( t'_2 \). This can only occur if there is another task \( \tau_i \) (\( \tau_i > \tau_j \)) which is active at \( t'_1 \) but not at \( t'_2 \) (i.e., \( \varepsilon_i(t'_1) < C_i \) and \( \varepsilon_i(t'_2) = C_i \)). But this means that \( \varepsilon_i(t'_1) < \varepsilon_i(t'_2) \) and thus repeating the above argument, we have: \( \varepsilon_k(t'_1) < \varepsilon_k(t'_2) \), etc. contradicting the fact that we have a finite number of tasks in the system.

As a consequence, the response time of requests during the periodic part of the schedule is worse than the corresponding ones in the initial phase, and if no deadline is missed during a period, the same is true in general. As a result of Theorem 7, we may limit a simulation to the interval \([0, S_n + P]\), or that \([0, S_n + P]\) is a feasibility interval, but the lower boundary of this interval can be improved.

**Definition.** We define the partial schedule \( \sigma_t \) of a system to be the schedule obtained by only considering, in the schedule of the system, instants greater than \( t \).

**Lemma 2.** Let \( S = \{O_i, C_i, D_i, T_i\} | i = 1, \ldots, n \} \) be a feasible asynchronous system with the priority assignment \( \tau_1 > \tau_2 > \cdots > \tau_n \), and \( t \) be a time instant. Let \( X_j \) inductively defined by \( X_{n+1} = t \), \( X_i = O_i + \left\lfloor \frac{X_{i+1}-O_i}{T_i} \right\rfloor T_i \) for \( i = n, n-1, \ldots, 1 \). If \( X_{i+1} \geq O_i \) (\( 1 < i \leq n \)) then the partial schedule \( \sigma_t \) only depends on the requests of \( \tau_j \) (\( 1 \leq j \leq n \)) which follow the time \( X_j \).

**Proof:** We shall demonstrate the property by (descending) induction on \( j \). The property is true in the trivial case, where \( j = n \); in the schedule \( \sigma_t \) we have only to consider the request of \( \tau_n \) from time \( X_n = O_n + \left\lfloor \frac{X_{n+1}-O_n}{T_n} \right\rfloor T_n \) (the time of the last request of \( \tau_n \) which occurs before or at \( t \); this request exists since \( t = X_{n+1} \geq O_n \)). Indeed, the requests of \( \tau_n \) which occur strictly before time \( X_n \) are terminated at time
\(X_i\) and do not have any impact on \(\sigma_i\) (in particular, they have no impact on higher priority tasks). Assume that the property is true for the tasks \(\tau_1, \tau_{i-1}, \ldots, \tau_{j+1}\) and let us consider the requests of task \(\tau_j\). The requests which occur before time \(X_j = O_j + \left[\frac{X_{j+1} - O_j}{T_j}\right] T_j\) (the time of the last request of \(\tau_j\) which occurs before or at \(X_{j+1}\); this request exists since \(X_{j+1} \geq O_j\) are terminated before \(X_j\) and do not impact on the requests of \(\tau_j\) after time \(X_k\) \((k = j + 1, \ldots, n)\), since \(X_j \leq X_{j+1} \leq \cdots \leq X_{n+1}\), nor of course on the schedule of higher priority tasks. The property that \(X_j \leq X_{j+1}\) follows from the definition of \(X_j\), the properties of the function \([\_]\) and from \(X_{j+1} \geq O_j\). Moreover, dropping some or all of these useless requests (from \(\sigma_j\)'s point of view) may not render the schedule infeasible. Hence, the schedule \(\sigma_j\) only depends on the requests of \(\tau_j\) which occur after time \(X_j\).

**Corollary 1** Let \(S = \{\tau_i = \{O_i, C_i, D_i, T_i\}\} ii = 1, \ldots, n\} be a feasible asynchronous system with the priority assignment \(\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n\), and \(t\) be a time instant. Let \(X_j\) be inductively defined by \(X_{n+1} = t\), \(X_i = O_i + \left[\frac{X_{i+1} - O_i}{T_i}\right] T_i\), for \(i = n, n-1, \ldots, 1\). If \(t \geq O_{\max} + \sum_{k=2}^{n}(T_k - 1)\) then the partial schedule \(\sigma_i\), only depends on the requests of \(\tau_j\) \((1 \leq j \leq n)\) from time \(X_j\).

