Real-Time Systems, 5, 1–26 (1997)
© 1997 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Scheduling of Offset Free Systems

JOËL GOOSSENS joel.goossens@ulb.ac.be

Université Libre de Bruxelles, Département d'Informatique CP 212, Boulevard du Triomphe, 1050 Brussels, Belgium

Received November 25, 1995; Revised March 30, 1996

Editor:

Abstract. In this paper, we study the problem of scheduling hard real-time periodic tasks. We consider independent tasks which are characterized by a period, a hard deadline and a computation time, but where the offsets may be chosen by the scheduling algorithm.

We first show that we can restrict the problem by considering non-equivalent offset assignments. More precisely, we show that there are finitely many non-equivalent offset assignments and we propose a method to reduce significantly this number and consider only the minimal number of non-equivalent offset assignments. We then propose an optimal offset assignment rule which considers only the non-equivalent offset assignments. However the number of combinations remains exponential; for this reason, we also propose a nearly optimal algorithm with a more reasonable time complexity.

Keywords: hard real-time scheduling, periodic task set, synchronous systems, asynchronous systems, rate monotonic, deadline monotonic, deadline driven scheduler.

1. Introduction

Real-time systems are characterized by stringent timing constraints; hence, the correctness of a computation depends not only on its logical or computational results, but also on the instant when the result is made available. The most important feature of a real-time system is its *predictability* (Stankovic et al., 1990), i.e., the ability to determine whether the system is capable of meeting all its timing requirements. Examples of such systems include the control of engines, traffic, nuclear power plants, time-critical packet communications, aircraft avionics and robotics. Typically, time-sensitive computations are modelled as *tasks* that need to be executed by the system, and the timing constraint is represented by a *deadline* denoting the time-instant by which the job should complete execution. We distinguish between two kinds of deadlines: If meeting a task deadline is absolutely critical for the system correctness, then the deadline is said to be *hard*; missing a hard deadline is considered a definite failure, and leads to catastrophic consequences. If it is desirable to meet a deadline but missing it can be tolerated episodically, then the deadline is said to be *soft*.

In this paper, we consider the scheduling of periodic hard real-time tasks on a uniprocessor. The set is composed of n periodic tasks $\tau_1, \ldots, \tau_n$. Each periodic task τ_i is characterized by the quadruple (T_i, D_i, C_i, O_i) with $0 < C_i \leq D_i$ and $C_i \leq T_i$ and $O_i \geq 0$, i.e., by a period T_i, a hard deadline delay D_i, an execution requirement C_i, and an offset O_i, giving the instant of the first request. The requests of τ_i are separated by T_i time units and occur at time $O_i + (k-1)T_i$ $(k = 1, 2, \ldots)$. The execution time required for each request is C_i time units; C_i can be considered as the worst-case execution time for a request of τ_i. The execution of the k^{th} request of task τ_i, which occurs at time $O_i + (k-1)T_i$, must finish before or at time $O_i + (k-1)T_i + D_i$; the deadline failure is fatal for the system: the deadlines are considered to be hard. All timing characteristics of the tasks in our model of computation are assumed to be non-negative integers, i.e., they are multiples of some elementary time interval (possibly the "CPU tick", the smallest indivisible CPU time unit, but more often some conventional unit like the millisecond, or the like). The various tasks and requests will be assumed here to be independent (no common resource but the processor, no precedence constraints, ...).

From a theoretical as well as from a practical point of view, it is interesting to distinguish between three classes of periodic task sets, regarding the offsets: *synchronous*, *asynchronous* and *offset free* task sets. Synchronous systems correspond to the case where the offsets are fixed by the constraints of the system and they are all the same, e.g., $O_1 = \cdots = O_n = 0$. In asynchronous systems the offsets are fixed by the constraints of the system, but the tasks are not started at the same time: the offsets are different. In offset free systems there is no definite requirement about the task start times; hence, the offsets will be chosen beforehand by the scheduling algorithm itself. We shall consider specifically this kind of systems here.

We shall also distinguish between three classes of periodic task sets regarding the relation between the period and the deadline of each task: the *implicit deadline* case, the *constrained deadline* case and the *arbitrary deadline* case. Implicit deadline case corresponds to the case where the deadline of each task coincides with the period ($T_i = D_i,\ i = 1, \ldots, n$). In this case, each request must simply be completed before the next request (of the same task) occurs. Constrained deadline case corresponds to the case where the deadlines are not greater than the periods ($D_i \leq T_i,\ i = 1, \ldots, n$). Arbitrary deadline case corresponds to the case where no constraint exists between the deadline and the period: the deadline of a task τ_i may be less ($D_i \leq T_i$) or greater ($D_i > T_i$) than the period; in the latter situation, many requests of a same task may coexist at some instants.

We shall also distinguish between two scheduling algorithm families: *static* and *dynamic* schedulers. Static schedulers correspond to the case where the priorities are computed beforehand and then assigned to the tasks; during the execution, the system selects the highest priority active request. In general, we denote by $\tau_i > \tau_j$ the fact that task τ_i has a greater static priority than task τ_j. The most popular static schedulers are the rate (Liu et al., 1973) and the deadline monotonic schedulers (Leung et al., 1982) which are optimal for synchronous implicit deadline and synchronous constrained deadline systems, respectively. Dynamic schedulers correspond to the case where the priority of each request is computed during the execution of the system and consequently may change with time. The most popular dynamic schedulers are the deadline driven (or earliest deadline first) scheduler (Liu et al., 1973) and the least laxity first scheduler (Mok et al., 1978), which are optimal for all the classes of periodic task sets considered here: synchronous/asynchronous/offset free and implicit/constrained/arbitrary deadline systems.

For the various periodic task sets considered here regarding the relation between the deadline and the period and for the various scheduling algorithms introduced above a very general remark can be raised:

From a schedulability point of view the synchronous case is the worst case, in the sense that, if the system is schedulable in the synchronous case it follows that this is also the case in all asynchronous situations.

More precisely for implicit deadline systems with a static priority assignment Liu and Layland have shown (Liu et al., 1973) that the largest response time of τ_i occurs for its first request in the synchronous case. From (Liu et al., 1973) it is not difficult to see that the property remains valid for constrained deadline systems. For arbitrary deadlines with static priority assignments, Lehoczky has shown that the largest response time for each task still occurs in the synchronous case, but not necessarily for the first request: we have to consider the interval $[0, \lambda_n)$ where λ_n is the size of the first busy period (see (Lehoczky, 1990) for details). For dynamic priority rules, and especially for the deadline driven scheduler, the largest response times do not necessarily occur in the synchronous schedule, nor in the first busy period, but from (Liu et al., 1973, Baruah et al., 1993) it is easy to show that the synchronous case remains the worst case from a schedulability point of view.

Consequently, for static as well as dynamic schedulers, it is pessimistic to consider only the synchronous case, since a system can be unschedulable in the synchronous case, while being schedulable in a particular asynchronous situation. We present two such systems, first for the static case, then for the dynamic one.

EXAMPLE: Consider the task set composed of three tasks τ_1, τ_2 and τ_3; $\tau_1 = \{C_1 = 3, T_1 = D_1 = 8\}$, $\tau_2 = \{C_2 = 6, T_2 = D_2 = 12\}$, $\tau_3 = \{C_3 = 1, T_3 = D_3 = 12\}$. In the synchronous case the system is statically unschedulable, even with the optimal priority assignments given by the rate monotonic rule: $\tau_1 > \tau_2 > \tau_3$ (see Figure 1) or $\tau_1 > \tau_3 > \tau_2$: the first request of task τ_3 misses its deadline; in those figures, $\downarrow$ represents a task request, $\bigcirc$ a deadline and $\underset{a\quad b}{\overset{c}{\boxed{}}}$ an execution of c units between time units a and b, included; in the special case where $a = b$ we omit b in our representation. But with the offsets $O_1 = O_2 = 0$ and $O_3 = 10$, the system is schedulable with the priority assignment $\tau_1 > \tau_2 > \tau_3$ (see Figure 2) and with the priority assignment $\tau_2 > \tau_1 > \tau_3$ (see Figure 3). □

EXAMPLE: Let us consider the following system with the deadline driven scheduler and the task set $\tau_1 = \{C_1 = 2, T_1 = D_1 = 6\}, \tau_2 = \{C_2 = 5, T_2 = 8, D_2 = 6\}$.

