Exact Schedulability Tests for Real-Time Scheduling of Periodic Tasks on Unrelated Multiprocessor Platforms

Liliana Cucu-Grosjean[a,1,*], Joël Goossens[b]

[a] *INRIA Nancy Grand Est, 615 rue du Jardin Botanique, 54600 Villers les Nancy, France*
[b] *Université Libre de Bruxelles, 50 Avenue Franklin D. Roosevelt, 1050 Brussels, Belgium*

Abstract

In this paper, we study the global scheduling of periodic task systems on unrelated multiprocessor platforms. We first show two general properties which are well-known for uniprocessor platforms and which are also true for unrelated multiprocessor platforms: *(i)* under few and not so restrictive assumptions, we prove that feasible schedules of periodic task systems are periodic starting from some point in time with a period equal to the least common multiple of the task periods and *(ii)* for the specific case of synchronous periodic task systems, we prove that feasible schedules repeat from their origin. We then present our main result: we characterize, for task-level fixed-priority schedulers and for asynchronous constrained or arbitrary deadline periodic task models, *upper bounds* of the first time-instant where the schedule repeats. For task-level fixed-priority schedulers, based on the upper bounds and the predictability property, we provide *exact* schedulability tests for asynchronous constrained or arbitrary deadline periodic task sets. Finally, we provide an *exact* schedulability test as well for the *job*-level fixed-priority Earliest Deadline First (EDF) scheduler, for which such an upper bound is unknown.

Keywords: Real-time systems, multiprocessors, scheduling

*Corresponding author
Email addresses: liliana.cucu@loria.fr (Liliana Cucu-Grosjean), joel.goossens@ulb.ac.be (Joël Goossens)

[1] This work was partially supported by the Fonds National de la Recherche Scientifique de Belgique.

1. Introduction

The use of computers to control safety-critical real-time functions has increased rapidly over the past few years. As a consequence, real-time systems — computer systems where the correctness of each computation depends on both the logical results of the computation and the time at which these results are produced — have become the focus of much study. Since the concept of "time" is of such importance in real-time application systems, and since these systems typically involve the sharing of one or more resources among various contending processes, the concept of scheduling is integral to real-time system design and analysis. Scheduling theory, as it pertains to a finite set of requests for resources, is a well-researched topic. However, requests in a real-time environment are often of a recurring nature. Such systems are typically modelled as finite collections of simple, highly repetitive tasks, each of which generates jobs in a predictable manner. In this work, we consider *periodic task systems*, where each periodic task τ_i generates jobs at each integer multiple of its period T_i with the restriction that the first job is released at time O_i (the task offset).

The *scheduling algorithm* determines which job[s] should be executed at each time-instant. When there is at least one schedule satisfying all constraints of the system, the system is said to be *schedulable*.

Uniprocessor real-time systems have been well studied since the seminal paper of Liu and Layland [15], which introduces a model of periodic systems. There is a tremendous amount of literature considering scheduling algorithms, feasibility tests and schedulability tests for uniprocessor scheduling. In contrast, for *multiprocessor* parallel machines, the problem of meeting timing constraints is a relatively new research area.

In the design of scheduling algorithms for multiprocessor environments, one can distinguish at least two distinct approaches. In *partitioned scheduling*, all jobs generated by a task are required to execute on the *same* processor. *Global scheduling*, by contrast, permits *task migration* (i.e., the different jobs of an individual task may be executed on different processors) as well as *job migration* (an individual job that is preempted may resume execution on a different processor than the one it was being

executed on prior to preemption).

From a theoretical point of view, we can distinguish at least three kinds of multiprocessor machines (from less general to more general):

Identical parallel machines Platforms on which all the processors are identical, in the sense that they have the same speed.

Uniform parallel machines By contrast, each processor in a uniform parallel machine is characterized by its own speed, a job that is executed on processor π_i of speed s_i for t time units completes $s_i \times t$ units of execution.

