Improving the Quality of Control of Periodic Tasks Scheduled by FP with an Asynchronous Approach

P. Meumeu Yomsi, L. George, Y. Sorel, D. de Rauglaudre
AOSTE Project-team
INRIA Paris-Rocquencourt
Le Chesnay, France
{*patrick.meumeu, laurent.george, yves.sorel, daniel.de_rauglaudre*}*@inria.fr*

Abstract

The aim of this paper is to address the problem of correctly dimensioning real-time embedded systems scheduled with Fixed Priority (FP) scheduling. It is well known that computers which control systems are greatly affected by delays and jitter occurring in the control loop. In the literature, a deadline reduction approach has been considered as one solution to reducing the jitter affecting a task, thereby obtaining better loop stability in the control loop. Here, in order to improve the sensitivity of the deadlines, we propose another solution for reducing the worst case response time of the tasks, hence reducing the jitter, when all the tasks are scheduled with the Deadline Monotonic Algorithm. This is performed for a specific asynchronous scenario for harmonic periodic tasks. We compare the results to those for the synchronous scenario in terms of minimum deadline reduction factor preserving the schedulability of tasks set in both cases.

Keywords: Real-time systems, Fixed-priority scheduling algorithms, Sensitivity analysis, Robust control.

1. Introduction

In this paper we consider the problem of correctly dimensioning real-time embedded systems ([1], [2], [3], [4]). The correct dimensioning of a real-time system strongly depends on the determination of the tasks' Worst-Case Execution Times (WCETs). Based on the WCETs, Feasibility Conditions (FCs) ([5], [6], [7]) can be established to ensure that the timeliness constraints of all the tasks are always met when tasks are scheduled by a fixed or a dynamic priority driven scheduling algorithm. We consider an application composed of a periodic task set $\Gamma_n = \{\tau_1, \cdots, \tau_n\}$ of n periodic tasks, scheduled with Fixed Priority (FP) preemptive scheduling. The classical definition of a periodic task τ_i, is:

- C_i: the Worst Case Execution Time (WCET) of τ_i.
- T_i: the period of τ_i.
- D_i: the relative deadline of τ_i (a task requested at time t must be terminated by its absolute deadline $t + D_i$), where $D_i \leq T_i$.

A recent research area called sensitivity analysis aims at providing interesting information on the validity of feasibility conditions by considering possible deviations of task WCETs ([2]), task periods ([2]), or task deadlines ([3]). This makes it possible, for example, to find a feasible task set, if the current one is not feasible, by modifying the task parameters or determining the impact of a change in architecture on the feasibility of a task set. A task set is declared feasible if for any task in the synchronous scenario, its worst case response time is less than or equal to its deadline. We are interested in the sensitivity of deadlines. Computer controlling systems are very much affected by delays and jitter occurring in the control loop. A deadline reduction has been considered by ([8]) as one solution to reducing the jitter affecting a task and therefore obtaining better loop stability in the control loop. The jitter of a task depends on the minimum and on the worst case response times. Reducing the deadline of a task can be a way to reduce the worst case response time of a task and thus can reduce the jitter of the task. However, this deadline reduction should be performed in such a way that it does not cause any task to fail at run-time. This supposes a scheduling driven by deadlines.

This paper proposes a solution to reduce as much as possible the worst case response time of each task when tasks are scheduled with fixed priorities, according to Deadline Monotonic Algorithm, by using a specific asynchronous first release times scenario. We show the benefits of our asynchronous scenario by comparing the minimum deadline reduction factor applied preserving the schedulability of the tasks in the synchronous and in the our asynchronous scenario.

With Deadline Monotonic Algorithm, tasks are scheduled according to their relative deadlines. The smaller the relative deadline, the higher the priority. Starting from a schedulable task set, we want to characterize the minimum deadline reduction factor $0 < \alpha \leq 1$ such that any task $\tau_i, i = 1, \ldots, n$ having a deadline $D_i = \alpha \times T_i$ is schedulable. α is such that any smaller reduction factor would lead to a non schedulable task set. We compare the value of α obtained in

the worst case synchronous scenario (all the tasks are first released at the same time) to that obtained with a particular asynchronous scenario that we propose, and which has some interesting properties. We show that the minimum reduction factor obtained in our asynchronous scenario is always less than or equal to the minimum reduction factor obtained in the synchronous scenario.

Reducing the deadline of a task makes it possible to reduce the jitter resulting from the execution of a task. In this paper we show that the maximum deadline reduction is minimized for the synchronous scenario where all the tasks are first released at the same time.

We then propose a particular asynchronous first release times scenario that allows us to obtain better feasibility conditions and a better deadline reduction factor than the one obtained with the synchronous scenario, thus reducing the jitter of the tasks for a better control.

The feasibility problem of asynchronous task sets is known to be more complex than for synchronous task sets. We introduce a new formalism to compute the worst case response time of a task for asynchronous task sets. We apply this approach to the case where the periods of the tasks are harmonic. We then show that in this case, the worst case response time is always obtained for the second instance of a task, which represents a significant reduction in complexity.

The rest of the paper is organized as follows. In section 2, we give a state of the art regarding sensitivity analysis of deadlines considering dynamic and fixed priority driven schedulings. We then focus on asynchronous task sets and recall existing feasibility conditions. In section 3, we introduce the concepts and notations and establish important properties for the particular asynchronous scenario that we have chosen. We consider harmonic periods. We show that, using this particular scenario, the worst case response time of every task is obtained for its second instance[1]. In section 4, we introduce a new scheduling representation which is more compact than the classical linear representation / Gantt Chart for a schedule. In section 5, we introduce the concept of Mesoid which is used to compute the worst case response time of an asynchronous task set. In section 6, we give an algorithm for the computation of the worst case response time of any task in our asynchronous scenario, then we show how to compute the minimum deadline reduction factor. An example is given in order to compare the deadline reduction factor obtained with our asynchronous scenario to that in the synchronous scenario. We provide experimental results in section 7 based on extensive simulations comparing the deadline reduction factor for several load configurations in both the synchronous case and in our asynchronous scenario. Finally, we conclude in section 8.

1. Throughout the paper all subscripts refer to tasks whereas all superscripts refer to instances.

2. State of the art

Sensitivity analysis for deadlines has been considered for Earliest Deadline First (EDF) scheduling algorithm by ([3]) showing how to compute the minimum feasible deadlines such that the deadline of any task τ_i equals αD_i, where α is reduction factor $0 < \alpha \leq 1$. In ([8]), the space of feasible deadlines (D-space), a space of n dimensions has been considered. Any task set having deadlines in the D-space is considered to be schedulable. To the knowledge of the authors, no work has been done on the sensitivity of deadlines for fixed priority scheduling algorithms.

Few results have been proposed to deal with the deadline assignment problem. As far as the authors are aware, no results are available for Fixed Priority (FP) scheduling. Baruah $\&$ al., in [9] propose modifying the deadlines of a task set to minimize the output, seen as a secondary criteria. In Cervin $\&$ al. ([10]), the deadlines are modified to guarantee close-loop stability of a real-time control system. Marinca $\&$ al. ([11]) focus on the deadline assignment problem in the distributed case for multimedia flows. The deadline assignment problem is formalized in terms of a linear programming problem. The scheduling considered on every node is non-preemptive EDF or FIFO, with a jitter cancellation applied on every node. A performance evaluation of several deadline assignment schemes is proposed.

A recent paper proposed by Balvastre $\&$ al. ([3]) proposes an optimal deadline assignment for periodic tasks scheduled with preemptive EDF in the case of deadlines less than or equal to periods. The goal is to find the minimum deadline reduction factor preserving all the deadlines of the tasks.

They first focus on the case of a single task deadline reduction and show how to compute D_i^{min}, the minimum deadline of task τ_i such that any deadline smaller than D_i^{min} for task τ_i will lead to a non-feasible task set.

They also show in [3] that when considering the reduction of a single task τ_i, D_i^{min} is the worst case response time of task τ_i for EDF scheduling. The maximum deadline reduction factor α_i for task τ_i is then: $\alpha_i = 1 - \frac{D_i^{min}}{D_i}$.

In the case of a deadline reduction applied to n tasks, the goal is to minimize all tasks' deadlines assuming the same reduction factor for all the tasks (with no preference regarding which task requires the greatest deadline reduction). Balbastre $\&$ al. in [3] show how to compute the maximum deadline reduction factor α applied to all the deadlines using an iterative algorithm. The principle is to compute the minimum slack $t - h(t)$ for any time $t \in [0, L)$ to determine the deadline reduction factor applied to all the tasks, where $h(t) = \sum_{i=1}^{n} max(0, 1 + \lfloor \frac{t-D_i}{T_i} \rfloor) C_i$ and L is the length of the first synchronous busy period, solution of the equation $t = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i$.