If we choose $O_1 = O_2 = 0$, the system is unschedulable: the first request of τ_2 misses its deadline (see Figure 4). However, the system is schedulable with $O_1 = 0$ and $O_2 = 1$ (see Figure 5). $\square$

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), considering only the synchronous case is too pessimistic. The popularity of the synchronous schedulings seems to stem from the fact that it is generally much more easy and quick to check schedulability of a synchronous system than for an asynchronous one, and not from the efficiency of the synchronous schedulings. Hence an interesting problem arises: if the offsets are not fixed by the constraints of the problem and the task set is unschedulable in the synchronous case, is there an assignment of the offsets (and priorities) such that it becomes schedulable? We call this kind of system: *offset free system.* We have already shown (Goossens et al., 1997) the interest of offset free systems and the non-optimality of monotonic priority assignments (notice that this non-optimality cannot be reduced to similar results for asynchronous systems). We shall elaborate a bit further on the subject here and present two scheduling rules to choose the offsets; the first one is optimal but its computational cost is generally unreasonable for large size systems; hence we present also a nearly optimal heuristic scheduling rule: the dissimilar offset assignment.

The remainder of this paper is organized as follows. In Section 2, we examine the granularity of the offsets and conclude that we can restrict the offsets to have the same granularity as the fixed characteristics (i.e., non-negative integers). In Section 3, we study the notion of (non-)equivalent asynchronous systems. We consider also the particular case of asynchronous systems which are equivalent to their synchronous case in Section 3.1. In Section 3.2 we enumerate the non-equivalent asynchronous situations. In Section 4, we propose an optimal scheduling rule for offset free systems which considers (in the worst case) all the non-equivalent asynchronous situations. More precisely, we first show that there are finitely many non-equivalent offset assignments and then we propose a method to reduce significantly this number and consider only the minimal number of non-equivalent offset assignments. Since the number of assignments of the optimal algorithms remains in general exponential we propose in Section 5 a nearly optimal algorithm with a more reasonable time complexity. In section 5.1 we present the evaluation of our heuristic rule and exhibits its good performance.

2. Offset granularity

We have assumed in our model of computation that the *fixed* characteristics of the various tasks (i.e., T_i's, D_i's, C_i's for offset free systems) are non-negative integers. Hence, since in this paper we consider real-time systems which have no requirement on the offset values, the latter may have a different granularity than the fixed characteristics and a priori it could be interesting to choose the offsets with a finer granularity. If a system is unschedulable for all non-negative integer offset assignments (i.e., $O_i \in \mathbb{N}$ for $i = 1, \ldots, n$), it is not obvious that this will still be the case if we allow, for instance, the offsets to be multiples of $\frac{1}{2}$ (i.e., $O_i \in \left\{\frac{p}{2} \mid p \in \mathbb{N}\right\}$ for $i = 1, \ldots, n$). It may be noticed that allowing the offsets to be multiple of $\frac{1}{2}$ for an offset free system $S = \left\{\tau_i = \{T_i, C_i, D_i\} \mid i = 1, \ldots, n\right\}$ with $T_i, C_i, D_i \in \mathbb{N}$ is equivalent to allow the offsets to be non-negative integers for the offset free system $S' = \left\{\tau'_i = \{T'_i = 2T_i, C'_i = 2C_i, D'_i = 2D_i\} \mid i = 1, \ldots, n\right\}$, where the fixed characteristics are multiplied by 2. Baruah and his colleagues (Baruah et al., 1990) have shown, for a computational model somewhat different from ours, that it is not relevant to have a finer granularity for the offsets. It may be noticed that we have adapted their results to the present framework see (Goossens, 1999) pp. 201–206 for details.

Definition 1. Let $S = \left\{\tau_i = \{T_i, C_i, D_i\} \mid i = 1, \ldots, n\right\}$ with $T_i, C_i, D_i \in \mathbb{N}$ be an offset free system. An offset assignment is said to have *a granularity of m* iff m is the smallest positive integer such that $O_i \in \left\{\frac{p}{m} \mid p \in \mathbb{N}\right\}$ $(i = 1, \ldots, n)$.

THEOREM 1 ((BARUAH ET AL., 1990, GOOSSENS, 1999)) *Let $S = \left\{\tau_i = \{T_i, C_i, D_i\} \mid i = 1, \ldots, n\right\}$ with $T_i, C_i, D_i \in \mathbb{N}$ be an offset free system with arbitrary deadlines. If S is not schedulable with the static priority assignment $\tau_1 > \tau_2 > \cdots > \tau_n$ for all non-negative integer offset assignments (i.e., $O_i \in \mathbb{N}$ for $i = 1, \ldots, n$), then this is also the case for all offset assignments with a granularity of m, for all m ($m \in \mathbb{N}_0 = \mathbb{N} \setminus \{0\}$).*

THEOREM 2 ((BARUAH ET AL., 1990, GOOSSENS, 1999)) *Let $S = \{\tau_i = \{T_i, C_i, D_i\} \mid i = 1, \ldots, n\}$ with $T_i, C_i, D_i \in \mathbb{N}$ be an offset free system with arbitrary deadlines. If S is not schedulable with the deadline driven scheduler for all non-negative integer offset assignments (i.e., $O_i \in \mathbb{N}$ for $i = 1, \ldots, n$), then this is also the case for all offset assignments with a granularity of m, for all m ($m \in \mathbb{N}_0$).*

Since both the deadline driven scheduler and the least laxity first scheduler are optimal it follows that Theorem 2 can be adapted to the least laxity first algorithm.

COROLLARY 1 *Let $S = \{\tau_i = \{T_i, C_i, D_i\} \mid j = 1, \ldots, n\}$ with $T_i, C_i, D_i \in \mathbb{N}$ be an offset free system. If S is not schedulable with the least laxity first algorithm for all non-negative integer offset assignments (i.e., $O_i \in \mathbb{N}, i = 1, \ldots, n$), then this is also the case for all offset assignments with a granularity of m, for all m ($m \in \mathbb{N}_0$).*

Consequently, in this paper we shall restrict without loss of generality the offsets to have the same granularity as the other task characteristics, i.e., to be non-negative integers, and we may even assume that $\gcd\{T_i, C_i, D_i \mid i = 1, \ldots, n\} = 1$.

3. Non-equivalent asynchronous systems

We shall in this section introduce the notion of (non-)equivalent asynchronous systems; this notion (and its interest) does not depend on the scheduling algorithm, not even on the scheduling family (i.e., static or dynamic). For this reason we shall present here results without considering a specific scheduling algorithm, but in particular circumstances. We suppose that the schedule is finally periodic with a period of $P \overset{\text{def}}{=} \text{lcm}\{T_1, \ldots, T_n\}$ time units (P is generally called the *hyper-period*), that only the periodic behavior is significant regarding the feasibility of the system, and that the periodic behavior only depends on the relative phasing of the task requests, i.e., on the tuple $(O_1 \pmod{T_1}, \ldots, O_n \pmod{T_n})$, which characterizes the relative time shifts between the requests of the various tasks, which is true under some non-restrictive assumptions for all the scheduling techniques we considered in this paper (see (Goossens et al., 1997) for static schedulers and (Goossens, 1999), pp. 207, for the deadline driven scheduler). We suppose also in the remainder of the paper that the scheduling policy does not require to examine a granularity

$m \geq 2$ (all the changes in the CPU allocation occur at integer instants), like static schedulers, deadline scheduling or least laxity scheduling as outlined in Section 2.