Unrelated parallel machines In unrelated parallel machines, there is an execution rate $s_{i,j}$ associated with each job-processor pair, a job J_i that is executed on processor π_j for t time units completes $s_{i,j} \times t$ units of execution. This kind of architecture is a model for dedicated processors (e.g., if $s_{i,j} = 0$ then π_j cannot execute job J_i).

Related research. The problem of scheduling periodic task systems on multiprocessors was originally studied in [14]. Recent studies provide a better understanding of this scheduling problem and provide the first solutions, e.g., [3] presents a categorization of real-time multiprocessor scheduling problems and [7] an up-to-date state of the art. To the best of our knowledge, a single work [1] provides *exact* schedulability test for the global scheduling of periodic systems on multiprocessors (except two past conference papers we have already published [4, 5]). Baker and Cirinei present a test for global preemptive priority-based scheduling of sporadic tasks on identical processors. Our work differs for several reasons: (i) we extend the model by considering unrelated platforms, and (ii) we provide a *schedulability interval* and the periodic characterization of the schedule. Such characterization is not straightforward since we know that not all uniprocessor schedulability results or arguments are true for multiprocessor scheduling. For instance, the synchronous case (i.e., considering that all tasks start their execution synchronously) is not the worst case on multiprocessors anymore [12]. Another example is the fact that the first busy period (see [13] for details) does not provide a schedulability interval on multiprocessors (see [12] for such counter-examples). By

schedulability interval we mean a finite interval such that, if no deadline is missed while only considering requests within this interval then no deadline will ever be missed.

Initial results indicate that real-time multiprocessor scheduling problems are not typically solved by applying straightforward extensions of techniques used for solving similar uniprocessor problems. Unfortunately, too often, researchers use uniprocessor arguments to study multiprocessor scheduling problems, which leads to incorrect properties. This fact motivated our rigorous and formal approach; in this paper, we will present our exact schedulability tests (and related properties), and rigorously prove them correct.

Our research. In this paper, we consider preemptive global scheduling and we present exact schedulability tests on multiprocessors for various scheduling policies and for various periodic task models. These latter tests have, for a set of n tasks, a time complexity $O(\text{lcm}\{T_1, T_2, \ldots, T_n\})$, which is unfortunately also the case of most schedulability tests for *simpler uni*processor scheduling problems. Our schedulability tests are based on *periodicity* properties of the schedules and on *predictability* properties of the considered schedulers. While the periodicity properties are proved in this work, we use predictability properties that were proved in a previous work [6].

Organization. This paper is organized as follows. Section 2 introduces the definitions, the model of computation and our assumptions. We prove the periodicity of feasible schedules of periodic systems in Section 3. In Section 4 we combine the periodicity and predictability properties to provide for these various kind of periodic task sets and various schedulers *exact* schedulability tests. Lastly, we conclude in Section 5.

2. Definitions and assumptions

We consider the scheduling of periodic task systems. A system τ is composed of n periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, where each task is characterized by a period T_i, a relative deadline D_i, an execution requirement C_i and an offset O_i. Such a periodic task generates an infinite sequence of jobs, with the k^{th} job arriving at time-instant $O_i + (k-1)T_i$ ($k = 1, 2, \ldots$), having an execution requirement of C_i units, and a deadline

at time-instant $O_i + (k-1)T_i + D_i$. It is important to note that we assume in the first part of this manuscript that each job of the same task (say τ_i) has the same execution requirement (C_i); we will relax this assumption in the second part of this manuscript showing that our analysis is *robust*.

We will distinguish between *implicit deadline* systems where $D_i = T_i, \forall i$; *constrained deadline* systems where $D_i \leq T_i, \forall i$ and *arbitrary deadline* systems where there is no relation between the deadlines and the periods. Note that implicit deadline systems are also constrained deadline ones. The latter are also arbitrary deadline ones.

In some cases, we will consider the more general problem of scheduling a set of jobs, each job J_j is characterized by a release time r_j, an execution requirement e_j and an absolute deadline d_j. The job J_j must execute for e_j time units over the interval $[r_j, d_j)$. A job becomes *active* from its release time to its completion.