$\tau = \{\tau_1, \ldots, \tau_n\}$: **task set**;
L $\leftarrow$ compute-L(τ) : **integer**; $\alpha \leftarrow 1$: **real**
$slack = min_{t \in [0,L)}(t - h(t))$: **real**;
While $(slack \neq 0)$ **do**
  $\alpha = min_{i=1\ldots n}(1 - \frac{slack}{D_i})$;
  For $(i = 1; i < n; i++)$ **do**
    $D_i = \alpha D_i$;
  end For
  $slack = min_{t \in [0,L)}(t - h(t))$;
done
Return α;

Algorithm 1: Computation of α for EDF scheduling

For Fixed Priority (FP) scheduling, necessary and sufficient FCs have been proposed in the case of non-concrete tasks where the first release times of the tasks can be arbitrary. A classical approach is based on the computation of the tasks' worst-case response times ([12], [6]). The worst-case response time, defined as the worst case time between the request time of a task and its latest completion time, is obtained in the worst case synchronous scenario where all the tasks are first released at the same time, and is computed by successive iterations. This worst case response time provides a bound on the response time valid for any other task first release times. It can be shown that considering only non-concrete tasks can lead to a pessimistic dimensioning [13].

The complexity of this approach depends on the worst case response time computation complexity. In the case of deadlines less than or equal to periods for all tasks, the worst-case response time R_i of a task τ_i is obtained in the synchronous scenario for the first release of τ_i at time 0 and is the solution of the equation ([12]) $R_i = W_i(R_i)$, where $W_i(t) = C_i + \sum_{\tau_j \in hp(i)} \left\lceil \frac{t}{T_j} \right\rceil C_i$ and $hp(i)$ denotes the set of tasks with a priority higher than or equal to that of τ_i except τ_i itself. The value of R_i is computed by successive iterations and the number of iterations is bounded by $1 + \sum_{\tau_j \in hp(i)} \left\lfloor \frac{D_i}{T_j} \right\rfloor$. A necessary and sufficient feasibility condition for a task set is: $\exists t \in S$, such that $W_i(t)/t \leq 1$, where $S = \cup_{\tau_j \in hp(i)} \{kT_j, k \in N\} \cap [0, D_i]$. For any task τ_i, the checking instants correspond to the arrival times of the tasks with a higher priority than τ_i within the interval $[0, D_i]$. This feasibility has been improved by ([14]), where the authors show how to reduce the time instants of S. For any task τ_i, they show how to significantly reduce the number of checking instants during the interval $[0, D_i]$ to at most 2^{i-1} times rather than $1 + \sum_{\tau_j \in hp(i)} \left\lfloor \frac{D_i}{T_j} \right\rfloor$. When deadlines and periods are independent, ([6]) shows that the worst-case response times of a sporadic task τ_i are not necessarily obtained for the first activation request of τ_i at time 0. The number of activations to consider is $1 + \left\lfloor \frac{L_i}{T_i} \right\rfloor$, where L_i is the length of the worst-case level-τ_i busy period defined in ([15]) as the longest period of processor activity running tasks of priority higher than or equal to τ_i in the synchronous scenario. It can be shown that $L_i = \sum_{\tau_j \in hp(i) \cup \tau_i} \left\lceil \frac{L_i}{T_j} \right\rceil C_j$. From its definition, L_i is bounded by:

$$Min \left\{ \sum_{\tau_j \in hp(i) \cup \tau_i} \frac{C_j}{1 - \sum_{\tau_j \in hp(i) \cup \tau_i} \frac{C_j}{T_j}}, \sum_{\tau_j \in hp(i) \cup \tau_i} \frac{C_j}{T_j} \cdot P \right\} \text{([7])}.$$

where $P = LCM(T_1, \ldots, T_n)$ is the least common multiple (*LCM*) of the periods of all tasks and it leads to a pseudo-polynomial time complexity for the feasibility conditions.

This is an interesting approach as it provides a pseudo-polynomial time complexity but it may lead to a pessimistic dimensioning as the synchronous scenario might not be likely to occur.

In order to improve the schedulability of the systems, offset strategies on the first release times of the tasks have been considered. A system where offsets are imposed is called an asynchronous system. ([13]) shows significant feasibility improvements considering offsets. Simulations show that the number of feasible schedulable systems with offsets (while unfeasible in the synchronous case) increases with the number of tasks for a processor load of 0.8 and ranges from 40.5% to 97% for different offset assignment strategies. This percentage strongly decreases when the load is high (tends to 1).

With asynchronous tasks, ([16]) shows that for a given offset assignment, the schedulability of the tasks must be checked in the interval $[0, max_{i=1\ldots n}(O_i) + 2P)$ where P is the least common multiple of the tasks and O_i is the offset of task τ_i, leading to an exponential time complexity. To provide less pessimistic FCs, it is furthermore mandatory to prove that the offsets will not result later in a synchronous scenario. This problem is referred to as the K-simultaneous congruence problem in the state of the art ([16]). This feasibility result has been significantly improved by ([17]) showing that the interval to check the feasibility of a periodic task set with offsets can be reduced to $[0, max_{i=1\ldots n}(O_i) + P)$.

Furthermore, ([16]) proves the non optimality of Deadline Monotonic scheduling algorithm for asynchronous systems when task deadlines are less than or equal to periods. An optimal priority assignment can be obtained in $O(n^2)$ using the Audsley procedure ([18]).

A particular case denoted *offset free systems* corresponds to the case where offsets can be chosen arbitrarily. An optimal offset assignment is given in ([19]). An offset assignment is optimal if it can find a schedulable offset whenever a feasible assignment exists. The complexity of

the offset assignment algorithm is exponential and is in $O((max_{2\leq j\leq n}T_j)^{n-1})$. The offset of task τ_1 is set to 0. Different offset strategies / heuristics have been considered in the literature. Among them, we can cite the dissimilar offset assignment proposed by ([19]) that consists in shifting (computing a distance between the offsets) the offset of the tasks to be as far as possible from the synchronous scenario. The algorithm sorts the couple of tasks (τ_i, τ_j) by decreasing values of $gcd\{T_i, T_j\}$ such that the distance belongs to $[0, gcd\{T_i, T_j\})$. The dissimilar offset assignment significantly reduces the number of offsets to consider, leading to a complexity in $O(n^2.(log(max_{i\in[1,n]}T_i)+log(n^2)))$. Other offset assignment strategies have been considered by ([13]) using the Audsley procedure to determine the subset of tasks of τ that can be feasibly scheduled in the synchronous scenario (setting their offset to 0). The offsets are only computed for the subset of tasks that are unfeasible with the Audsley procedure in the synchronous case. The authors consider different criteria to assign the offsets, based on the criteria used to sort the couple of tasks (τ_i, τ_j). The complexity is the same as that of the dissimilar offset assignment.

In this paper we consider a particular asynchronous harmonic concrete task set where $\forall 2 \leq i \leq n, T_{i-1} \mid T_i$ (i.e. there exists $k \in \mathbb{Z}$ such that $T_i = kT_{i-1}$) with particular offsets. In the case of non-concrete harmonic tasks, when tasks are scheduled with Rate Monotonic Algorithm (the shorter the period, the higher the priority) and in the case where deadlines are equal to periods, a necessary and sufficient condition for the feasibility of such a system is given by $U = \sum_{i=1...n} \frac{C_i}{T_i} \leq 1$ (see [20]). This potentially proves the benefits of considering harmonic tasks in order to get better feasibility conditions. This property does not hold when deadlines can be shorter than periods. In this case we show how to determine in $\mathcal{O}(n)$ the offset of the tasks to obtain a pseudo-polynomial time feasibility condition instead of an exponential one. In the case of asynchronous tasks, the worst case response time cannot be computed with a recursive equation as for the synchronous tasks. This is due to the fact that with offsets, there is not necessarily a continuous busy period from time 0 to the release time of a task. In this paper we investigate a new approach to compute the worst case response time of a task based on the Mesoid approach. This approach was first introduced by ([4]) in the context of real-time scheduling with preemption cost. This approach does not require a continuous busy period to compute the worst case response times of the tasks. We propose a particular offset assignment, such that the worst case response time of any task is obtained for its second request time, providing an exponential time improvement in the complexity of the FCs.

More recently, for control systems, [21] has proposed to include the control delay resulting from the response time of a task as a cost function for the controllers. They show how to solve the optimal period assignment problem analytically.

3. Properties of the asynchronous harmonic task set

3.1. Concepts and notations

We recall classical results in the uniprocessor context for real-time scheduling.

- Time is assumed to be discrete (task arrivals occur and task executions begin and terminate at clock ticks; the parameters used are expressed as multiples of the clock tick); in [22], it is shown that there is no loss of generality with respect to feasibility results by restricting the schedules to be discrete, once the task parameters are assumed to be integers (multiples of the clock tick) i.e. a discrete schedule exists, if and only if a continuous schedule exists.
- A task set is said to be valid with a given scheduling policy if and only if no task occurrence ever misses its absolute deadline with this scheduling policy.
- $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ is commonly called the processor utilization factor associated to the task set Γ_n, i.e., the fraction of processor time spent in the execution of the task set ([23]). If $U > 1$, then no scheduling algorithm can meet the tasks' deadlines.
- The synchronous scenario corresponds to the scenario where all the tasks are released at the same time (at time 0).