First let us define the schedule function, $\sigma(t)$, which specifies which task has the CPU at each instant. For constrained deadline systems, $\sigma(t)$ is an integer function and $\sigma(t) = j$, with $j > 0$, means that a request of task τ_j is executing at time t during (at least) one time unit, while $\sigma(t) = 0$ means that the CPU is idle at time t (during at least one time unit). For arbitrary deadline systems, where many requests of the same task may be active simultaneously, we shall assume that the oldest active request of τ_j receives the CPU, so that the same form may be used for the schedule function; notice that for the deadline driven or the least laxity first schedulers this condition is always satisfied; for static schedulers, it lifts in a common way the ambiguity. If another policy is desired, we shall then extend the notation with $\sigma(t) = (j, k)$ to represent the fact that at time t the k^{th} request of τ_j is executing.

Several asynchronous systems may lead to the same periodic behavior, and can be considered as *equivalent* in terms of feasibility according to the previous remark.

Definition 2. Let S and S' be two asynchronous arbitrary deadline systems: $S = \{\tau_i = \{O_i, C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$ and $S' = \{\tau'_i = \{O'_i, C'_i = C_i, D'_i = D_i, T'_i = T_i\} \mid i = 1, \ldots, n\}$. S and S' are said to be *equivalent* ($S \equiv S'$) if $\exists k_1, \ldots, k_n, A \in \mathbb{Z} : O_i = O'_i + k_i \cdot T_i + A \qquad (1 \leq i \leq n)$.

THEOREM 3 *Let S and S' be two equivalent asynchronous arbitrary deadline systems: $S = \{\tau_i = \{O_i, C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$ and $S' = \{\tau'_i = \{O'_i, C'_i = C_i, D'_i = D_i, T'_i = T_i\} \mid i = 1, \ldots, n\}$. Then S and S' have the same periodic behavior, i.e., $\exists t_1, a \in \mathbb{N}\ \forall t \geq t_1 : \sigma_S(t) = (i, k) \Leftrightarrow \sigma_{S'}(t + a) = (i, k + k_i)$ with $k_i \in \mathbb{Z}$ for $i = 1, \ldots, n$, (σ_S and $\sigma_{S'}$ being the schedule of S and S', respectively).*

Proof: The difference between S and S' lies in the offsets. The schedule of the system S can be obtained from the one of S' by adding or subtracting requests of task τ_i at times O_i and by changing the time origin. These transformations do not change the periodic behavior of the system S' from our assumptions. ■

3.1. Systems equivalent to the synchronous case

For the various kinds of periodic task sets considered here (regarding the relation between the deadline and the period and for the various scheduling algorithms considered), the feasibility problem of synchronous systems is a simpler problem in terms of time complexity, especially for static scheduling algorithms applied to constrained deadline systems. Hence, it is interesting to use feasibility tests defined for synchronous systems if the considered asynchronous system is equivalent to the synchronous one. Definition 2 can be simplified in this case.

Definition 3. Let S be an asynchronous and arbitrary deadline systems: $S = \{\tau_i = \{O_i, C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$ is said to be *equivalent to its synchronous case* if

$$\exists t, k_1, \ldots, k_n \in \mathbb{N} \text{ such that } \forall i: \; t = O_i + k_i \cdot T_i. \tag{1}$$

Let us determine how to check if an asynchronous system matches Equation (1). First, let us remark that Equation (1) may reformulated as follows:

$$\exists t \in \mathbb{N} \text{ such that } \forall i: \; t \equiv O_i \pmod{T_i}.$$

If the values T_i are pairwise prime this problem is known as the *Chinese Remainder Theorem* or the *Chinese Lemma*, which may be found for instance in (Knuth, 1969), pp. 249.

THEOREM 4 (CHINESE REMAINDER THEOREM) *Let $T_1, \ldots, T_n$ be positive integers which are relatively prime pairwise, i.e.,*

$$\gcd(T_i, T_k) = 1 \textit{ when } i \neq k.$$

Let $\mathcal{P} = T_1 \times T_2 \times \cdots \times T_n$, then the congruence system:

$$
\begin{aligned}
t &\equiv O_1 \pmod{T_1}\\
t &\equiv O_2 \pmod{T_2}\\
&\vdots\\
t &\equiv O_n \pmod{T_n}
\end{aligned}
$$

has exactly one solution modulo $\mathcal{P}$.

Consequently, if the periods are relatively prime (pairwise), the asynchronous system is always equivalent to the synchronous case. In the framework of this paper, we are only interested in the existence of a solution, but its construction can be found in (Knuth, 1969), pp. 250. In order to verify if the periods are pairwise relatively prime, we can apply the Euclid's algorithm to each pair (T_i, T_j), for $j > i$; the time complexity of this procedure is $\mathcal{O}(n^2 \times \log T^{max})$, where $T^{max} \overset{\text{def}}{=} \max\{T_i \mid i = 1, \ldots, n\}$.

We shall now consider the case where the periods are not relatively prime. Knuth gives a generalization of Theorem 4 to this case (see exercise 3, section 4.3.2 of (Knuth, 1969)).

Theorem 5 (Generalized Chinese Remainder Theorem) *Let $T_1, T_2, \ldots, T_n$ be positive integers. Let P be the least common multiple of $T_1, T_2, \ldots, T_n$ and let $a, O_1, O_2, \ldots, O_n$ be any integers. There is exactly one integer t which satisfies the conditions*

$$a \leq t < a + P, \qquad t \equiv O_j \pmod{T_j} \qquad 1 \leq j \leq n,$$

provided that

$$O_i \equiv O_j \pmod{\gcd(T_i, T_j)} \qquad 1 \leq i < j \leq n; \tag{2}$$

and there is no such integer t when the latter condition fails.

In the framework of this paper, we are only interested in the existence of a solution, but its construction can be found in (Knuth, 1969), pp. 513.

From Theorem 5 it follows that $O_i \equiv O_j \pmod{\gcd(T_i, T_j)}$, for $j > i$, is a necessary and sufficient condition for an asynchronous system to be equivalent to the corresponding synchronous one. Checking if an asynchronous system matches Equation (2) can again be resolved by applying the Euclid's algorithm to each pair (i, j), with a maximal time complexity $\mathcal{O}(n^2 \times \log T^{max})$.

3.2. Enumerating the non-equivalent asynchronous situations

We shall show in this section that there are $\frac{\prod_{i=1}^n T_i}{P}$ different classes of equivalent asynchronous situations (for the same values of $T_1, T_2, \ldots, T_n$), based on the relationship given by Definition 2. Regarding the scheduling of offset free systems, it is interesting to find these non-equivalent asynchronous situations in order to construct an optimal offset assignment but this problem is studied in the next section.

If we fix the periods, there is still an *infinite* number of asynchronous systems. However, we may first remark that without loss of generality we can restrict the offsets as follows.

THEOREM 6 *We may restrict the offsets in such a way that*

$$\begin{cases} O_1 = 0, \\ O_i \in [0, T_i) \qquad i = 2, \ldots, n. \end{cases}$$

without emptying any equivalence class of asynchronous systems.