A periodic system is said to be *synchronous* if there is a time-instant where all tasks make a new request simultaneously, i.e., $\exists t, k_1, k_2, \ldots k_n$ such that $\forall i : t = O_i + k_i T_i$ (see [9] for details). Without loss of generality, we consider $O_i = 0, \forall i$ for synchronous systems. Otherwise the system is said to be *asynchronous*.

We denote by $\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_i\}$, by $O^i_{\max} \stackrel{\text{def}}{=} \max\{O_1, \ldots, O_i\}$, by $O_{\max} \stackrel{\text{def}}{=} O^n_{\max}$, by $P_0 \stackrel{\text{def}}{=} 0$, $P_i \stackrel{\text{def}}{=} \text{lcm}\{T_1, \ldots, T_i\}$ $(0 < i \leq n)$ and by $P \stackrel{\text{def}}{=} P_n$. In the following, the quantity P is called the task set *hyper-period*.

We consider multiprocessor platforms π composed of m unrelated processors (or one of its particular cases: uniform and identical platforms): $\{\pi_1, \pi_2, \ldots, \pi_m\}$. Execution rates $s_{i,j}$ are associated with each task-processor pair. A task τ_i that is executed on processor π_j for t time units completes $s_{i,j} \times t$ units of execution. For each task τ_i we assume that the associated set of processors $\pi_{n_{i,1}} > \pi_{n_{i,2}} > \cdots > \pi_{n_{i,m}}$ are ordered in a decreasing order of the execution rates relative to the task: $s_{i,n_{i,1}} \geq s_{i,n_{i,2}} \geq \cdots \geq s_{i,n_{i,m}}$. For identical execution rates, the ties are broken arbitrarily, but consistently, such that the set of processors associated with each task is *totally* ordered. Consequently, the *fastest* processor relative to task τ_i is $\pi_{n_{i,1}}$, i.e., the first processor of the ordered set associated with the task. Moreover, for a task τ_i in the following we consider that a processor π_a is *faster* than π_b (relative to its associated set of processors) if $\pi_a > \pi_b$ even if we have $s_{i,a} = s_{i,b}$. For the processor-task pair (π_j, τ_i) if $s_{i,j} \neq 0$ then π_j is said to be an *eligible*

processor for τ_i. Note that these concepts and definitions can be trivially adapted to the scheduling of jobs on unrelated platforms.

In this work we consider a discrete model, i.e., the characteristics of the tasks and the time are integers. Moreover, in the following we will illustrate main concepts and definitions using the following system.

Example 1. Let τ be a system of three periodic tasks

	O_i	C_i	D_i	T_i
τ_1	0	2	6	6
τ_2	5	4	6	6
τ_3	4	5	7	6

Let π be a multiprocessor platform of 2 unrelated processors π_1 and π_2. We have $s_{1,1} = 1, s_{1,2} = 2, s_{2,1} = 2, s_{2,2} = 1, s_{3,1} = 2$ and $s_{3,2} = 1$. Note that according to our definition above and regarding task τ_1 we have that $\pi_2 > \pi_1$.

The notions of the state of the system and the schedule are as follows.

Definition 1 (State of the system $\theta(t)$)**.** For any arbitrary deadline task system $\tau = \{\tau_1, \ldots, \tau_n\}$ we define the *state* $\theta(t)$ of the system τ at time-instant t as $\theta : \mathbb{N} \to (\mathbb{Z} \times \mathbb{N} \times \mathbb{N})^n$ with $\theta(t) \stackrel{\text{def}}{=} (\theta_1(t), \theta_2(t), \ldots, \theta_n(t))$ where

$$\theta_i(t) \stackrel{\text{def}}{=} \begin{cases} (-1, x, 0), & \text{if no job of task } \tau_i \text{ was activated before or at } t. \text{ In this case, } x \text{ time} \\ & \text{units remain until the first activation of } \tau_i. \text{ We have } 0 < x \leq O_i. \\ (y, z, u), & \text{otherwise. In this case, } y \text{ denotes the number of active jobs of } \tau_i \\ & \text{at } t, z \text{ denotes the time elapsed at time-instant } t \text{ since the arrival} \\ & \text{of the oldest active job of } \tau_i, \text{ and } u \text{ denotes the amount that the} \\ & \text{oldest active job has executed for. If there is no active job of } \tau_i \\ & \text{at } t, \text{ then } u \text{ is } 0. \end{cases}$$