The model depicted in figure 1 is Liu & Layland's pioneering model [23] for systems executed on a single processor.

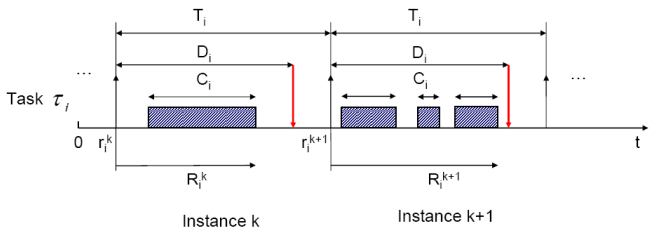

Figure 1. Model

Throughout the paper, we assume that all timing characteristics are non-negative integers, i.e. they are multiples of some elementary time interval (for example the "CPU tick", the smallest indivisible CPU time unit):

We introduce several notations for a periodic task $\tau_i = (C_i, D_i, T_i)$ used to compute the worst case response time of a task:

- τ_i^k: The k^{th} instance of τ_i

- r_i^1: Release time of the first instance of τ_i
- $r_i^k = r_i^1 + (k-1)T_i$: Release time of τ_i^k
- R_i^k: Response time of τ_i^k released at time r_i^k
- R_i: Worst-case response time of τ_i

3.2. The specific asynchronous scenario

Here we give some interesting properties which are satisfied by the specific asynchronous scenario we propose and which lead to the conclusion that the worst case response time of a task in our asynchronous scenario is obtained for any task for its second release.

In this section we assume that the relative deadline for each task equals its period, i.e. $D_i = T_i$. This assumption will be weakened in section 6.

We first show in lemma 1 that with harmonic asynchronous tasks, two instances belonging to any two tasks can never be released at the same time if their release times are not equal modulo their periods.

Lemma 1: Let $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a system of n independent harmonic (i.e. $T_i \mid T_{i+1}, \forall i \in \{1, \cdots, n-1\}$) preemptive tasks ordered by decreasing priorities ($T_i \leq T_{i+1}, \forall i \in \{1, \cdots, n-1\}$).

If there exist two tasks $\tau_i, \tau_j \in \Gamma_n$, $(i < j)$ such that $r_j^1 \neq r_i^1 \bmod [T_i]$ [2], then $\nexists k, l \geq 0$ such that $r_j^k = r_i^l$.

Proof: (by contradiction)

Let us assume that there exist two tasks $\tau_i, \tau_j \in \Gamma_n$, $(i < j)$ such that $r_j^1 \neq r_i^1 \bmod [T_i]$, and $\exists k, l \geq 0$ such that $r_j^k = r_i^l$.

$$\begin{aligned} r_j^k = r_i^l \quad &\Leftarrow \quad r_j^1 + (k-1)T_j = r_i^1 + (l-1)T_i \\ &\Leftarrow \quad r_j^1 = r_i^1 + (l-1)T_i - (k-1)T_j \\ &\Leftarrow \quad r_j^1 = r_i^1 \bmod [T_i] \text{ as } T_i \mid T_j. \end{aligned}$$

Contradicts the hypothesis and thus, ends the proof. □

We now show in theorem 1 that from the point of view of any task in the system, the schedule repeats identically from the second instance.

Theorem 1: (inspired by theorem 2.48 in [24])

Let $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ be a system of n asynchronous independent periodic preemptive tasks ordered by decreasing priorities ($T_i \leq T_{i+1}, \forall i \in \{1, \cdots, n-1\}$). Let $r_1^1, r_2^1, \cdots, r_n^1$ be respectively the release time of their first instances. Let $(s_i)_{1 \leq i \leq n}$ be the sequence inductively defined by

$$\begin{cases} s_1 = r_1^1 \\ s_i = r_i^1 + \left\lceil \dfrac{(s_{i-1} - r_i^1)^+}{T_i} \right\rceil \cdot T_i \quad \forall i \in \{2, \cdots, n\} \end{cases} \tag{1}$$

Then,
if Γ_n is schedulable up to $s_n + H_n$, with $H_n = LCM(T_1, T_2, \cdots, T_n)$ and $x^+ = max\{x, 0\}$, then Γ_n is schedulable and periodic from s_n with period H_n.

Proof: (By induction on the number of tasks n)

The property is straightforward for the simple case where $n = 1$: indeed, the schedule for task τ_1 is periodic of period T_1 from its first release ($s_1 = r_1^1$) since $C_1 \leq T_1$, otherwise the deadline of the first instance is missed. Let us now assume that the property is true up to $n = i - 1$ and $\Gamma_i = \{\tau_1, \tau_2, \cdots, \tau_i\}$ is schedulable up to $s_i + H_i$, with $H_i = LCM(T_1, T_2, \cdots, T_i)$. Notice that s_i is the first release time of task τ_i after (or at) s_{i-1}. We have $s_i + H_i \geq s_{i-1} + H_{i-1}$ and by induction hypothesis, the subset $\Gamma_{i-1} = \{\tau_1, \tau_2, \cdots, \tau_{i-1}\}$ is schedulable and periodic from s_{i-1} of period H_{i-1}. As tasks are ordered by priority, the instances of the first ones are not changed by the requests of task τ_i and the schedule repeats at time $s_i + LCM(H_{i-1}, T_i) = s_i + H_i$. Consequently, $\Gamma_i = \{\tau_1, \tau_2, \cdots, \tau_i\}$ is schedulable and its schedule repeats from s_i with period H_i.

□

We now characterize the asynchronous scenario we consider in this paper in corollary 1. This leads to providing a simple method for computing the worst response time of each task in section 5 by using corollary 2, and then a pseudo polynomial FC detailed in section 6.1.

Corollary 1: From the point of view of any task τ_i of a schedulable system $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ ordered by decreasing priorities ($T_i \leq T_{i+1}, \forall i \in \{1, \cdots, n-1\}$) such that $T_i \mid T_{i+1}$ and $r_{i+1}^1 = r_i^1 - C_{i+1}$, the schedule is periodic from the second instance with period $H_i = T_i$.

Proof: (By induction on the index i of the task)

Let us consider a task τ_i of a schedulable system $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$, we assume that $T_i \mid T_{i+1}$ and $r_{i+1}^1 = r_i^1 - C_{i+1}, \quad \forall i \geq 1$. Thanks to the previous theorem, it is sufficient to prove that $s_i - r_i^1 = T_i, \quad \forall i \geq 2$. This is done by induction on i.

The property is straightforward for the simple case where $i = 2$: indeed, as $C_2 \leq T_2$ and $H_2 = LCM(T_1, T_2) = T_2$, the schedule for task τ_2 is periodic of period T_2 from its second release since $s_2 = r_2^1 + \left\lceil \dfrac{(s_1 - r_2^1)^+}{T_2} \right\rceil \cdot T_2 = r_2^1 + \left\lceil \dfrac{C_2}{T_2} \right\rceil \cdot T_2 = r_2^1 + T_2$ is the first release time of task τ_2 after (or at) $s_1 = r_1^1$. Let us now assume that the property is true up to index $i - 1$ and $\Gamma_i = \{\tau_1, \tau_2, \cdots, \tau_i\}$ is schedulable. Thanks to the previous theorem, we have

$$s_i = r_i^1 + \left\lceil \frac{(s_{i-1} - r_i^1)^+}{T_i} \right\rceil \cdot T_i = r_i^1 + \left\lceil \frac{(T_{i-1} + r_{i-1}^1 - r_i^1)^+}{T_i} \right\rceil \cdot T_i$$

by induction hypothesis.
Thus, $s_i = r_i^1 + \left\lceil \dfrac{(T_{i-1} + C_i)^+}{T_i} \right\rceil \cdot T_i$ since $r_{i-1}^1 = r_i^1 + C_i$.
Now, as $0 < T_{i-1} + C_i < T_i$ due to the scenario imposed

2. Given $a, b, c \in \mathbb{Z}$: $a = b \bmod [c]$ means that there exists $d \in \mathbb{Z}$ such that $a = b + cd$.

to the first instance of each task and the fact that $T_{i-1} \mid T_i$, then we obtain $s_i = r_i^1 + T_i$.

□

Corollary 2: The worst response time R_i of each task τ_i is obtained in the second instance and is equal to that in all instances greater than 2.

Proof:

Immediately follows from corollary 1 and the fact that $R_i^1 = C_i$ by construction, $R_i^k \geq C_i \quad \forall k \geq 1$, and we consider harmonic tasks.

□

4. A new scheduling representation

A direct consequence of corollary 2 leads us to the conclusion that in the case of a valid schedule, i.e. when all deadlines are met for all tasks, the schedule obtained at level i (the resulting schedule of the i tasks with the highest priority) is periodic with the period $T_i = LCM\{T_j | j = 1, \cdots, i\}$ from the second instance. As such, from the point of view of each task, the interval preceeding the second instance necessarily contains the *transient phase*, corresponding to the initial part of the schedule at level i, and the interval starting at date r_i^2 with the length T_i is isomorphic to the *permanent phase* of the schedule at level i, corresponding to the periodic part of the schedule. The transient phase is always finite due to the existence of the permanent phase and the permanent phase repeats indefinitely.