Proof: This results immediately from our assumptions, in particular from the fact that the periodic part of the schedule is not altered[1] by suppressing the request of τ_i at time O_i and that the scheduling policy does not require to a examine granularity $m \geq 2$. ■

For the very same reason, we also get:

THEOREM 7 *We may restrict the offsets in such a way that they fulfill the* limited growing offset *property*

$$\begin{cases} O_1 = 0, \\ O_i \in [O_{i-1}, O_{i-1} + T_i) \qquad i = 2, \ldots, n. \end{cases}$$

without emptying any equivalence class of asynchronous systems.

It follows that the number of classes of equivalent asynchronous systems is finite and not greater than $\prod_{i=2}^{n} T_i$. In order to identify this number *exactly*, we shall base our study on the request separation time notion.

For the convenience of the presentation, let us introduce the notation $R_i^k \stackrel{\text{def}}{=} O_i + (k-1)T_i$, which corresponds to the release time of the kth request of τ_i.

Definition 4. Let $\Gamma = \{\tau_i = \{C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$ be an offset free task set. For the offset assignment $\vec{O} =< O_1, \ldots, O_n >$ and for all $k > 0$ such that R_1^k is in the periodic part of the system, we define the *request separation* for the kth request of τ_1 as $\vec{\Delta}(k, \vec{O}) =< \Delta_2(k, \vec{O}), \ldots, \Delta_n(k, \vec{O}) >$, where $\Delta_j(k, \vec{O})$ is the delay between R_1^k and the first request of τ_j which occurs after or at R_1^k, i.e., $\Delta_j(k, \vec{O}) = (O_j - R_1^k) \mod T_j$ (notice that $\Delta_1(k, \vec{O}) = 0$).

The request separation time is used here to compare asynchronous systems in terms of their relative phasings and check their equivalence.

THEOREM 8 *Let S and S' be two asynchronous arbitrary deadline systems: $S = \{\tau_i = \{O_i, C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$ and $S' = \{\tau_i' = \{O_i', C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$; $S \equiv S'$ iff $\exists k_1, k_2 \in \mathbb{N} : \vec{\Delta}(k_1, \vec{O}) = \vec{\Delta}(k_2, \vec{O'})$.*

Proof:
(if part). If $\vec{\Delta}(k_1, \vec{O}) = \vec{\Delta}(k_2, \vec{O'})$, it follows that $\forall j \geq 1$ we have $(O_j - R_1^{k_1}) \mod T_j = (O_j' - R_1'^{k_2}) \mod T_j$, hence $\exists r_j : O_j = O_j' + r_j T_j + (R_1^{k_1} - R_1'^{k_2})$ and $S \equiv S'$ from Definition 2.

(only if part). If $S \equiv S'$, $\exists k_1, \ldots, k_n, A \in \mathbb{Z} : O_i = O_i' + k_i \cdot T_i + A$ for $1 \leq i \leq n$. Let $k > 0$ such that R_1^k is in the periodic part of the system, $R_1^k = O_1 + (k-1)T_1 = O_1' - A - k_1 \cdot T_1 + (k-1)T_1$; consequently we have $O_i - R_1^k = O_i' + k_i \cdot T_i + A - (O_1' - A - k_1 \cdot T_1 + (k-1)T_1) = O_i' + k_i \cdot T_i + k_1 \cdot T_1 - (k-1)T_1 - O_1'$; it follows that for $k' equiv k - k_1 \pmod{T_1} : O_i - R_1^k = O_i' - R_1'^{k'} \pmod{T_i}$ and such a k' may be chosen in such a way that $R_1'^{k'}$ is in the periodic part of S'. The property follows.

■

Definition 5. Let $\vec{O_1}$ and $\vec{O_2}$ be two offset assignments. $\vec{O_1}$ and $\vec{O_2}$ are equivalent ($\vec{O_1} \equiv \vec{O_2}$) iff $\exists k_1, k_2 \in \mathbb{N} : \vec{\Delta}(k_1, \vec{O_1}) = \vec{\Delta}(k_2, \vec{O_2})$.

It follows from Definition 5 and Theorem 8 that equivalent offset assignments define equivalent asynchronous systems (and inversely).

LEMMA 1 *Let* $\Gamma = \{\tau_i = \{C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$. *There are* $\prod_{i=2}^{n} T_i$ *different request separations for any request (say the* k^{th}*) of* τ_1, *when the task periods are fixed and the offsets are free.*

Proof: Since $0 \leq \Delta_i(k, \vec{O}) < T_i$, we have that the number of different request separations is the number of different tuples $< x_2, x_3, \ldots, x_n >$ with $0 \leq x_i < T_i$ and $x_i \in \mathbb{N}$. ■

LEMMA 2 *Let* $\Gamma = \{\tau_i = \{C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$. *The offset assignment* $\vec{O}$ *defines* $\frac{P}{T_1}$ *equivalent and different request separations for any request of* τ_1.

Proof: The behavior of the system is periodic with a period of P (moreover, P is the smallest such period). Hence, the successive request separations for the requests of τ_1 $\vec{\Delta}(k, \vec{O}), \vec{\Delta}(k+1, \vec{O}), \ldots$ are also periodic, with a period $\frac{P}{T_1}$ (since the requests of τ_1 are separated by T_1 time units) and the interval $[R_1^k, R_1^k + P)$ contains $\frac{P}{T_1}$ request separations. We have also to prove that in the interval $[R_1^k, R_1^k + P)$ all the request separations for the requests of τ_1 are different. Suppose that this is not true: there exists $t_1 = R_1^k + p_1 T_1$ and $t_2 = R_1^k + p_2 T_1$ with $R_1^k \leq t_1 < t_2 < R_1^k + P$ such that

$$O_j - (R_1^k + p_1 T_1) \equiv O_j - (R_1^k + p_2 T_1) \pmod{T_j}$$

which implies that $p_1 T_1$ and $p_2 T_1$ are multiples of T_j ($j = 1, \ldots, n$), hence of $\mathrm{lcm}\{T_j \mid j = 1, \ldots, n\}$, but $0 < t_2 - t_1 < P$, a contradiction with the fact that $P = \mathrm{lcm}\{T_j \mid j = 1, \ldots, n\}$. ■

Now, we have the material to show the main result of this section, i.e., the exact number of non-equivalent asynchronous situations.

THEOREM 9 *Let* $\Gamma = \{\tau_i = \{C_i, D_i, T_i\} \mid i = 1, \ldots, n\}$. *There are* $\frac{\prod_{i=1}^{n} T_i}{P}$ *different equivalence classes of offset assignments.*

Proof: Let x be the number of such classes. By Lemma 1 and Lemma 2, we have that $x \cdot \frac{P}{T_1} = \prod_{i=2}^{n} T_i$. Hence, $x = \frac{\prod_{i=1}^{n} T_i}{P}$. ■

4. Optimal scheduling rule for offset free systems

For static priority assignments we have shown (Goossens et al., 1997) in previous works the non-optimality of the rate/deadline monotonic priority assignments and we have shown that this (non-)optimality cannot be reduced to similar results for asynchronous systems. It may be noticed that for asynchronous systems there is an optimal static priority assignment which considers $\mathcal{O}(n^2)$ distinct priority assignments, see (Audsley, 1991) for details. By combining it to an optimal offsets assignment, we shall consequently get an optimal static scheduling rule for offset free systems. The deadline driven scheduler and least laxity first schedulers remain of course optimal for offset free systems among the dynamic techniques since both scheduling algorithms are optimal for asynchronous (and arbitrary deadline) systems.

More generally, let $\mathcal{Q}$ be some scheduling rule for asynchronous systems ($\mathcal{Q}$ may be Audsley's optimal static technique, or the classical optimal deadline driven, least laxity first rule, but also non-optimal rule like the rate monotonic or the deadline monotonic ones), we shall propose in this section a $\mathcal{Q}$-optimal method to choose the offsets, in the following sense.