Note that at any time-instant t several jobs of the same task might be active and we consider that the oldest job is scheduled first, i.e., the FIFO rule is used to serve the various jobs of a given task.

Definition 2 (Schedule $\sigma(t)$)**.** For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$ we define the *schedule* $\sigma(t)$ of system τ at time-instant t as $\sigma : \mathbb{N} \to \{0, 1, \ldots, n\}^m$ where $\sigma(t) \stackrel{\text{def}}{=} (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ with

$$\sigma_j(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if there is no task scheduled on } \pi_j \\ & \text{at time-instant } t; \qquad \forall 1 \leq j \leq m. \\ i, & \text{if task } \tau_i \text{ is scheduled on } \pi_j \text{ at time-instant } t. \end{cases}$$

Note that Definition 2 can be extended trivially to the scheduling of jobs.

A system τ is said to be *schedulable* on a multiprocessor platform if there is at least one feasible schedule, i.e., a schedule in which all tasks meet their deadlines. If A is an algorithm which schedules τ on a multiprocessor platform to meet its deadlines, then the system τ is said to be A*-schedulable*.

Note that for a feasible schedule from Definition 1 we have $0 \leq y \leq \lceil \frac{D_i}{T_i} \rceil$, $0 \leq z < T_i \cdot \lceil \frac{D_i}{T_i} \rceil$ and $0 \leq u < C_i$.

In this work, we consider that *task parallelism is forbidden*: a task cannot be scheduled at the same time-instant on different processors, i.e. $\nexists j_1 \neq j_2 \in \{1, 2, \ldots, m\}$ and $t \in \mathbb{N}$ such that $\sigma_{j_1}(t) = \sigma_{j_2}(t) \neq 0$.

The scheduling algorithms considered in this paper are *deterministic* and work-conserving with the following definitions:

Definition 3 (Deterministic algorithm)**.** A scheduling algorithm is said to be *deterministic* if it generates a unique schedule for any given set of jobs.

In uniprocessor (or identical multiprocessor) scheduling, a *work-conserving* algorithm is defined as the one that never idles a processor when there is at least one active task. For unrelated multiprocessors we adopt the following definition:

Definition 4 (Work-conserving algorithm)**.** An unrelated multiprocessor scheduling algorithm is said to be *work-conserving* if, at each instant, the algorithm schedules jobs on processors as follows: the highest priority (active) job J_i is scheduled on its fastest (and eligible) processor π_j. The same rule is then applied to the remaining active jobs on the remaining available processors.

Notice we assume in this work that no two jobs/tasks share the same priority. It

follows by Definition 4 that a processor π_j can be idle and a job J_i can be active at the same time if and only if $s_{i,j} = 0$.

Moreover, we will assume that the decision of the scheduling algorithm at time t is not based on the past, nor on the actual time t, but only on the characteristics of active tasks and on the state of the system at time t. More formally, we consider *memoryless* schedulers.

Definition 5 (Memoryless algorithm)**.** A scheduling algorithm is said to be *memoryless* if the scheduling decision it made at time t depends only on the characteristics of active tasks and on the current state of the system, i.e., on $\theta(t)$.

Consequently, for memoryless and deterministic schedulers we have the following property:

$$\forall t_1, t_2 \text{ such that } \theta(t_1) = \theta(t_2) \text{ then } \sigma(t_1) = \sigma(t_2).$$

Note that popular real-time schedulers, e.g., EDF and Deadline Monotonic (DM), are memoryless: the priority of each task is only based on its absolute (EDF) or relative (DM) deadline.