For a system of n periodic harmonic tasks for which there exists a valid schedule, since the permanent phase repeats indefinitely, we introduce a new scheduling representation. This scheduling representation is obtained by graphically using an *oriented circular disk* called *Dameid* with a reference time instant $t_0 = 0$ corresponding to the time reference in the *classical linear representation* or *Gantt Chart*. The positive direction in *Dameid* is the trigonometrical one, i.e. opposite to that of the hands of a watch. The circumference of *Dameid* at level n corresponds to $H_n = LCM\{T_i \mid i = 1, \cdots, n\}$ where T_i means the period of the i^{th} task and n denotes the number of tasks in the system. In *Dameid*, the different release times for each task are unambiguously determined by the value of their first release time relatively to that of other tasks with respect to the reference date $t_0 = 0$, and the ratio $\frac{H_n}{T_i}$ for task τ_i. As an example, figure 2 illustrates the release times of each task for a system consisting of 4 periodic harmonic tasks. In this figure, the first release time of task t_1 is -2, while that of task t_3 is 0.

Figure 3 clarifies our idea for the construction of *Dameid* for a given set of harmonic periodic tasks. This figure illustrates, for the same system with 4 tasks (see Figure 2), the correspondence of the release times of each periodic task in *Dameid* relative to the reference date $t_0 = 0$. The main intuition behind this new representation is to reduce

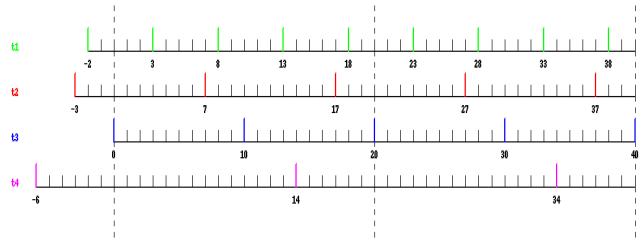

Figure 2. Release times of each task in the classical linear representation or Gantt Chart

the interval of analysis for a system harmonic periodic tasks whatever their first release times are.

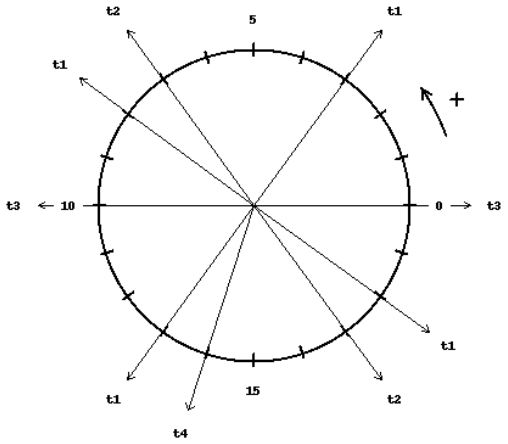

Figure 3. Release times of each task in Dameid

Now, in addition to the release times of each task, let us add the WCETs and explain how *Dameid* can represent schedules.

During the scheduling process from the highest priority task to the lowest priority task, some of the available time units at a given level i, i.e. those which are not executed after the schedule of the first $i-1$ highest priority tasks, are executed by time units corresponding to the WCET C_i of the current task τ_i. This is done in order to obtain the next result for the scheduling analysis of the next task τ_{i+1} with respect to the priorities. As the considered scheduling policy (DM) determines the *total* order in which to perform the scheduling analysis, it follows that the circular representation, i.e. *Dameid*, of circumference corresponding to the *LCM* of periods of all tasks that we have introduced allows us to build directly the *permanent phase* of the system if it is schedulable. Indeed, *Dameid* can be constructed completely independently from the linear representation. In this representation, the WCETs of the tasks correspond to angular sectors, where the angular unit is given by $\frac{1}{H_n}$ and

$H_n = LCM\{T_1, T_2, \cdots, T_n\}$.

Figure 6 shows an example of the *Dameid* for the system of which the schedule and the curve of response time as a function of time for each task are illustrated in figure 4 and figure 5. For this system, whose characteristics are summarized in table 1, we assume that task t_1 has a higher priority than task t_2, i.e. tasks are scheduled by using DM. In figure 4, the permanent phase is illustrated by the highlighted zone (blue zone). The curve of the response time of each task according to time (see figure 5) shows that from the time $t = 15$, the response time of each task is constant. We find this result by constructing the *Dameid*. Indeed, the *LCM* of the periods of both tasks t_1 and t_2 is given by $H_2 = LCM(5, 15) = 15$. The release times of task t_1 in *Dameid* with respect to its first release time are given by $r_1^1 = 4$, $r_1^2 = 9$ and $r_1^3 = 14$. For task t_2, we have a single release time equal to $r_2^1 = 0$ because its period $T_2 = H_2 = 15$. Since task t_1 has a higher priority than task t_2, then at each release time of t_1, i.e. at the dates r_1^1, r_1^2 and r_1^3, a sector corresponding to its worst case execution time ($C_1 = 2$ time units) is executed. As task t_2 has a lower priority than task t_1, the filling of the sectors of circumference corresponding to its worst case execution time ($C_2 = 4$ time units) can only be done between the time instants 1 and 4, then time instants 6 and 7. *Dameid* builds the permanent phase of the system directly: in figure 4, task t_2 has two distinct response times, 4 time units for the first activation and 7 time units afterwards, while in the circular representation through *Dameid*, it has a single response time, 7 time units, which corresponds to its response time in the permanent phase.

Tche	r_i^1	C_i	D_i	T_i
t_1	4	2	5	5
t_2	0	4	15	15

Table 1. Characteristics of the tasks

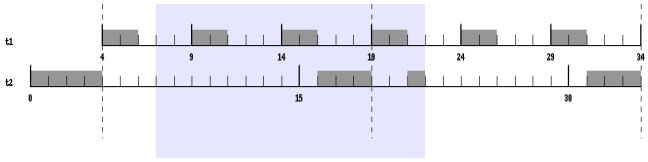

Figure 4. Linear representation / Gantt Chart of the schedule.

This new representation of the schedule is more interesting than the linear representation / Gantt Chart because it is more *compact* and puts greater emphasis on the *available time units* in the resulting schedule. In his thesis ([24]), *Joel Goossens* suggested that the permanent phase is sufficient to guaranteeing the schedulability of a given periodic task set

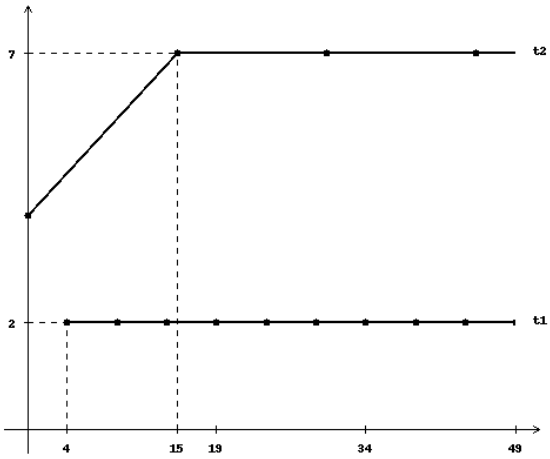

Figure 5. Response time of each task as a function of time.

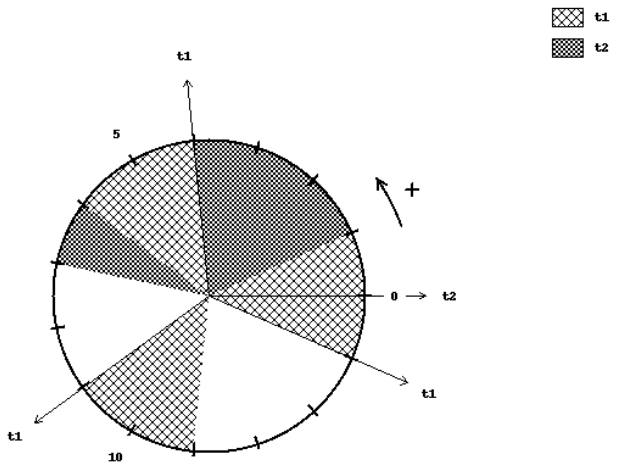

Figure 6. Circular representation of the schedule by using *Dameid*.

when the cost of preemption is neglected and this permanent phase is directly built by using *Dameid*. We now suppose the asynchronous task set defined in corollary 1 and present the Mesoid approach used to compute the worst case response time of each periodic task.

5. Worst case response time: the Mesoid approach

In this section we provide the method for computing the worst response time of each task in order to check its schedulability. Actually, three classical methods may be used to do so: the utilisation factor of the processor ([25]), the worst response time of each task, or the processor

demand ([26]). In this paper we have chosen to use the second approach as it provides a schedulability condition for each task individually. The main idea behind the Mesoid approach is to fill some available time units left by the schedule of higher priority tasks with executed time units corresponding to the execution time of the current task. Since the worst response time is obtained in the second instance w.r.t. corollary 2, we will achieve this goal by applying the method described in [4] to a system where the tasks are not all released simultaneously and where the cost of a preemption is assumed to be zero. This method, unlike those proposed in ([27], [7], [28]), is of lesser complexity since it is not necessary to determine the releases of every task w.r.t. those of higher priority tasks.