Definition 6. An offset assignment rule (say $\mathcal{A}$) is *$\mathcal{Q}$-optimal* for an offset free task set family if, when a feasible offset assignment exists for a task set of the family using the scheduling rule $\mathcal{Q}$, the offset assignment given by the rule $\mathcal{A}$ is also feasible for that task set with the scheduling rule $\mathcal{Q}$.

The method proposed here is not dedicated to a particular scheduling rule, like in Section 3; we shall present results without considering a scheduling algorithm in particular, not even a specific family of scheduling algorithms. Again, we only suppose that the schedule is finally periodic with a period of P time units, that only the periodic behavior is significant regarding the feasibility of the system, that this periodic behavior only depends on the relative phasings between task requests and that the scheduling policy does not require to examine a granularity $m \geq 2$.

A simple (regarding its principle) $\mathcal{Q}$-optimal offset assignment can be defined by searching a feasible offset assignment among all offset combinations. According to Theorem 6 we may restrict the search to T_i possible values for O_i $(i > 1)$ and a single possibility for O_1 (i.e., $O_1 = 0$), consequently the total number of combinations is $\prod_{i=2}^{n} T_i = \mathcal{O}((\max_{j=2}^{n} T_j)^{n-1})$, and the time complexity of the corresponding offset assignment is $\mathcal{O}((\max_{j=2}^{n} T_j)^{n-1} \times \mathcal{R})$, where $\mathcal{R}$ is the maximal time complexity of a (necessary and sufficient) schedulability test for the considered scheduling rule $\mathcal{Q}$ and the asynchronous systems given by the considered family (e.g., $\mathcal{R} = n^2 \cdot P$ for constrained deadline systems with static priority assignment, combining the Audsley's optimal priority assignment and the response times calculation (explained in (Goossens, 1999), pp.99–105) in a feasibility interval (see (Goossens et al., 1997) for instance). We shall now present a (better) method to only consider the $\frac{\prod_{i=1}^{n} T_i}{P}$ non-equivalent offset assignments as described in Theorem 9.

4.1. Two tasks

We introduce our method by considering first the non-equivalent offset assignments for an offset free system composed of two tasks: τ_1 and τ_2.

According to Theorem 6 and without loss of generality we can choose $O_1 = 0$. Our goal is to find $\frac{T_1 \times T_2}{\text{lcm}\{T_1, T_2\}} = \gcd\{T_1, T_2\}$ non-equivalent choices for O_2. From Theorem 6 we know that we can restrict our search among the values $O_2 = 0, 1, \ldots, T_2 - 1$. We may notice that, with respect to the relative phasings between the requests of τ_1 and those of τ_2, some choices for O_2 can be considered as equivalent according to Definition 2.

EXAMPLE: Consider the following characteristics for tasks τ_1 and τ_2: $\tau_1 = \{T_1 = 4, O_1 = 0\}, \tau_i = \{T_2 = 5\}$ (we do not specify other task characteristics since they do not interfere in the relative phasings between task requests). We see (Figure 6) that choosing $O_2 = 0$ is equivalent to choose $O_2 = 1$, or $O_2 = 2$, etc.

The first occurrence of τ_2 is synchronous with τ_1 if $O_2 = 0$; the second occurrence of τ_2 then has a difference of phase equal to 1 with τ_1 and corresponds to choose

$O_2 = 1$, etc. Regarding task τ_1, choosing $O_2 = 0, O_2 = 1, O_2 = 2$ or $O_2 = 3$ is equivalent; it may be noticed that this is also the case whatever the value of O_1, since these equivalent choices of O_2 generate the same relative phasings between requests of τ_2 and τ_1. □

Two choices (say $O_2 = O_1 + v_1$ and $O_2 = O_1 + v_2$) are equivalent if they define the same relative phasing, in the sense that:

$$\exists k_1, k_2 \in \mathbb{N} : (O_1 + v_1 + k_1 \cdot T_2) \mod T_1 = (O_1 + v_2 + k_2 \cdot T_2) \mod T_1$$

i.e., if

$$\exists k_1, k_2 \in \mathbb{N} : (v_1 + k_1 \cdot T_2) \mod T_1 = (v_2 + k_2 \cdot T_2) \mod T_1. \tag{3}$$

It may be noticed that this relation does not depend on O_1 (we shall see the interest of this property in the more general case for the optimal offset assignment for a set of n tasks). We shall show that Equation (3) is equivalent to

$$v_1 \equiv v_2 \pmod{\gcd\{T_1, T_2\}}. \tag{4}$$

From Equation (3) we get that, for some k_1, k_2, x, k, k':

$$\begin{cases} v_1 + k_1 T_2 = x + kT_1 \\ v_2 + k_2 T_2 = x + k'T_1 \end{cases}$$

Or $v_1 + k_1 T_2 = v_2 + k_2 T_2 + kT_1 - k'T_1$ implying $v_1 \equiv v_2 \pmod{\gcd\{T_1, T_2\}}$. And conversely, since $\gcd\{T_1, T_2\} = \lambda_1 T_1 + \lambda_2 T_2$ for some $\lambda_1, \lambda_2 \in \mathbb{Z}$, equation (4) implies that $v_1 = v_2 + k\lambda_1 T_1 + k\lambda_2 T_2$ for some $k \in \mathbb{Z}$ so that $v_1 + k'T_2 \equiv v_2 + (k' + k\lambda_2)T_2 \pmod{T_1}$ for any $k' \in \mathbb{N}$, and for k' large enough $k' + k\lambda_2$ will be non-negative.

Since $0 \not\equiv 1 \cdots \not\equiv \gcd\{T_i, T_j\} - 1 \pmod{\gcd\{T_i, T_j\}}$, it follows that the values $0, 1, \ldots, \gcd\{T_1, T_2\} - 1$ are non-equivalent choices for O_2.

Our $\mathcal{Q}$-optimal offset assignment checks the feasibility of the asynchronous system given by all these values.

4.2. n tasks

We consider now the offset assignment for a set of n tasks. Our method constructs non-equivalent asynchronous systems first by considering non-equivalent choices for O_2 (O_1 is already fixed, e.g., $O_1 = 0$); for each of these non-equivalent choices for O_2, the method considers next the non-equivalent choices for O_3 regarding the requests of τ_1 and τ_2 (O_1 and O_2 are already fixed), etc.

Suppose that the offsets $O_1, \ldots, O_{i-1}$ are fixed and consider the non-equivalent choices for O_i regarding the relative phasings between the requests of τ_i and those of τ_j ($j < i$). The request pattern limited to the requests of the task sub-set $\{\tau_1, \ldots, \tau_{i-1}\}$ is periodic with a period of $\text{lcm}\{T_1, \ldots, T_{i-1}\}$; from the study of the case $n = 2$ it follows that there are $\gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\}$ non-equivalent choices for O_i, e.g., all the integer values in the half-open interval $[0, \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\})$.