**Proof:** The following property holds:
\[t \geq O_{\max} + \sum_{k=2}^{n}(T_k - 1) \Rightarrow X_{j+1} \geq O_{\max} + \sum_{k=2}^{j}(T_k - 1) \geq O_j\]. By induction on \(j\). The property is true if \(j = n\): \(X_{n+1} = t \geq O_{\max} + \sum_{k=2}^{n}(T_k - 1) \geq O_n\). Assume that the property is true until \(j + 1\): \(X_{j+1} \geq O_{\max} + \sum_{k=2}^{j}(T_k - 1) \geq O_j\). Assume that \(0 \leq (X_{j+1} - X_j) < T_j\) follows from the definition of \(X_j\), the properties of the function \([\_]\) and from \(X_j \geq O_j\). Hence \(X_j \geq O_{\max} + \sum_{k=2}^{j}(T_k - 1) \geq O_{j+1}\). Therefore, \(X_{j+1} \geq O_j(1 \leq j \leq n)\) and the corollary follows from Lemma 2.

**Corollary 2** Let \(S = \{\tau_i = \{O_i, C_i, D_i, T_i\}\} ii = 1, \ldots, n\} be a feasible asynchronous system with the priority assignment \(\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n\), and \(t\) be a time instant. Let \(X_j\) inductively defined by \(X_{n+1} = t\), \(X_i = O_i + \left[\frac{X_{i+1} - O_i}{T_i}\right] T_i\), for \(i = n, n-1, \ldots, 1\). If \(X_{i+1} \geq O_i\) \((0 < i \leq n)\) then \(\forall t' \geq t\) the partial schedule \(\sigma_{t'}\), only depends on the requests of \(\tau_j\) \((1 \leq j \leq n)\) from time \(X_{t'}\). \(X_j\) are inductively defined by \(X_{j+1} = t'\), \(X_i = O_i + \left[\frac{X_{i+1} - O_i}{T_i}\right] T_i\), for \(i = n, n-1, \ldots, 1\).

**Proof:** The following property holds:
\(X_{t'} \geq X_j\). By induction. The property is true in the trivial case: \(t' = X_{n+1} = t\). Assume that the property is true until \(j + 1\): \(X_{j+1} \geq X_{j+1}\). It is easy to see that the last request of \(\tau_j\) which occurs before or at time \(X_{j+1}\) occurs before or at the last request of \(\tau_j\) before or at \(X_{j+1}\). Therefore \(X_{j+1} \geq O_j(1 \leq j \leq n)\) and the corollary follows from Lemma 2.
Corollary 3 Let $S = \{\tau_i = (O_i, C_i, D_i, T_i)| i = 1, \ldots, n\}$ be a feasible asynchronous system with the priority assignment $\tau_1 > \tau_2 > \cdots > \tau_n$, and $t$ be a time instant. Let $X_j$ inductively defined by $X_{n+1} = t$, $X_i = O_i + \left\lfloor \frac{X_i - \frac{S_i}{T_i}}{P_i} \right\rfloor T_i$, for $i = n, n-1, \ldots, 1$. Let $S_i$ be inductively defined by $S_i = O_i$, $S_i = \max\{O_i, O_i + \left\lfloor \frac{S_i - \frac{O_i}{T_i}}{P_i} \right\rfloor T_i\} \ (i = 2, 3, \ldots, n)$. If $t \geq S_n$ then the partial schedule $S_i$, only depends on the requests of $\tau_j (1 \leq j \leq n)$ from time $X_j$.

Proof: The following property holds: $X_{j+1} \geq S_j \geq O_j (1 \leq j \leq n)$. The property is true in the trivial case, where $j = n$: $t = X_{n+1}$ by hypothesis and $S_n \leq O_n$ according to the properties of the values $S_i$ (see Theorem 7). Assume that the property is true until $j + 1$: $X_{j+1} \geq S_j \geq O_j$. $S_j$ is the first request of $\tau_j$ after or at $S_{j-1}$; hence $X_j$, the last request of $\tau_j$ before or at $X_{j+1}$, is certainly after or at $S_{j-1}$, while is itself after or at $O_{j-1}$. Therefore, $X_{j+1} \geq O_j (1 \leq j \leq n)$ and the corollary follows from Lemma 2.

Theorem 8 Let $X_i$ be inductively defined by $X_n = S_n$, $X_i = O_i + \left\lfloor \frac{X_n - \frac{S_n}{T_n}}{P_i} \right\rfloor T_i$ $(i = n - 1, n - 2, \ldots, 1)$; then $[X_1, S_n + P]$ is a feasibility interval. Moreover, for each $\tau_i$ one only has to check the deadlines in the interval $[S_i, S_i + \text{lcm}\{T_j | j \leq i\}]$.