As we are in a fixed priority context, the proposed method checks for the schedulability of each task by computing its worst response time, from the task with the highest priority to that with the lowest priority. Hence, from the point of view of any task τ_i of a system $\Gamma_n = \{\tau_1, \tau_2, \cdots, \tau_n\}$ ordered by decreasing priorities ($T_{i-1} \leq T_i, \forall i \in \{2, \cdots, n\}$) such that $T_{i-1}|T_i$ and $r_i^1 = r_{i-1}^1 - C_i$, the elapsed duration between the release of the second instance and the first release r_{i-1}^1 of task τ_{i-1} is given by $T_i - C_i$. Before providing the computation method of the worst case response time, we provide some necessary definitions below.

5.1. Definitions

All the definitions and terminologies used in this section are directly inspired by ([4]) and are applied here to the case of a model where the cost of preemption is assumed to be zero. From the point of view of any task τ_i, the *hyperperiod at level i*, H_i, is given by $H_i = LCM\{T_j\}_{\tau_j \in sp(\tau_i)} = T_i$ as $T_{i-1}|T_i$ for every $i \in \{2, \cdots, n\}$, and $sp(\tau_i)$ is the set of tasks with a period shorter than that of task τ_i. Without any loss of generality we assume that the first task τ_1 starts its execution at time $t = 0$ and that all tasks have different periods. Since at each level the schedule repeats indefinitely from the second instance thanks to corollary 1, it is sufficient to perform the scheduling analysis in the interval $[r_i^1 + T_i, r_i^1 + 2T_i]$ for task τ_i as its response time in its first instance equals its WCET.

We proceed the schedule from the task with the shortest period towards the task with the longest period. Thus, at each level in the scheduling process the goal is to fill available time units in the previous schedule, obtained up to now, with slices of the WCET of the current task, and hence we obtain the next current schedule. Consequently, we represent the previous schedule of every instance τ_i^k of the current task $\tau_i = (C_i, T_i)$ by an ordered set of T_i time units where some have already been executed because of the execution of tasks with shorter periods, and the others are still available for the execution of task τ_i in that instance. We call this ordered set which describes the state of each instance τ_i^k the $\mathcal{M}_i^k$ T_i-*mesoid*. More details on the definition of a T_i-*mesoid* are given in [4]. For the current task $\tau_i = (C_i, T_i)$, there are as many T_i-mesoids as instances. We call $\mathcal{M}_i^{b,2}$ the T_i-*mesoid* corresponding to the second instance of task τ_i **before** being scheduled in the current schedule. The process used to build $\mathcal{M}_i^{b,2}$ for task τ_i will be detailed later in this subsection. Still, from the point of view of task τ_i, we define for the mesoid $\mathcal{M}_i^{b,2}$ the corresponding *universe* X_i^2 to be the ordered set, compatible with that of the mesoid, which consists of all the availabilities of $\mathcal{M}_i^{b,2}$ – that is to say, all the possible values that C_i can take in $\mathcal{M}_i^{b,2}$. Task τ_i will be said to be *potentially schedulable* if and only if

$$C_i \in X_i^2 \quad \forall\, i \in \{1, \cdots, n\} \tag{2}$$

This equation verifies that C_i belongs to the universe at level i. If it does not, then the system is clearly not schedulable. When equation (2) holds for a given task τ_i, we call $\mathcal{M}_i^{a,2}$ the T_i-*mesoids* corresponding to the second instance of task τ_i **after** τ_i has been scheduled. $\mathcal{M}_i^{a,2}$ is a function of $\mathcal{M}_i^{b,2}$ which itself is a function of $\mathcal{M}_{i-1}^{a,2}$, both detailed as follows.

Let f be the function such that $\mathcal{M}_i^{b,2} = f(\mathcal{M}_{i-1}^{a,2})$ which transforms the T_{i-1}-mesoid after task τ_{i-1} has been scheduled at level $i-1$ into the T_i-mesoid before task τ_i is scheduled at level i.

As mentioned in [4], a mesoid consists only of time units already executed denoted by "e" and time units still available denoted by "a". Moreover, the cardinal of a mesoid is equal to the period of the task under consideration whatever the level is. As such, the function f transforms a time unit already executed (resp. still available) in $\mathcal{M}_{i-1}^{a,2}$ into a time unit already executed (resp. still available) in $\mathcal{M}_i^{b,2}$ by following an index ψ which enumerates, according to naturals, the time units (already executed or still available) in $\mathcal{M}_{i-1}^{a,2}$ of task τ_{i-1} after τ_{i-1} has been scheduled. As the elapsed duration between the release of the second instance of task τ_i and the release of the first instance of τ_{i-1} is $T_i - C_i$, then ψ starts from the time unit right after $\gamma_i = T_i - C_i \bmod [T_{i-1}]$ time units in the mesoid $\mathcal{M}_{i-1}^{a,2}$ towards the last time unit, and then circles around to the beginning of the mesoid $\mathcal{M}_{i-1}^{a,2}$ again, until we get the T_i-mesoid $\mathcal{M}_i^{b,2}$. This T_i-mesoid is obtained when $\psi = T_i$. Indeed, the previous schedule at level i (the schedule obtained at level $i-1$) consists of $H_{i-1} = T_{i-1}$ time units whereas the schedule of the current task τ_i is computed upon $H_i = T_i$ time units. Thus, that amounts to extending the previous schedule from T_{i-1} to T_i time units by identically repeating the previous schedule as often as necessary to obtain H_i time units. Due to the particular releases of the first instance of each task, i.e. $r_{i+1}^1 = r_i^1 - C_{i+1}$ $\forall\, i \in \{1, \cdots, n-1\}$, notice that index ψ in contrast to index ζ used in [4] which started from the first time unit, starts from the time unit right after $\gamma_i = T_i - C_i \bmod [T_{i-1}]$ time units in the mesoid $\mathcal{M}_{i-1}^{a,2}$. Since τ_1 is the task with the shortest

period, then $sp(\tau_1) = \{\tau_1\}$. Because τ_1 is never preempted, we have $\mathcal{M}_1^{b,2} = \{1, 2, \cdots, T_1\}$ and therefore we obtain $\mathcal{M}_1^{a,2} = \{(C_1), 1, 2, \cdots, T_1 - C_1\}$.

Let g be the function such that $\mathcal{M}_i^{a,2} = g(\mathcal{M}_i^{b,2})$ which transforms the T_i-mesoid $\mathcal{M}_i^{b,2}$ before task τ_i has been scheduled at level i into the T_i-mesoid $\mathcal{M}_i^{a,2}$ after task τ_i has been scheduled at level i.

5.2. Worst case response time with a Mesoid

For the T_i-mesoid $\mathcal{M}_i^{b,2}$, we will compute the response time R_i^2 of task τ_i in the second instance by adding to the WCET C_i all the consumptions appearing in that T_i-mesoid before the availability corresponding to C_i [4]. This yields the worst-case response time R_i of task τ_i since at each level the schedule becomes periodic from the second instance, that is to say $R_i^k = R_i^2 \ \forall\, k \geq 2$, and $R_i^1 = C_i \ \forall\, i \geq 1$.

Now we can build $\mathcal{M}_i^{a,2} = g(\mathcal{M}_i^{b,2})$: function g transforms a time unit already executed in $\mathcal{M}_i^{b,2}$ into a time unit already executed in $\mathcal{M}_i^{a,2}$, and transforms a time unit still available into either a time unit still available or a time unit already executed w.r.t. the following condition. We use an index which enumerates according to numerals the time units in $\mathcal{M}_i^{b,2}$ from the first to the last one, at each step in the incremental process, if the current value of the index is less than or equal to R_i^2, function g transforms the time unit still available into a time unit already executed due to the execution of instance τ_i^2, otherwise g transforms it into a time unit still available. Indeed, function g fills available time units in the current schedule with slices of the WCET in each T_i-mesoid, leading to the previous schedule for the next task at level $i+1$ w.r.t. priorities. To summarize, for every task τ_i, we have

$$\tau_i : \begin{cases} \mathcal{M}_i^{b,2} : T_i\text{-mesoid before } \tau_i \text{ is scheduled at level } i \\ \\ \mathcal{M}_i^{a,2} : T_i\text{-mesoid after } \tau_i \text{ is scheduled at level } i. \end{cases}$$

6. Deadline reduction factor

6.1. Worst case response time computation

The approach proposed here leads to a new schedulability condition for harmonic hard real-time systems. This condition is new in the sense that in addition to providing a necessary and sufficient schedulability condition, it also reduces the feasibility interval for a given harmonic asynchronous system.