This method constructs $\prod_{i=2}^{n} \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\}$ non-equivalent asynchronous systems; let us now check that all the $\frac{\prod_{i=1}^{n} T_i}{\text{lcm}\{T_i | i=1,\ldots,n\}}$ non-equivalent asynchronous systems are yielded, i.e., that

$$\prod_{i=2}^{n} \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\} = \frac{\prod_{i=1}^{n} T_i}{\text{lcm}\{T_i | i = 1, \ldots, n\}}.$$

THEOREM 10

$$\prod_{i=2}^{n} \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\} = \frac{\prod_{i=1}^{n} T_i}{\text{lcm}\{T_i | i = 1, \ldots, n\}}.$$

Proof: We show the property by induction on n. The property is obvious in the trivial case, $n = 2$, since $\gcd\{T_1, T_2\} = \frac{T_1 \times T_2}{\text{lcm}\{T_1, T_2\}}$. Suppose the property is true up to $n - 1$ and consider the case of n. By induction hypothesis we have that

$$\prod_{i=2}^{n} \gcd\{T_i, \text{lcm}\{T_1, \ldots, T_{i-1}\}\} = \frac{\prod_{i=1}^{n-1} T_i}{\text{lcm}\{T_1, \ldots, T_{n-1}\}} \cdot \gcd\{T_n, \text{lcm}\{T_1, \ldots, T_{n-1}\}\}$$

and by definition of the function lcm and gcd we have that

$$\begin{aligned} \text{lcm}\{T_1, \ldots T_{n-1}, T_n\} &= \text{lcm}\{T_n, \text{lcm}\{T_1, \ldots, T_{n-1}\}\} \\ &= \frac{T_n \times \text{lcm}\{T_1, \ldots, T_{n-1}\}}{\gcd\{T_n, \text{lcm}\{T_1, \ldots, T_{n-1}\}\}}. \end{aligned}$$

The property follows. ■

The computation of non-equivalent offsets can be resolved by applying the Euclid's algorithm to each pair $(T_i, \mathrm{lcm}\{T_1, \ldots, T_{i-1}\})$. The maximal time complexity of this procedure is $\mathcal{O}(n \times \log P)$. Consequently the maximal time complexity of our $\mathcal{Q}$-optimal offset assignment is $\mathcal{O}(n \times \log P) + \mathcal{O}(\frac{\prod_{i=1}^{n} T_i}{P} \times \mathcal{R})$, where $\mathcal{R}$ is the maximal time complexity of the exact schedulability test. Remark that the second term dominates in general the first one.

5. Dissimilar offset assignment

We have studied in section 4 a $\mathcal{Q}$-optimal offset assignment rule which considers "all" offset assignments; more precisely, we have first shown that there are finitely many non-equivalent offset assignments to be considered (i.e., $\prod_{i=2}^{n} T_i$) and then we have presented a method to reduce this number and consider only the $\frac{\prod_{i=1}^{n} T_i}{P}$ non-equivalent offset assignments. The simplification reduces significantly the number of assignments that the optimal algorithm has to consider, but this number remains exponential despite simplifications (for instance when $T_1 = T_2 = \cdots = T_n$, $P = T_1$ is minimal but there are T_1^{n-1} non-equivalent offset assignments). Moreover, this $\mathcal{Q}$-optimal algorithm is based on the feasibility test for the corresponding asynchronous task set, and we know that the maximal time complexity of such a test is in all generality proportional to P, which may be huge. For these reasons, it seems interesting to define a heuristic offset assignment rule which considers a single value for each offset.

We shall present here a rule to choose a single value for O_i among the non-equivalent possibilities. The $\mathcal{Q}$-optimality of the corresponding offset assignment is not preserved of course, but we shall see that this offset assignment schedules a good proportion of systems which are not schedulable in the synchronous case while the time complexity of the offset assignment is polynomial in terms of the number of tasks and the maximal period of the system.

Remark that, before applying a sophisticated offset assignment in order to schedule an offset free system, there are some preliminary points to consider.

First, for the various classes of periodic task sets (regarding their deadlines) considered in this work, we can first check if $U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$ (which is sufficient for implicit deadline systems using the deadline driven scheduler) and then check if the system is schedulable in the synchronous case. The time complexity of feasibility tests for synchronous systems are in all generality simpler than for asynchronous systems. For a static scheduler in particular, except for arbitrary deadline systems, this can be checked by a pseudo-polynomial algorithm since in this case $[0, \max\{D_i | i = 1, \ldots, n\})$ is a feasibility interval (see (Leung et al., 1982) for more details). If the system is not schedulable in the synchronous case, it may exist a judicious choice of the offsets which schedules the system if $U \leq 1$.

We now come back to the problem of offset assignment and the presentation of our rule. The main principle of our heuristic offset assignment rule is to choose the offsets in order to move away from the worst case, i.e., from the synchronous case, as much as possible. This problem is not obvious and we have investigated many solutions. Our heuristic is closely based on the manner to estimate if a given offset assignment is close (or not) to the synchronous case.

We estimate the proximity of an offset assignment with the synchronous case by considering *minimal length such that, for any (different) tasks τ and τ', there is an interval of that length containing a request of both τ and τ' (in the periodic part of the schedule)*; we shall see later that this definition can be refined, we do not give details here. Note that this length is 0 for synchronous systems (or for asynchronous systems which are equivalent to the synchronous case). The more this length is large, the more the requests are dissimilar (from the synchronous case) in the whole schedule.

We shall present here our offset assignment rule, which maximizes this length. For convenience, let us call our rule the *dissimilar offset assignment*.

We introduce our rule by considering first only the requests of τ_i and τ_j. We have seen in section 4 that if we consider only the requests of τ_j and those of τ_i, there are $\gcd\{T_i, T_j\}$ non-equivalent choices for O_i (whatever O_j): $0, 1, \ldots, \gcd\{T_i, T_j\} - 1$. Here we are concerned by the time which separates the requests of τ_i and those of τ_j.

THEOREM 11 *Let* $r \in [0, \gcd\{T_i, T_j\})$. *If* $O_i = O_j + r$ *(or* $O_j = O_i + r$*), the minimum distance between a request of* τ_i *and a request of* τ_j *is* $\min\{r, \gcd\{T_i, T_j\} - r\}$.

Proof: Without loss of generality we can assume that $O_i = O_j + r$. First remark that the offset assignment $O_i = O_j + r$ induces a distance r between a request of τ_i and a request of τ_j; consequently the minimum distance between a request of τ_i and a request of τ_j cannot be larger than $\min\{r, \gcd\{T_i, T_j\} - r\}$. We now show the property by contradiction: suppose that the property is false, i.e., the minimum distance between a request of τ_i and a request of τ_j is b with $0 \leq b < \min\{r, \gcd\{T_i, T_j\} - r\}$. In this case we have to distinguish between two cases:

1. $\exists k_1, k_2 \in \mathbb{N} : \; 0 \leq O_j + r + k_1 T_i - (O_j + k_2 T_j) = b$,

2. $\exists k_1, k_2 \in \mathbb{N} : \; 0 \leq O_j + k_2 T_j - (O_j + r + k_1 T_i) = b$.

The first relation implies that $b \equiv r \pmod{\gcd\{T_i, T_j\}}$, which leads to a contradiction since $0 \leq b < r < \gcd\{T_i, T_j\}$.

The second relation implies that $b \equiv -r \pmod{\gcd\{T_i, T_j\}}$, which leads to a contradiction since $0 \leq b < \gcd\{T_i, T_j\} - r$ or $b + r < \gcd\{T_i, T_j\}$ and $b + r \not\equiv 0 \pmod{\gcd\{T_i, T_j\}}$. ■

It follows from Theorem 11 that the minimum distance between a request of τ_i and a request of τ_j is $\left\lfloor \frac{\gcd\{T_i, T_j\}}{2} \right\rfloor$ and corresponds to the offset assignment $O_i = O_j + \left\lfloor \frac{\gcd\{T_i, T_j\}}{2} \right\rfloor$ (or $O_j = O_i + \left\lfloor \frac{\gcd\{T_i, T_j\}}{2} \right\rfloor$).