Proof: Lemma 1 means that the load before $S_n$ is certainly less than or equal to the one during the period $[S_n, S_n + P]$; hence, if no deadline is missed between $S_n$ and $S_n + P$, this will also be the case before $S_n$, since the requests are fulfilled earlier there. As a consequence, the load before $S_n$ is only necessary to lead the system to its periodic behavior from $S_n$ (or earlier). By Corollary 3 we have that the periodic behavior of the system $\sigma_{S_n}$, only depends on the requests of $\tau_j$ from time $X_j$.

3. Optimality for asynchronous systems

In this section we study the optimality of the monotonic priority assignments for systems where the tasks are not synchronized, but where the offsets are fixed: asynchronous systems. First, let us define what optimality means in this case.

Definition. A priority assignment rule is optimal for asynchronous systems if when a feasible priority assignment exists for some asynchronous task set, the priority assignment given by the rule is also feasible for that task set.

Leung and Whitehead have shown that the rate monotonic scheduler is not optimal for asynchronous systems:

Lemma 3 ([5]) The rate monotonic priority assignment is not optimal for asynchronous systems.

Proof: This can be seen with the following system (introduced by Leung and Whitehead): $\tau_1 = \{C_1 = 1, T_1 = D_1 = 12, O_1 = 10\}, \tau_2 = \{C_2 = 6, T_2 =$
Figure 3. The task set is schedulable with $\tau_3 \succ \tau_2 \succ \tau_1$: at $t = 34$ the situation is the same as at $t = 10$ and the schedule repeats.

Figure 4. The task set is not schedulable with $\tau_3 \succ \tau_1 \succ \tau_2$: the first request of $\tau_2$ fails.

$D_2 = 12, O_2 = 0$, $\gamma_3 = \{C_3 = 3, T_3 = D_3 = 8, O_3 = 0\}$. This system can be scheduled with priority assignment ($\tau_3 \succ \tau_2 \succ \tau_1$) (see figure 3) while the rate monotonic priority assignment ($\gamma_3 \succ \tau_1 \succ \tau_2$) is not feasible (see figure 4).

**Corollary 4** The deadline monotonic priority assignment is not optimal for asynchronous systems.

**Proof:** This results immediately from the previous example since, when $D_i = T_i$ $\forall i$, deadline monotonicity coincides with rate monotonicity.

This example, largely used in the literature [3, 2] to illustrate the non-optimality of the rate monotonic priority assignment for asynchronous systems raises two points.

1. Both priority assignments ($\gamma_3 \succ \tau_2 \succ \tau_1$) and ($\gamma_3 \succ \tau_1 \succ \tau_2$) are rate monotonic priority assignments. In this case, the non-optimality of the rate monotonic priority assignment is due to the choice made to resolve the tie between $T_1$ and $T_2$. Hence, the example shows more precisely that, in case of ambiguity, the rate/deadline monotonic assignment is non-optimal for asynchronous systems; but nothing can be inferred at that point for non-ambiguous situations. Subse-
Figure 5. Since \(O_1 = O_2\) and \(T_1 = T_2 = D_1 = D_2\), the tasks \(\tau_1\) and \(\tau_2\) are like a simple task 
\[
\tau' = \{C' = C_1 + C_2 = \tau_1T' = D' = T_1 = T_2 = 12, O' = O_1 = O_2 = 2\};
\]
at time \(t = 26\) the situation is like at \(t = 2\) and the schedule repeats.

1. Consequently, for this reason, we shall only consider non-ambiguous situations where all periods are distinct.

2. We cannot conclude from it that the rate/deadline monotonic priority assignment is not optimal for the offset free systems; if we choose in the previous example \(O_1 = O_2 = 2\) and \(O_3 = 0\), with both rate monotonic priority assignments \((\tau_3 > \tau_2 > \tau_1)\) and \((\tau_3 > \tau_1 > \tau_2)\) the system becomes schedulable (see figure 5). The optimality analysis of asynchronous systems cannot thus be applied directly to the offset free systems.

4. Optimality for offset free systems

We have seen in the previous section that the optimality problem for offset free systems is of a different nature in comparison to asynchronous systems, due to the fact that the offsets can be chosen in order to schedule these systems.

4.1. Definitions and properties

The definition of the optimality for offset free systems must be redefined:

Definition. A priority assignment rule is optimal for offset free systems if when a feasible priority assignment \((\mu)\) and offset assignment \((\rho)\) exist for some offset free task set, there is a feasible offset assignment \((\rho')\) for the priority assignment given by the rule.