In the scheduling process, at each level i, the basic idea consists in filling availabilities in the mesoid $\mathcal{M}_i^{b,2}$ before task τ_i is scheduled, with slices of its WCET. This is why it is fundamental to calculate the corresponding response time. This yields the worst case response time and allows us to conclude on the schedulability of task τ_i w.r.t. priorities. In the case where τ_i is schedulable, we build $\mathcal{M}_i^{a,2}$, after τ_i has been scheduled, in order to check the schedulability of the next task, and so on, otherwise the system is not schedulable. Thanks to everything we have presented up to now, τ_1 is scheduled first and $r_1^1 = 0$. The latter statement implies that **before** τ_1 is scheduled, its WCET can potentially take any value from 1 up to the value of its period T_1. Since task τ_1 is never preempted, then $\mathcal{M}_1^{b,2} = \{1, 2, \cdots, T_1\}$ and $X_1^2 = \{1, 2, \cdots, T_1\}$. Moreover, its response time is also equal to C_1. Consequently, the corresponding T_1-*mesoids* associated to task τ_1 are given by

$$\tau_1 : \begin{cases} \mathcal{M}_1^{b,2} = \{1, 2, \cdots, T_1\} \\ \\ \mathcal{M}_1^{a,2} = \{(C_1), 1, 2, \cdots, T_1 - C_1\} \end{cases}$$

We assume that the first $i-1$ tasks with $2 \leq i \leq n$ have already been scheduled, i.e. the T_{i-1}-mesoid $\mathcal{M}_{i-1}^{a,2}$ of task τ_{i-1} is known, and that we are about to schedule task τ_i.

As explained in the previous section, the T_i-mesoid $\mathcal{M}_i^{b,2} = f(\mathcal{M}_{i-1}^{a,2})$ of task τ_i is built thanks to index ψ on $\mathcal{M}_{i-1}^{a,2}$ of task τ_{i-1} without forgetting to start from the time unit right after $\gamma_i = T_i - C_i \text{mod}[T_{i-1}]$ time units rather than the first time unit as in [4]. Again this is due to the particular release of the first instances of tasks: $r_i^1 = r_{i-1}^1 - C_i$. We can therefore determine the universe X_i^2 when the T_{i-1}-mesoid $\mathcal{M}_{i-1}^{a,2}$ is known. Unless the system is not schedulable, i.e. $C_i \notin X_i^2$, we assume that task τ_i is potentially schedulable, i.e. $C_i \in X_i^2$. The response time R_i^2 of task τ_i in its k^{th} instance (with $k \geq 2$), i.e. in the k^{th} T_i-*mesoid* will be obtained by summing C_i with all consumptions prior to C_i in the corresponding mesoid. The worst-case response time R_i of task τ_i will then be given by

$$R_i = R_i^2$$

This equation leads us to say that task τ_i is schedulable if and only if

$$R_i \leq T_i \tag{3}$$

If for task τ_i expression (3) holds, then $\mathcal{M}_i^{a,2} = g(\mathcal{M}_i^{b,2})$ will be deduced as explained in the previous section. For the sake of clarity, whenever there are two consecutive consumptions in a *mesoid*, this amounts to considering only one consumption which is the sum of the previous consumptions. That is to say that after determining the response time of task τ_i in its k^{th} mesoid, if $\mathcal{M}_i^{a,k} = \{(c_1), (c_2), 1, 2, \cdots\}$, then this is equivalent to $\mathcal{M}_i^{a,k} = \{(c_1 + c_2), 1, 2, \cdots\}$.

Below, we present our scheduling algorithm which, for a given task, on the one hand first determines the value of $\gamma_i = T_i - C_i \text{ mod } [T_{i-1}]$ relative to priorities, then, on the other hand the schedulability condition. Recall that the elapsed duration between the release of the second instance and the first release is $T_i - C_i$. The scheduling algorithm has the following nine steps. Since the task with the shortest

period, namely task τ_1, is never preempted, the loop starts from the index of the task with the second shortest period, namely task τ_2 as the schedule proceeds towards tasks with longer periods.

1: **for** $i = 2$ to n **do**

2: Determine the release time of the first instance of task τ_i:

$$r_i^1 = r_{i-1}^1 - C_i$$

and compute $\gamma_i = T_i - C_i \bmod [T_{i-1}]$ of the second instance of τ_i w.r.t. τ_{i-1}.

3: Build the T_i-mesoid $\mathcal{M}_i^{b,2} = f(\mathcal{M}_{i-1}^{a,2})$ of task τ_i before it is scheduled. This construction is based on a modulo T_i arithmetic using index ψ on $\mathcal{M}_{i-1}^{a,2}$ without forgetting to start from the time unit right after $\gamma_i = T_i - C_i \bmod [T_{i-1}]$ time units rather than the first time unit as in [4]. This is due to the particular release of tasks.

4: For the T_i-*mesoid* $\mathcal{M}_i^{b,2}$ resulting from the previous step, build the corresponding universe X_i^2 which consists of the ordered set of all availabilities of $\mathcal{M}_i^{b,2}$. Notice that this set corresponds to the set of all possible values that the WCET C_i of task τ_i can take in $\mathcal{M}_i^{b,2}$.

5: Since τ_i is potentially schedulable, i.e. its WCET $C_i \in X_i^2$, we must verify that it is actually schedulable. Clearly, if $C_i \notin X_i^2$, then task τ_i is not schedulable because the deadline of the task is exceeded.

6: Determine the response time R_i^k of task τ_i in its k^{th} instance, i.e. in the k^{th} T_i-*mesoid*. This is obtained by summing C_i with all the consumptions prior to C_i in the corresponding mesoid. Deduce the worst-case response time R_i of task τ_i.

$$R_i = R_i^2$$

It is worth noticing that task τ_i is schedulable if and only if

$$R_i \leq D_i.$$

7: If $R_i \leq D_i$, then build $\mathcal{M}_i^{a,2} = g(\mathcal{M}_i^{b,2})$, increment i, and go back to step 2 as long as there remain potentially schedulable tasks in the system.

8: If $R_i > D_i$, then the system $\{\tau_i = (C_i, T_i)\}_{1 \leq i \leq n}$ is not schedulable.

9: **end for**

Thanks to the above algorithm, a system of n tasks $\{\tau_i = (C_i, T_i)\}_{1 \leq i \leq n}$, with harmonic periods and first released such that $r_i^1 = r_{i-1}^1 - C_i$, is schedulable if and only if

$$R_i = R_2^2 \leq D_i \qquad \forall i \in \{1, 2, \cdots, n\} \tag{4}$$

6.2. Computation of α

The value of α is given by: $\alpha = \max_{1 \leq i \leq n} \left(\frac{R_i}{T_i} \right)$.

This value of α guarantees that no task fails at run-time. We recall that for the synchronous scenario, the worst case response time of task τ_i is given by:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Example

Let us consider $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ to be a system of four tasks with harmonic periods and first released such that $r_i^1 = r_{i-1}^1 - C_i$. The characteristics are defined in table 2.

Table 2. Characteristics of the tasks

	C_i	T_i
τ_1	2	5
τ_2	4	15
τ_3	5	30
τ_4	7	60

The shorter the period of a task is, the higher its level is. Thus, as depicted in table 2, τ_1 has the highest level and task τ_4 the lowest level. Thanks to our scheduling algorithm, for task τ_1 whose first release time is $r_1^1 = 0$, we have

$$\tau_1 : \begin{cases} \mathcal{M}_1^{b,2} = \{1, 2, 3, 4, 5\} \\ R_1 = 2 \\ \mathcal{M}_1^{a,2} = \{(2), 1, 2, 3\} \end{cases}$$

$\gamma_2 = T_2 - C_2 \bmod [T_1] = 15 - 4 \bmod [5] = 1$, thus for task τ_2 whose first release time is $r_2^1 = r_1^1 - C_2 = -4$, we have

$$\tau_2 : \begin{cases} \mathcal{M}_2^{b,2} = \{(1), 1, 2, 3, (2), 4, 5, 6, (2), 7, 8, 9, (1)\} \\ R_2 = 4 + 2 + 1 = 7 \\ \mathcal{M}_2^{a,2} = \{(7), 1, 2, (2), 3, 4, 5, (1)\} \end{cases}$$

$\gamma_3 = T_3 - C_3 \bmod [T_2] = 30 - 5 \bmod [15] = 10$, thus for task τ_3 whose first release time is $r_3^1 = r_2^1 - C_3 = -4 - 5 = -9$, we have

$$\tau_3 : \begin{cases} \mathcal{M}_3^{b,2} = \{(1), 1, 2, 3, (8), 4, 5, (2), 6, 7, 8, (8), 9, 10, (1)\} \\ R_3 = 5 + 8 + 1 = 14 \\ \mathcal{M}_3^{a,2} = \{(16), 1, 2, 3, (8), 4, 5, (1)\} \end{cases}$$