Suppose that $\delta = \gcd\{T_i, T_j\} = \max\{\gcd\{T_k, T_r\} \mid k \neq r\}$ and consider the offset assignment $O_i = 0$ and $O_j = \lfloor \frac{\delta}{2} \rfloor$. This offset assignment maximizes the minimal distance between two requests of different tasks and consequently maximizes the minimal length such that, for any (different) tasks τ and τ', there is an interval of that length containing a request of both τ and τ'. Although our criterion is satisfied, we shall apply the same principle for the remaining free offsets, in order to move away from the synchronization of the remaining task requests. This leads to Algorithm 1.

Algorithm 1 fixes the n offsets (see the while loop, line 7) of the periodic task set by considering the values $\gcd\{T_i, T_j\}$ by decreasing order. Suppose that p, q are

Algorithm 1 The dissimilar offset assignment

```
1: $G \Leftarrow \{(i, j, \gcd(T_i, T_j)) \mid 1 \le i < j \le n\}$;
2: $\mathcal{G} \Leftarrow ((i_1, j_1, \gcd(T_{i_1}, T_{j_1})), (i_2, j_2, \gcd(T_{i_2}, T_{j_2})), \dots)$,
   with $\{(i_k, j_k, \gcd(T_{i_k}, T_{j-k}) \mid k = 1, \dots, \frac{n(n-1)}{2}\} = G$, such that $r < p \Longrightarrow$
   $\gcd(T_{i_r}, T_{j_r}) \ge \gcd(T_{i_p}, T_{j_p})$;
3: {The vector $\mathcal{G}$ is a sorted version of the set $G$. In the following we shall use the
   "dot notation" to denote the 3 fields of each entry of $\mathcal{G}$, which are *row*, *col* and
   *gcd*, respectively.}
4: $assignment \Leftarrow n$; {The remaining number of offset assignments.}
5: $Mark \leftarrow$ (false, ... , false); {$n$ components.}
6: $k \Leftarrow 1$;
7: while $assignment > 0$ do
8:   if $\neg(Mark_{\mathcal{G}_k.col}) \wedge \neg(Mark_{\mathcal{G}_k.row})$ then
9:     $O_{\mathcal{G}_k.row} \Leftarrow$ rand(); $O_{\mathcal{G}_k.col} \Leftarrow O_{\mathcal{G}_k.row} + \mathcal{G}_k.gcd$ div 2;
10:    $assignment \Leftarrow assignment - 2$; $Mark_{\mathcal{G}_k.row} =$ true; $Mark_{\mathcal{G}_k.col} =$ true;
11:  else if $\neg(Mark_{\mathcal{G}_k.col})$ then
12:    $O_{\mathcal{G}_k.col} \Leftarrow O_{\mathcal{G}_k.row} + \mathcal{G}_k.gcd$ div 2;
13:    $assignment \Leftarrow assignment - 1$; $Mark_{\mathcal{G}_k.col} =$ true;
14:  else if $\neg(Mark_{\mathcal{G}_k.row})$ then
15:    $O_{\mathcal{G}_k.row} \Leftarrow O_{\mathcal{G}_k.col} + \mathcal{G}_k.gcd$ div 2;
16:    $assignment \Leftarrow assignment - 1$; $Mark_{\mathcal{G}_k.row} =$ true;
17:  end if
18:  $k \Leftarrow k + 1$;
19: end while
```

such that $\delta' = \gcd\{T_p, T_r\}$; we have to distinguish between three cases: O_p and O_r are not already fixed, p (or r) is already fixed or p and q are already fixed. In the first case the rule fixes two more offsets: $O_p = 0$ and $O_r = O_p + \left\lfloor \frac{\delta'}{2} \right\rfloor$ (see lines 9–10). The choice $O_p = 0$ is not optimal: what is significant is $|O_p - O_r|$. But it is not obvious to choose the optimal value, for this reason we choose O_p randomly in order to avoid a synchronization with already fixed offsets (if any). In the second case the rule fixes a single offset (e.g., $O_r = O_p + \left\lfloor \frac{\delta'}{2} \right\rfloor$, see lines 12–13, 15–16). In the third case the algorithm does not fix any new offset and considers the next value $\gcd\{T_i, T_j\}$ (i.e., continues with the next iteration, see line 18).

Remark that this offset assignment rule does not rely on the (priority) scheduling algorithm nor on the kind of deadline (implicit/constrained/arbitrary case) nor on the computation times (C_i's) nor on the deadlines (D_i's). The rule only depends on the periods of the system and tries as much as possible to move away from the synchronous case.

We consider now the (maximal) complexity of our offset assignment rule.

THEOREM 12 *The maximal time complexity of the dissimilar offset assignment rule is $\mathcal{O}(n^2 \cdot (\log T^{max} + \log n^2))$ and the maximal space complexity is $\mathcal{O}(n^2)$.*

Proof: The dissimilar offset assignment rule computes (see line 1) first the value $\gcd\{T_i, T_j\}$ for each pair (T_i, T_j), this can be resolved by applying the Euclid's algorithm to each pair $(T_i, T_j), j \neq i$, hence the time complexity of this procedure is $\mathcal{O}(n^2 \times \log T^{max})$. Then the algorithm sorts the set G, the time maximal complexity of this procedure is $\mathcal{O}(n^2 \log n^2)$. Finally, the algorithm fixes the offsets with a maximal time complexity $\mathcal{O}(n^2)$. The dissimilar offset assignment stores $\mathcal{O}(n^2)$ integers (the vector $\mathcal{G}$). Hence, the maximal time complexity of the dissimilar offset assignment rule is $\mathcal{O}(n^2(\log T^{max} + \log n^2))$ and the maximal space complexity is $\mathcal{O}(n^2)$. ■

5.1. Experimental results

We shall present now the evaluation of our heuristic rule. Since monotonic priority assignments are not optimal for offset free systems (Goossens et al., 1997) and since the Audsley's optimal priority assignment is somewhat heavy; we shall rather study the effectiveness of our rule for the dynamic deadline driven scheduler which remains optimal for offset free systems, and we consider constrained deadline offset free systems. We present here simulation results of our heuristic applied to a large number of task sets chosen randomly. For each task set we first check if the system is schedulable in the synchronous situation if this is not the case, we check if the system is schedulable in an asynchronous situation, i.e., we consider at worst the $\frac{\prod_{i=1}^{n} T_i}{P}$ situations. If a feasible offset assignment exists we check if the dissimilar offset assignment is also feasible for the same task set.

We have oriented our task set random generation in order to have "critical" systems, where the utilization factor is large, and the interest of choosing offsets and then our heuristic rule is relevant. It may be noticed that checking if a system is schedulable for some offsets leads to extremely long computations since we have to consider $\frac{\prod_{i=1}^{n} T_i}{P}$ exact feasibility tests. For this reason we have strongly limited n and the T_i's in our simulations: the number of tasks n is chosen equiprobably in $[5, 13]$, the T_i's in $[5, 30]$, the D_i's in $[\frac{T_i}{2}, T_i]$ and then the computation times (C_j) are chosen randomly in the interval $[1, D_j]$ and we only consider systems where the utilization factor is in $[0.65, 1)$.

Figure 7 shows the proportion of systems schedulable in the synchronous case, schedulable only in a asynchronous situation (non-equivalent to the synchronous one) and unschedulable whatever the offsets, in function of U. From Figure 7 it can be noticed that the interest of offset free systems again occurs in an obvious way, since the proportion of systems schedulable in the synchronous situation decreases when U increases, while the proportion of systems schedulable only in an asynchronous situation increases steadily (especially when $U > 0.9$).

We have also considered these characteristics in function of the number n of tasks but it occurs that the phenomenon does not depend significantly on n (in our limited simulations).