The definitions of optimality for offset free systems and for asynchronous systems are close in fact the optimality for offset free systems is more general than the optimality for asynchronous systems. Indeed, if in the definition of the optimality for offset free systems we required that both offset assignments \(\rho\) and \(\rho'\) must be identical, we get the definition of the optimality for asynchronous systems and in this case the offsets given by the assignment \(\rho = \rho'\) are the offsets of the corresponding asynchronous system. Hence, the optimality for asynchronous systems implies the optimality for offset free systems, in the general case, and conversely the non-optimality of a rule for offset free systems implies it also for asynchronous systems.
systems. We have seen the non-optimality of the rate/deadline monotonic assignments for (ambiguous) asynchronous systems, but in some special cases we may get optimality results, both for asynchronous and offset free systems.

4.2. Optimality in special cases of offset free systems

Lemma 4 If an assignment rule is optimal for a subclass of asynchronous systems and the definition of the subclass does not rely on special restrictions about the offsets, the same assignment rule is also optimal for the corresponding subclass of offset free systems.

Proof: Any counter-example for a subsequent would also be a counter-example for the corresponding antecedent. ■

Leung and Whitehead [5] have identified two special cases where the deadline monotonic priority assignment is optimal for asynchronous systems, hence by Lemma 4 in these special cases, the deadline and the rate monotonic priority assignment are optimal for offset free systems.

Corollary 5

1. The deadline monotonic priority assignment is optimal for offset free systems having only two tasks.

2. The rate monotonic priority assignment is optimal for offset free systems with late deadlines, having only two tasks.

3. The deadline monotonic priority assignment is optimal for offset free systems satisfying the conditions that $T_i$ is an integral multiple of $T_j$ ($T_i = m_{ij}T_j$, $m_{ij} \in \mathbb{N}$) when $T_i > T_j$.

4. The rate monotonic priority assignment is optimal for offset free systems with late deadlines, satisfying the conditions that $T_i$ is an integral multiple of $T_j$ when $T_i > T_j$.

But we may also devise another special case (Theorem 9), the following lemma is used for proving it.

Lemma 5 Let $S = \{\tau_1 = \{T_i, C_i, D_i, O_i\}; i = 1, \ldots, n\}$ be an asynchronous system and $\tau_1 > \tau_2 > \cdots > \tau_n$ be a priority assignment. The cpu allocation for any request of task $\tau_i$ depends on the task $\tau_j$ and the subset $S_h = \{\tau_1, \tau_2, \cdots, \tau_{i-1}\}$ of all higher priority tasks than $\tau_i$ but not on their relative priorities.

Proof: By induction on $n$. The property is true in the trivial case where $n = 1$; the cpu allocation on task $\tau_1$ depends only on $\tau_1 = \{T_1, C_1, D_1, O_1\}$: for any time $t \geq O_1$, $\tau_1$ is active (and running) iff $t - \left\lfloor \frac{O_1}{T_1} \right\rfloor T_1 + O_1 \leq C_1$. Consider now the
case of task $\tau_i$ ($i > 1$), for any time $t$ where $\tau_i$ is active, the CPU is allocated to task $\tau_i$ if all tasks in the set $S_i = \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ are inactive (i.e., at time $t$, all requests of tasks in the set $S_i$ are finished), which by induction does not depend on the relative priorities of the latter tasks.

Lemma 5 shows that the schedulability of task $\tau_i$ depends on the set of all higher priority tasks whatever the priority assignment in this set.

**Theorem 9** The rate monotonic priority assignment is optimal for asynchronous systems with late deadlines satisfying the condition that all the periods are distinct and $\left\lceil \frac{x}{T_j} \right\rceil \geq 2$ whenever $T_i > T_j$.

**Proof:** We must prove that if a feasible priority assignment exists for any task set satisfying the given condition, the rate monotonic priority assignment is also feasible for that task set. Let $\tau_1, \ldots, \tau_n$ be a set of $n$ such tasks with a feasible priority assignment ($\tau_1 > \tau_2 > \cdots > \tau_j > \tau_j > \cdots > \tau_n$). Let $\tau_i$ and $\tau_j$ be two tasks of adjacent priorities ($\tau_i > \tau_j$). Suppose that $T_i > T_j$. Let us exchange the priorities of $\tau_i$ and $\tau_j$: if the task set is still schedulable, since the rate monotonic priority assignment can be obtained from any priority ordering by a sequence of such priority exchanges, we may deduce that the rate monotonic priority assignment is schedulable.