$\gamma_4 = T_4 - C_4 \bmod [T_3] = 60 - 7 \bmod [30] = 23$, thus for task τ_4 whose first release time is $r_4^1 = r_3^1 - C_4 = -9 - 7 = -16$, we have

$$\tau_4 : \begin{cases} \mathcal{M}_4^{b,2} = \{(4), 1, 2, (17), 3, 4, 5, (8), 6, 7, (17), 8, 9, 10, (4)\} \\ R_4 = 7 + 8 + 17 + 4 = 36 \\ \mathcal{M}_4^{a,2} = \{(53), 1, 2, 3, (4)\} \end{cases}$$

Consequently, the set of tasks $\{\tau_1, \tau_2, \tau_3, \tau_4\}$ with harmonic periods and first released such that $r_i^1 = r_{i-1}^1 - C_i$ is schedulable. The schedule with the above characteristics

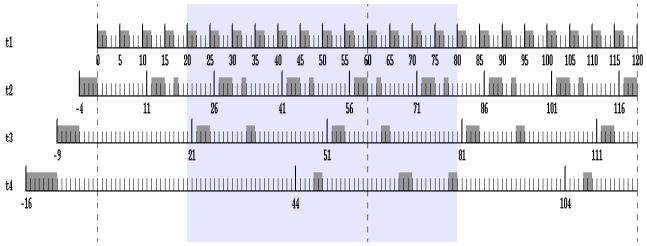

Figure 7. Execution of a set of harmonic tasks with $r_i^1 = r_{i-1}^1 - C_i, \ \forall\, i \in \{2, \cdots, 4\}$

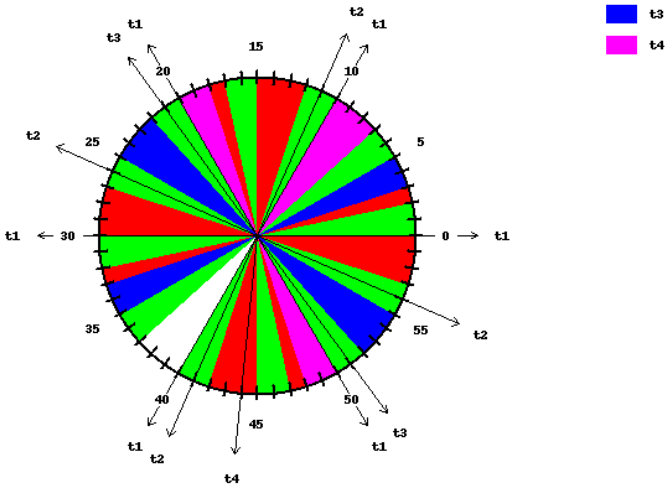

Figure 8. Circular representation of the schedule for a set of harmonic tasks with $r_i^1 = r_{i-1}^1 - C_i, \forall i \in \{2, \cdots, 4\}$

is depicted in figure 7 and the circular representation of the schedule by using *Dameid* is depicted in figure 8.
The schedule of the same set of tasks released simultaneously is depicted in figure 9 and the circular representation of the schedule by using *Dameid* is depicted in figure 10.

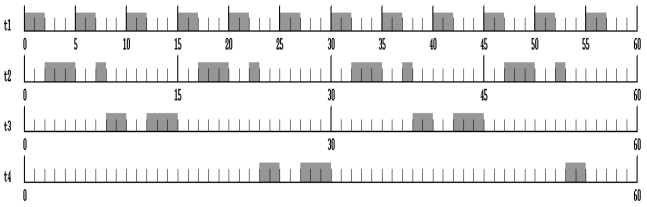

Figure 9. Execution of a set of harmonic tasks with $r_i^1 = 0 \ \forall\, i \in \{1, \cdots, 4\}$

It is worth noticing here the large variation between the two scenarios in terms of the tasks' response times. In fact, the worst case response time of task τ_4 in figure 7 and figure

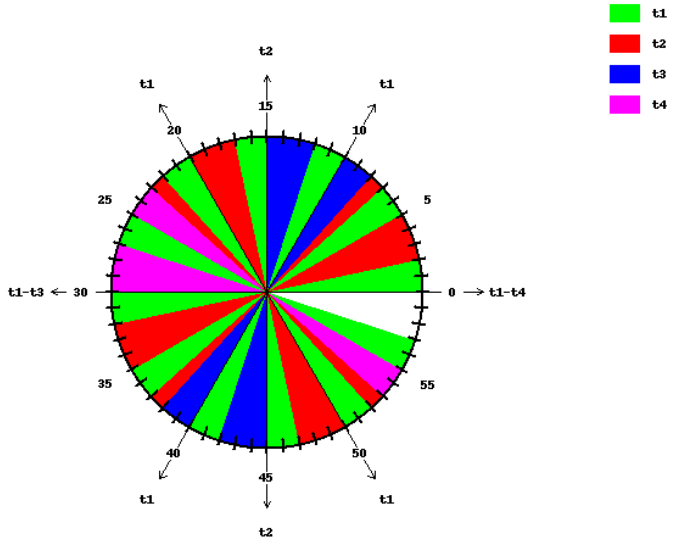

Figure 10. Circular representation of the schedule for a set of harmonic tasks with $r_i^1 = 0 \ \forall\, i \in \{1, \cdots, 4\}$

8 is 36 time units whereas it is 55 time units in figure 9 and figure 10. This phenomenon is even more apparent in the next section with the experimental results where we gradually and uniformly decrease the value of the relative deadlines for all tasks by the same factor to highlight the advantage of our approach.

Tasks	$R_i^{synchronous}$	$R_i^{asynchronous}$
τ_1	2	2
τ_2	8	7
τ_3	15	14
τ_4	55	36

This leads us to obtain $\alpha^{synchrnous} = max(2/5, 8/15, 15/30, 55/60) = 0.91$ whereas $\alpha^{asynchrnous} = max(2/5, 7/15, 14/30, 36/60) = 0.60$, which means the improvement performed in this case is of 34.54%

7. Experimental results

In this section we present some experimental results found by using the approach we have developed above. To achieve these experimental results, we proceed in two steps. First, we compare the minimum deadline reduction factor α obtained in the synchronous scenario with that obtained in our specific asynchronous scenario. Second, we extend this comparison concerning the value α to the value of α obtained for an arbitrarily generated scenario of the first release times for all tasks. This extension is performed by using more extensive experiments in order to get more accurate conclusions with

regard to the contributions of the proposed approach. As in ([1]), we consider a set of harmonic tasks scheduled with the Deadline Monotonic algorithm.

The first step in our process of comparing the value of α for given scenarii of first release for all tasks consists in performing 10000 experiments for each graph, where every task set consists of $n = 10$ harmonic tasks. The total utilization factor of the processor is randomly chosen between 0.7 and 1 for each task set. Hence, we can evaluate the gain of our specific asynchronous scenario defined in corollary 1 in section 3, compared to the synchronous one. We set $\alpha = \frac{D_i}{T_i}$, and we gradually and uniformly decrease the value of the relative deadlines D_i by the same factor for all tasks in each set. In both the synchronous and the asynchronous scenario, we plot the curves corresponding to the smallest value of α, as a function of the total utilization factor of the processor, for the task set to remain schedulable. The resulting graphic is displayed in figure 11. If the value of α is denoted $\alpha^{synchronous}$ in the synchronous scenario and $\alpha^{asynchronous}$ in our asynchronous scenario, the gain can be computed as follows:

$$gain = \frac{\alpha^{synchronous} - \alpha^{asynchronous}}{\alpha^{synchronous}} \times 100$$

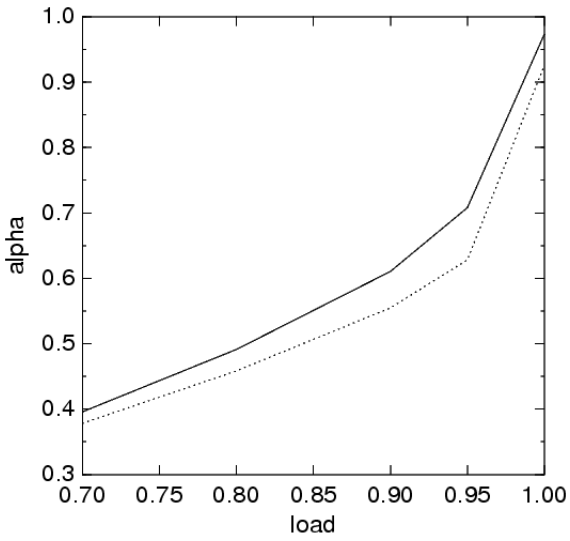

Figure 11. Value of α with our asynchronous scenario and with the synchronous scenario

In figure 11, the solid curve represents the result obtained for α in our specific asynchronous case whereas the dotted curve represents the result obtained in the synchronous case. In both cases, we start with a schedulable task set $\forall \tau_i, D_i = T_i$. From [20], $U \leq 1$ is a necessary and sufficient condition for the schedulability of a harmonic task set as tasks are scheduled with DM, equivalent to RM when $\forall \tau_i, D_i = T_i$. We can see that for a small load, we obtain almost the same α both in the synchronous and in the specific asynchronous cases.