We have also computed for each task set schedulable in a particular asynchronous situation (non-equivalent to the synchronous situation) the proportion of task sets which remain schedulable with our dissimilar offset assignment and with randomly chosen offsets: these proportions are 82% and 60% respectively. This second experiment shows that:

- It is very pessimistic to consider the feasibility of systems only in the synchronous case. Choosing the offsets randomly already increases considerably the number of feasible systems.

- Our heuristic offset assignment rule, like the random one, increases considerably the number of feasible systems in comparison with the pessimistic synchronous case. Moreover, our rule increases nicely the number of feasible systems in comparison with the random rule and may be considered as 82% optimal. That leads to think that our criterion is a good measure of the proximity with the synchronous situation and exhibits the efficiency of our rule.

We have also considered the phenomenon in function of U, and with our (limited) simulations it occurs that the phenomenon does not depend significantly on n nor on U; "larger" simulations (e.g., larger number of tasks, larger T_i's, ...) could be interesting for a finer study of the phenomenon but remains for further researches.

6. Conclusion

In this paper we have studied the scheduling problem of offset free systems. We have shown that we can restrict the problem by considering $\frac{\prod_{i=1}^{n} T_i}{P}$ non-equivalent offset assignments and we have proposed a method to construct these values. We have proposed an optimal offset assignment which considers only these non-equivalent offset assignments; however the number of combinations remains in general exponential. For this reason, we have defined a rule to choose a single offset for each task in order to try to move away from the worst case as most as possible. This algorithm is nearly optimal and has a reasonable time complexity in terms of the task characteristics.

Interesting questions for further research related to offset free systems include: the study of optimal (or pseudo-optimal, i.e., heuristic) static priority assignments for offset free systems; better statistical analysis of the actual benefit of our dissimilar offset assignment, investigate other heuristic rules (e.g., based also on the C_i's and the D_i's), ...

Acknowledgments

The author gratefully acknowledges the significant contributions of R. Devillers. The author also wishes to thank J. Doyen and O. Markowitch for their help. Finally the detailed comments of two anonymous referees greatly helped in improving the presentation of the paper.

Notes

1. In this context, by "not altered" we mean: from time t_0 the original schedule is identical to those of the modified system from time $t_0 + a$ (for some $t_0, a \in \mathbb{N}$).

References

N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. Technical report, The University of York, UK, 1991.

S. K. Baruah, R. R. Howell, and L. E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoret. Comput. Sci.*, 1(118), 93.

S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 2:301–324, 1990.

J. Goossens. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints.* PhD thesis, Université Libre de Bruxelles, Belgium, 1999.

J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. *Real-Time Systems*, 13(2):107–126, September 1997.

D. E. Knuth. *The Art of Computer Programming*, volume 2 of *Seminumerical Algorithms.* Addison-Wesley, 1969.

J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the Real-Time Systems Symposium - 1990*, pages 201–213, Lake Buena Vista, Florida, USA, December 1990.

J. Y. -T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3):115–118, November 1980.

J. Y. -T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

A. Mok and M. Dertouzos. Multiprocessor scheduling in a hard real-time environment. In *Proceedings of the Seventh Texas Conference on Computing Systems*, 1978.

J. A. Stankovic and K. Ramamritham. What is predictability for real-time systems? *Real-Time Systems*, 2:247–254, 1990.

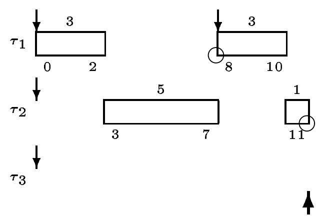

Figure 1. The task set is unschedulable in the synchronous case.

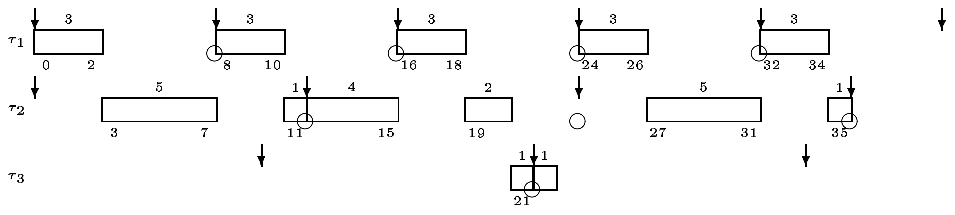

Figure 2. The task set is schedulable; at $t = 34$ the situation is the same as at $t = 10$ and the schedule repeats.

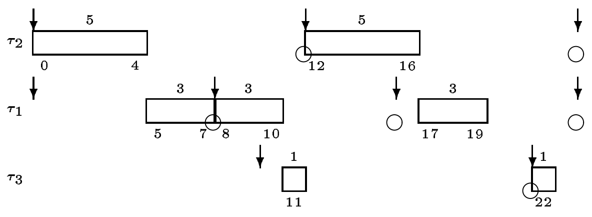

Figure 3. The task set is schedulable; at $t = 24$ the situation is the same as at $t = 0$ and the schedule repeats.

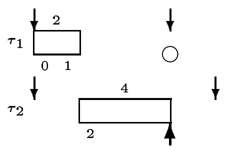

Figure 4. The synchronous task set is unschedulable.

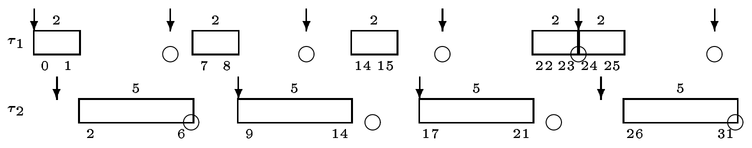

Figure 5. The asynchronous task set is schedulable; from time $t = 24$, the schedule repeats.

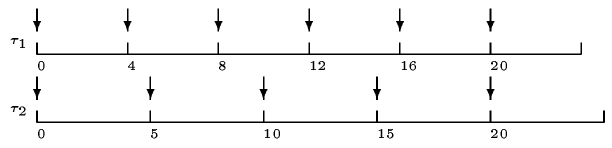

Figure 6. Choosing $O_2 = 0$ is equivalent to choose $O_2 = 1$,...

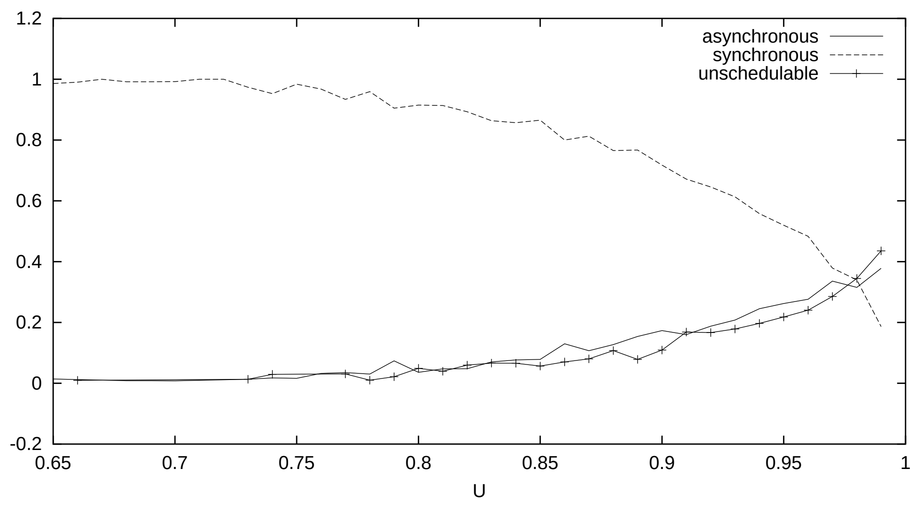

Figure 7. The proportion of systems schedulable in the synchronous case, schedulable only in an asynchronous situation and unschedulable whatever the offsets, in function of U.