The priority exchange does not modify the schedulability of the tasks with a higher priority than $\tau_j$ ($\tau_j \forall j < i$). The task $\tau_j$ remains of course schedulable after the priority exchange, since it may use all the free slots left by $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$ instead of only those left by $\{\tau_1, \tau_2, \ldots, \tau_{i-1}, \tau_j\}$. For any task $\tau_k$ with $k > j$, the set of all higher priority tasks than $\tau_k$ is not altered by this priority exchange, and by Lemma 5 this task remains schedulable. Thus we must only verify that $\tau_i$ also remains schedulable. To prove this, let us consider any request of $\tau_i$ (at time $x$) and the previous request of $\tau_j$ (at time $y$; from Theorem 4 we may assume that all tasks have already started their work). Since $\left\lceil \frac{x}{T_i} \right\rceil \geq 2$, we have necessarily at least one $\tau_i$’s request completely included in the $\tau_j$’s request: $x < y + T_i \leq y + 2T_j \leq x + T_j$ (see figure 6). Before the priority exchange either the $\tau_j$ request at time $y$ is not completed at time $x$ (case 1) or this request is completed at time $x$ (case 2).

1. The $\tau_j$ request at time $y$ is not completed at time $x$, let $C_j$ be the remaining process time at time $x$ for this request of $\tau_j$. In this case in the interval $[x, y + T_j]$
the $C_j + C_i$ first free units of cpu are consumed by $\tau_i$ (first) and $\tau_j$, and since the schedule is feasible both requests are fulfilled at $y + T_j$. In the interval $[y + T_j, x + T_i]$ $\tau_j$ is not pre-empted by $\tau_i$ since the last request of $\tau_i$ is completed. After the priority exchange, let $C'_j$ be the remaining process time at time $x$ for the request of $\tau_j$ at time $y$: $C'_j \leq C_j$ since $\tau_j$ is no longer pre-empted by $\tau_i$ in $[y, x]$; hence $\tau_i$ is schedulable in the interval $[x, y + T_j]$ since only $C'_j + C_i$ free units of cpu are needed by $\tau_j$ and $\tau_i$; since we know there are at least $C'_j + C_i$ free units in there, no problem may occur in $[x, y + T_j]$, since there is $(C_j - C'_j)$ more free cpu time left. In the interval $[y + T_j, x + T_i]$ the situation is the same as before since, $\tau_j$ was not pre-empted by $\tau_i$ there.

2. The request of $\tau_j$ at time $y$ is completed when reaching time $x$. In this case, before the priority exchange, in the interval $[x, y + 2T_j]$ the $C_j$ first free units of cpu are consumed by $\tau_i$ ($C_i < y + 2T_j - x$) and after that, from $y + T_j$ to $y + 2T_j$, there is still at least $C_j$ free units available for $\tau_j$. In the interval $[y + 2T_j, x + T_i]$ the task $\tau_j$ is not pre-empted by $\tau_i$ since the request of $\tau_i$ at $x$ is finished before $y + 2T_j$. Then, it is not difficult to see that, after the priority exchange, $\tau_i$ is still schedulable (its request at $x$ will be fulfilled by $y + 2T_j$).

\[\square\]

**Corollary 6** The rate monotonic priority assignment is optimal for offset free systems with late deadlines satisfying the condition that all the periods are distinct and $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 2$ whenever $T_i > T_j$.

**Proof:** Immediate from Theorem 9 and Lemma 4. \[\square\]

### 4.3. Non-optimality in the general case of offset free systems

In the definition of the optimality of assignment rules for offset free systems, we considered offset assignments and the schedulability of the corresponding asynchronous systems for the assigned priorities; a priori, there is an infinite number of offset assignments, thus yielding a direct check impractical; now we refine the offsets to be considered for a task set.

**Lemma 6** In the definition of the optimality of an assignment rule for offset free systems, we may restrict the offsets in such a way that:

1. $O_1 = \min\{O_1, \ldots, O_n\} = 0$
2. $O_i \in [0, T_i)$ for all $1 < i \leq n$

**Proof:** Theorem 4 and 8 show that the feasibility of an asynchronous task set, for any static priority assignment, only depends on the relative phasing of the successive
requests of each task when they are all started, and adding/subtracting a multiple of \( T_j \) to \( O_j \) does not change this long term phasing, nor adding/subtracting a common delay to all offsets; hence the property. \[\blacklozenge\]

Hence, for a task set \( \tau_1, \ldots, \tau_n \), we have at most \( \prod_{i=2}^{n} T_i \) combinations of possible offsets which define different asynchronous systems with respect to long term relative phasing.

In order to prove our next result, we have to introduce the notion of the relative phasing between two requests.

**Definition.** Let \( \tau_i \) and \( \tau_j \) be two tasks with \( \tau_i > \tau_j \). For the \( k\text{th} \) request of \( \tau_j \) (which occurs at time \( O_j + (k-1)T_j \)), we define \( \Delta_{i,j}(k) \), the relative phasing between \( \tau_i \) and the \( k\text{th} \) request of \( \tau_j \), as the difference between \( O_j + (k-1)T_j \) and the time of the previous request of \( \tau_i \) (assuming there is one).