Concerning the second step in our process of comparing the value of α for given scenarii of first release times for all tasks, we perform twice as many experiments than for the first step. That is to say, we perform 20000 experiments for each graph, and every task set still consists of $n = 10$ harmonic tasks. Again, the total utilization factor of the processor is randomly chosen between 0.7 and 1 for each task set. As such, we can evaluate the gain of α obtained in our specific asynchronous scenario, compared to that obtained in the synchronous scenario on the one hand, and to the mean value obtained for a set of arbitrarily generated scenarii on the other hand. As for the first step, we set $\alpha = \frac{D_i}{T_i}$, and we gradually and uniformly decrease the value of the relative deadlines D_i by the same factor for all tasks in each set. For the synchronous, and the asynchronous scenarii, we plot the curves corresponding to the smallest value of α. For the set of arbitrarily generated scenarii, we plot the curves corresponding to the mean value of α. This is performed in each case as a function of the total utilization factor of the processor, for the task set to remain schedulable. The curves obtained are displayed in figure 12.

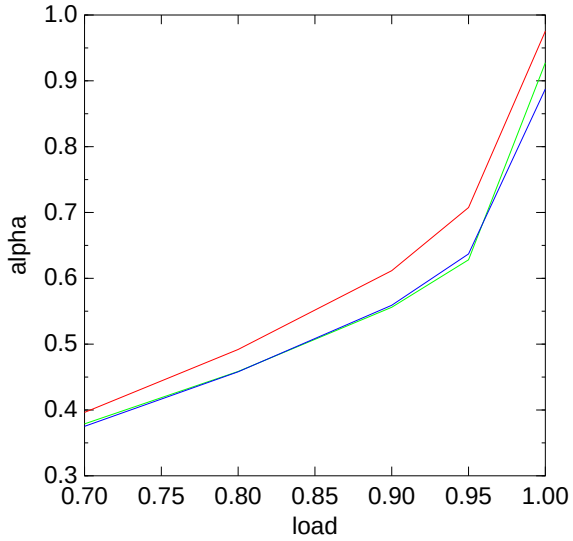

Figure 12. Value of α with our asynchronous scenario, then with the synchronous scenario and the mean of a set of arbitrarily generated scenarii

In figure 12, the curve in *red* represents the result obtained for α by using our specific asynchronous scenario. The curve in *green* represents the result obtained for the synchronous case and the curve in *blue* represents the mean value obtained for a set of arbitrarily generated scenarii. In all the cases, we start with a schedulable task set $\forall \tau_i, D_i = T_i$ and $U \leq 1$ remains a necessary and sufficient condition for the schedulability of a harmonic task set as tasks are scheduled

with DM. It is worth noticing that DM is equivalent to RM when $\forall \tau_i, D_i = T_i$.

We can see that we always obtain almost the same value for α both in the synchronous case and for the mean value obtained for a set of arbitrarily generated scenarii.

For a small load, the value of α varies very slightly whatever the scenario of first release for all tasks is. In both steps, this is due to the fact that with a small load the worst case response times of the tasks are less influenced by the first release times of other tasks. When the load increases, the gain also increases, reaches and remains at a maximum of 14.3% for $U = 0.95$. Over the load $U = 0.95$, the gain steadily decreases when U tends to 1 and α tends to 1. At high loads, the worst case response time of a task tends to its period and thus α tends to 1. In this latter case, the improvement obtained with our spacific asynchronous scenario becomes less significant.

8. Conclusion

In this paper we have proposed a new approach for a better control of periodic tasks scheduled with Deadline Monotonic scheduling algorithm. We have considered a specific asynchronous task set and harmonic tasks that enables us to significantly reduce the worst case response time of each task thus reducing the jitter of each task for a better control. The asynchronous scenario we considered makes it possible to significantly reduce the complexity of the worst case response time computation. We have then considered the Mesoid approach to compute the worst case response time of a task in an asynchronous scenario. We have used the Mesoid approach to compute the minimum deadline reduction factor characterizing the benefit in terms of worst case response time reduction. We have proved by extensive simulations that the gain in terms of deadline reduction can reach 14.3% with our particular asynchronous scenario compared to the synchronous scenario and to an arbitrarily generated scenario. This makes it possible to better control the jitter of the tasks when considering control loops. Future work will compare the deadline reduction factor obtained with EDF with the one we have obtained with our specific asynchronous scenario.

References

[1] P. Meumeu Yomsi, L. George, Y. Sorel, and D. De Rauglaudre. Improving the Sensitivity of Deadlines with a Specific Asynchronous Scenario for Harmonic Periodic Tasks scheduled by FP. *The Fourth International Conference on Systems (ICONS'09)*, Cancun, Mexico, March 1 - 6 2009.

[2] Giorgio Buttazzo Enrico Bini, Marco Di Natale. Sensitivity Analysis for Fixed-Priority Real-Time Systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, Dresden, Germany July 5-7, 2006.

[3] Ismael Ripoll Patricia Balbastre and Alfons Crespo. Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, Dresden, Germany July 5-7, 2006.

[4] P. Meumeu Yomsi and Sorel Y. Extending Rate Monotonic Analysis with Exact Cost of Preemptions for Hard Real-Time Systems. *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*, Pisa, Italy, Jul. 2007.

[5] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor. *Real-Time Systems*, Vol. 2, pp. 301-324, 1990.

[6] K. Tindell, A. Burns, and A. J. Wellings. Analysis of hard real-time communications. *Real-Time Systems*, Vol. 9, pp. 147-171, 1995.

[7] L. George, N. Rivierre, and M. Spuri. Preemptive and non-preemptive scheduling real-time uniprocessor scheduling. *INRIA Research Report*, No. 2966, September 1996.

[8] E. Bini and G. Buttazzo. The Space of EDF Feasible Deadlines. *Proceedings of the 19th Euromicro Conference on Real-Time Systems (ECRTS'07)*, Pisa, Italy, July 4-6 2007.

[9] S. Baruah, G. Buttazo, S. Gorinsky, and G. Lipari. Scheduling periodic task systems to minimize output jitter. *In 6^{th} Conference on Real-Time Computing Systems and Applications*, pp. 62-69, 1999.

[10] A. Cervin, B. Lincoln, J. Eker, K. Arzen, and Buttazzo G. The jitter margin and its application in the design of real-time control systems. *In proceedings of the IEEE Conference on Real-Time and Embedded Computing Systems and Applications*, 2004.

[11] D. Marinca, P. Minet, and L. George. Analysis of deadline assignment methods in distributed real-time systems. *Computer Communications*, Elsevier, To appear, 2004.

[12] M. Joseph and P. Pandya. Finding response times in a real-time system. *BCS Comp. Jour.*, 29(5), pp. 390-395,, 1986.

[13] M. Grenier, J. Goossens, and N. Navet. Near-optimal fixed priority preemptive scheduling of offset free systems. *Proc. of the 14th International Conference on Network and Systems (RTNS'2006)*, Poitiers, France, May 30-31, 2006 2006.

[14] Giorgio Buttazzo Enrico Bini. Schedulability Analysis of Periodic Fixed Priority Systems. *IEEE Transactions On Computers, Vol. 53, No. 11*, Nov.2004.

[15] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. *Proceedings 11th IEEE Real-Time Systems Symposium*, pp 201-209, Dec. Lake Buena Vista, FL, USA, 1990.

[16] J. Y. T. Leung and M.L. Merril. A note on premptive scheduling of periodic, Real Time Tasks. *Information Processing Letters*, Vol 11, num 3, Nov. 19980.

[17] Annie Choquet-Geniet and Emmanuel Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Theor. Comput. Sci.*, 310(1-3):117–134, 2004.

[18] N. C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times. *Dept. Comp. Science Report YCS 164, University of York*, 1991.

[19] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2):239-258, March 2003.

[20] G. C. Buttazzo. Rate Monotonic vs. EDF: Judgment Day. *Real-Time Systems, 29, 5-26*, 2005.

[21] E. Bini and A. Cervin. Delay-Aware Period Assignment in Control Systems. *Proceedings of the 26th IEEE International Real-Time Systems Symposium (RTSS'08)*, Barcelona, Spain, Nov. 30 to Dec. 3 2008.

[22] S. Baruah, A. K. Mok, and L. Rosier. Preemptively scheduling hard real-time sporadic tasks on one processor. *Proceedings of the 11th Real-Time Systems Symposium*, pp. 182-190, 1990.

[23] L. C. Liu and W. Layland. Scheduling algorithms for multiprogramming in a hard real time environment. *Journal of ACM*, Vol. 20, No 1, pp. 46-61, January 1973.

[24] J. Goossens. *Scheduling of Hard Real-Time Periodic Systems with Various Kinds of Deadline and Offset Constraints*. PhD thesis, Université Libre de Bruxelles, 1998.

[25] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 1973.

[26] A.K. Mok S.K. Baruah and L.E. Rosier. Preemptively scheduling hard realtime sporadic tasks on one processor. *In proc. 11th IEEE Real-Time Systems Symposium*, 1990.

[27] Joseph Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 1980.

[28] J. Leung and Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation(4)*, 1982.