It may be checked that:

\[
\begin{align*}
\Delta_{i,j}(k) &= (O_j - O_i + (k-1)T_j) \mod T_i \\
\Delta_{i,j}(k+1) &= (\Delta_{i,j}(k) + T_j) \mod T_i
\end{align*}
\]

Hence, the values \( \Delta_{i,j}(k) \) for successive values of \( k \), form a cycle of length \( q \):

\(<\Delta_{i,j}(k), \Delta_{i,j}(k+1), \ldots, \Delta_{i,j}(k+q)>\) with \( \Delta_{i,j}(k+d) = \Delta_{i,j}(k+d \mod q) \) and \( q = \frac{\text{gcd}(T_i, T_j)}{T_j} \).

**Example:** Consider the task set composed by two tasks \( S = \{ \tau_1 = \{T_1 = 4, O_1 = 0\}, \tau_2 = \{T_2 = 5, O_2 = 0\} \} \); the relative phasings between \( \tau_1 \) and \( \tau_2 \) form the cycle:

\(<\Delta_{1,2}(1) = 0, \Delta_{1,2}(2) = 1, \Delta_{1,2}(3) = 2, \Delta_{1,2}(4) = 3>\) (see figure 7). \[\blacklozenge\]

![Figure 7](image)

**Figure 7.** Relative phasings between \( \tau_1 \) and \( \tau_2 \).

Now, we are able to solve the problem of the optimality of the rate monotonic priority assignment for offset free systems:

**Theorem 10.** The rate monotonic priority assignment is not optimal for the offset free systems with late deadlines.
OFFSET FREE SYSTEMS

\[\begin{align*}
\tau_1 \quad & \quad \tau_2 \\
\tau_3 \\ & \quad \tau_2 \\
\tau_4 \\
\tau_5 \\
\tau_6 \\
\end{align*}\]

Figure 8. The system is schedulable in the interval \([0, 244]\); at time \(t = 244\) the situation is like at time \(t = 4\).

**Proof:** To show this, we consider a task set (with distinct periods) which is a counter-example for the definition of the optimality of a priority assignment rule for offset free systems: \(S = \{\tau_1, \tau_2, \tau_3\}\) with \(\tau_1 = \{T_1 = D_1 = 10, C_1 = 7\},\)
\(\tau_2 = \{T_2 = D_2 = 15, C_2 = 3\}, \tau_3 = \{T_3 = D_3 = 16, C_3 = 1\}.

1. With the priority assignment \(\tau_1 > \tau_2 > \tau_3\) and the offsets \(O_1 = O_3 = 0\) and \(O_2 = 4\) the system is schedulable. To verify this, from Theorem 8, we only have to check the interval \([4, 244]\) in a simulation from \(0\) (see figure 8; \(X_1 = 0, S_2 = 4, P = 240\)).

2. For the unique rate monotonic priority assignment \(\tau_1 > \tau_2 > \tau_3\) and for all offsets (according to Lemma 6, i.e., \(O_1 = 0, 0 \leq O_2 < 15, 0 \leq O_3 < 16\)) this system is unschedulable. This may be checked by a small computer program, but may also be proved formally as follows:

If we apply Theorem 8, we know that for \(\tau_3\) we only have to check the interval \([S_3, \text{km}(T_1, T_2, T_3) + S_3] = [S_3, 240 + S_3]\), in a simulation from \(X_1\) (or from \(0\); notice that \(S_3\) and \(X_1\) depend on the choice of these offsets); this interval contains 15 releases of \(\tau_3\). We shall show that, for all offsets, there is a release
of \( t_3 \) which misses its deadline in this interval. We can see that the relative phasings between \( t_1 \) and \( t_3 \) (\( \Delta_{13}(k) \)) are either in the cycle \( < 0, 6, 2, 8, 4 > \) or in the cycle \( < 1, 7, 3, 9, 5 > \) and the relative phasings between \( t_2 \) and \( t_3 \) (\( \Delta_{23}(k) \)) are in the cycle \( < 0, 1, 2, \ldots, 14 > \). Hence, after at most 4 releases of \( t_3 \) from \( S_3 \), we have a relative phasing between \( t_1 \) and \( t_3 \) equal to 0 or 1. Let \( r \) be the rank of the request of \( t_3 \) at time \( s_3 \), \( t_3 \) be the time of the request of \( t_3 \) with phasing 1 or 0, \( h \) be the rank of this request of \( t_3 \) and \( t_1 \) be the time of the last request of \( t_1 \) before or at \( t_3 \).

(A) If the relative phasing between \( t_1 \) and \( t_3 \) is 1 (\( \Delta_{13}(h) = 1 \), \( r \leq h \leq r + 4 \)), i.e., \( t_3 = t_1 + 1 \), we see that \( t_3 \) is schedulable only if some CPU unit is free in the interval \([t_1 + 7, t_1 + 10]\); this may only be the case if there is no request for \( t_2 \) arriving between \( t_1 \) and \( t_1 + 7 \), i.e. \( \Delta_{23}(h) \in \{2, 3, 4, 5, 6, 7, 8\} \) (otherwise, the 3 units before \( t_1 + T_1 \) are used by \( t_2 \), see figure 9). If \( \Delta_{23}(h) \geq 4 \) then 5 releases of \( t_3 \) later, the relative phasing between \( t_1 \) and \( t_3 \) is necessary the same (\( \Delta_{13}(h+5) = 1 \)) and the relative phasings between \( t_2 \) and \( t_3 \) satisfies the condition: \( \Delta_{23}(h+5) = (\Delta_{23}(h)+5\times16) \mod 15 \in \{9, 10, 11, 12, 13\} \), hence \( t_3 \) misses its deadline. If \( \Delta_{23}(h) < 4 \) then 10 releases of \( t_1 \) later, the relative phasing between \( t_1 \) and \( t_3 \) is necessary the same (\( \Delta_{13}(h+10) = 1 \)) and the relative phasing between \( t_2 \) and \( t_3 \) satisfies the condition: \( \Delta_{23}(h+10) = (\Delta_{23}(h)+10\times16) \mod 15 \in \{12, 13\} \), hence still \( t_3 \) misses its deadline.

(B) If the relative phasing between \( t_1 \) and \( t_3 \) is 0 (\( \Delta_{13}(h) = 0 \), \( r \leq h \leq r + 4 \)), for similar reasons \( t_3 \) is schedulable only if the relative phasings between \( t_2 \) and \( t_3 \) satisfies the condition: \( \Delta_{23}(h) \in \{1, 2, 3, 4, 5, 6, 7\} \). If \( \Delta_{23}(h) \geq 3 \) then 5 releases of \( t_3 \) later, the relative phasing between \( t_1 \) and \( t_3 \) is necessary the same (\( \Delta_{13}(h+5) = 0 \)) and the relative phasing between \( t_2 \) and \( t_3 \) satisfies the condition: \( \Delta_{23}(h+5) \in \{8, 9, 10, 11, 12, 13\} \), hence \( t_3 \) misses its deadline. If \( \Delta_{23}(h) < 3 \) then 10 releases of \( t_3 \) later, the relative phasing between \( t_1 \) and \( t_3 \) is necessary the same (\( \Delta_{13}(h+10) = 0 \)) and the relative phasing between \( t_2 \) and \( t_3 \) satisfies the condition: \( \Delta_{23}(h+10) \in \{11, 12\} \), hence still \( t_3 \) misses its deadline.
We have thus shown that in any cases before the \((r+15)^{th}\) release of \(r_5\), it misses its deadline.

Consequently we also have that:

**Corollary 7** The deadline monotonic priority assignment is not optimal for the offset free systems.

**Proof:** This results immediately from the fact that when \(D_i = T_i \ \forall i\), deadline monotonicity coincides with rate monotonicity.

The task set introduced in the proof of Theorem 10 shows that the non-optimality of the rate monotonic scheduler for asynchronous and offset free systems is not due to the choice to resolve the tie between two periods of tasks.

**Corollary 8** The rate monotonic priority assignment is not optimal for asynchronous systems with distinct periods.

**Proof:** Any example for the non-optimality for offset free systems may also serve for the non-optimality for asynchronous systems.

5. Practical interest of offset free systems

We have seen in the introduction the interest to consider systems where the offsets can be computed by the scheduling algorithm, and we have presented systems which are only schedulable with a judicious choice of the offsets. In this section we are concerned with the practical interest to compute the offsets. For this purpose, we shall consider the ratio between systems which are schedulable in the synchronous case and systems which are only schedulable in an asynchronous case, i.e., if we get some (randomly chosen) system, what is the probability to be in the first or in the second case? It is difficult to answer this question in all generality, since it depends on the real-time system itself, in particular on the system characteristics, i.e., the distribution of the number of tasks, the period values, the load of the systems,... It is not possible of course to consider all distributions of hard real-time periodic task sets. Moreover, it is hard to determine which distributions are (possibly) realistic.

However, the study of special cases of hard real-time periodic task sets can give a good indication of the interest of offset free systems in practical cases. For this reason we have studied a special case: we consider the late deadline case and we suppose that the priority assignment is resolved with the rate monotonic scheduler.

In this special case of hard real-time periodic task sets, the offset free systems have an interest only if the utilization factor \(U = \sum_{i=1}^{n} \frac{1}{p_i}\) is greater than \(n(\sqrt{2} - 1)\), since in the opposite case the system is certainly schedulable in the synchronous case [16]. Hence, we only consider systems with \(1 \geq U > n(\sqrt{2} - 1)\). Each of these systems can be either schedulable in the synchronous case (case a), only in an asynchronous case (case b) or unschedulable in all asynchronous cases (case c). This
'classification' for a set of \( n \) tasks has a time complexity of \( O(\prod_{i=2}^{n} T_i \times R) \) where \( \prod_{i=2}^{n} T_i \) represents the maximal number of offset assignments to be considered and \( R \) is the time complexity of the schedulability test. We have seen that we can restrict the schedulability test to a finite interval but the length of this interval is proportional to \( P = \text{lcm}(T_i| i = 1, \cdots, n) \), which may grow exponentially with \( n \). Hence, the classification has an exponential complexity in \( n \) and in the maximal period (\( \max_{i=1, \cdots, n} T_i \)); consequently, we have strongly limited these values in our study. We present in this section simulation results of this classification applied to a large number of task sets. The task sets are generated with a pseudo-random algorithm; the number of tasks (\( n \)) is chosen with equal probability in an interval \([3, 10]\), the periods (\( T_i \)) in an interval \([10, 25]\). For each task set we have identified its class (case a, case b or case c). Figure 10 only concerns schedulable systems (i.e., systems in the class a or in the class b), the graph represents the ratio between the task sets in case b and in case a in function of the utilization factor.

![Graph showing the ratio between the number of task sets in the case b and in the case a.](image)

**Figure 10.** Ratio between the number of task sets in the case b and in the case a.

From the simulation results, it can be noticed that the practical interest of offset free systems is obvious, especially when the utilization factor is large. Indeed, the more the utilization factor is close to 1, the more the corresponding systems are only schedulable with a judicious choice of the offsets. For example, for an utilization factor of 0.9 we have about 60% of schedulable systems which are schedulable only in an asynchronous situation. Figure 11 is made of 3 graphs: each graph represents (in function of the utilization factor) the ratio between the number of task sets in a class (a, b or c) and the total number of task sets (e.g. \( \frac{\text{a}}{\text{a}+\text{b}+\text{c}} \)). Figure 11 shows that when the utilization factor is large the proportion of unschedulable task sets (case c) is large too, but the interest to compute the offsets remains relevant in that case in order to schedule a larger number of systems.
OFFSET FREE SYSTEMS

\[ u = \sum_{j=1}^{n} \frac{c_j}{T_j} \]

\[ \frac{a/\#(a\cup b\cup c)}{a/\#(a\cup b\cup c)} \]

\[ \frac{b/\#(a\cup b\cup c)}{b/\#(a\cup b\cup c)} \]

\[ \frac{c/\#(a\cup b\cup c)}{c/\#(a\cup b\cup c)} \]

Figure 11. Ratio between the number of task sets in the case a (or b or c) and the total number of task sets.

6. Conclusion and future work

In this paper we have studied the scheduling problem of hard real-time periodic tasks with static priority pre-emptive algorithms. We have shown the interest to consider systems where the offsets can be chosen by the scheduling algorithm.

We have studied the optimality of the rate (deadline) monotonic priority assignment for these systems. We have shown that the (non) optimality for offset free systems cannot be reduced to similar results for asynchronous systems. We have studied the relations between these two kinds of systems in the context of the rate (deadline) monotonic priority assignment optimality. We have shown that the rate (deadline) monotonic priority assignment is not optimal for offset free systems.

Therefore, a scheduling algorithm for offset free systems must compute a priority assignment and the offsets in order to schedule a larger number of task sets. It may be noticed that we can always build an optimal priority assignment and/or offset assignment; we can consider the first feasible assignment in an exhaustive search among the \( n! \) combinations for the priorities and \( \prod_{i=2}^{n} T_i \) for the offsets. Hence, the optimality and the complexity of the priority/offset assignment must be considered together. We have shown [3] that the number of combinations for offset assignment can be reduced significantly, with an exponential profit; however the number of combinations remains exponential. The next step of our work would be to define a rule to choose a single offset for each task in order to have a polynomial number of combinations. A (nearly) optimal priority assignment would be defined to schedule task sets that are not schedulable when all tasks are started at the same time, or when the rate/deadline monotonic schedules are adopted. This problem is relevant and should be solved with reasonable efficiency, but this remains for a future work.
Acknowledgments

We would like to thank an anonymous referee, whose remarks led us to better present our results.

References