In the following, we will distinguish between two kinds of schedulers:

Definition 6 (Task-level fixed-priority)**.** A scheduling algorithm is a *task-level fixed-priority* algorithm if it assigns the priorities to the tasks beforehand; at run-time each job priority corresponds to its task priority.

Definition 7 (Job-level fixed-priority)**.** A scheduling algorithm is a *job-level fixed-priority* algorithm if and only if it satisfies the condition that: for every pair of jobs J_i and J_j, if J_i has higher priority than J_j at some time-instant, then J_i always has higher priority than J_j.

Popular task-level fixed-priority schedulers include the RM and the DM algorithms; popular job-level fixed-priority schedulers include the EDF algorithm, see [15] for details.

Definition 8 (Job J_i^k of a task τ_i)**.** For any task τ_i, we define J_i^k to be the k^{th} job of task τ_i, which becomes active at time-instant $R_i^k \stackrel{\text{def}}{=} O_i + (k-1)T_i$. For two tasks τ_i and τ_j, by $J_i^k > J_j^\ell$ we mean that the job J_i^k has a higher priority than the job J_j^ℓ.

Definition 9 (Executed time $\epsilon_i^k(t)$ for a job J_i^k)**.** For any task τ_i, we define $\epsilon_i^k(t)$ to be the amount of time already executed for J_i^k in the interval $[R_i^k, t)$.

We now introduce the availability of the processors for any schedule $\sigma(t)$.

Definition 10 (Availability of the processors $a(t)$)**.** For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$ we define the *availability of the processors* $a(t)$ of system τ at time-instant t as the set of available processors $a(t) \stackrel{\text{def}}{=} \{j \mid \sigma_j(t) = 0\} \subseteq \{1, \ldots, m\}$.

In Example 1 for a preemptive task-level fixed-priority algorithm with τ_1 the highest priority task and τ_3 the lowest priority task, we obtain the schedule provided in Figure 1. In this figure the execution on the first processor is indicated by black boxes. The releases of jobs are indicated by $\uparrow$ and the deadlines by $\downarrow$.

For instance we have $\theta_2(2) = (-1, 3, 0)$, $\theta_1(3) = (3, 0, 0)$ and $\theta_3(6) = (2, 1, 3)$. Moreover $\sigma(6) = (2, 1)$ and $\sigma(7) = (3, 0)$. Concerning task τ_1, its job J_1^2 becomes active at $R_1^2 = 6$ and this job has a higher priority than the job J_2^1 of task τ_2. For this schedule $a(1) = \{1\}$ and $a(6) = \emptyset$.

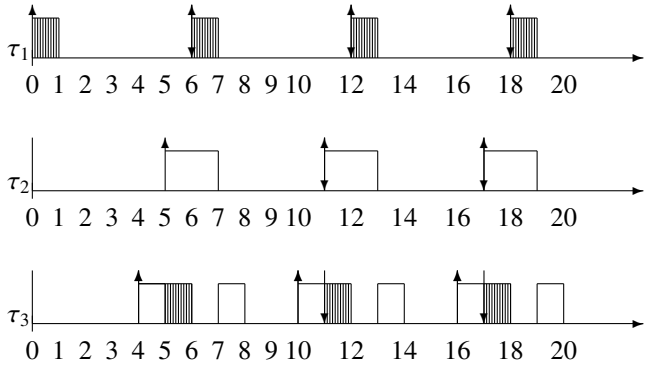

Figure 1: Schedule (obtained using a task-level fixed-priority scheduling) for the task set of Example 1

3. Periodicity of feasible schedules

It is important to keep in mind that we assume in this section that each job of the same task (say, τ_i) has the same execution requirement (C_i). We will relax this assumption in Section 4.

This section contains four parts. In each part of this section we give results concerning the periodicity of feasible schedules. By periodicity (assuming that the period is γ) of a schedule σ, we mean there is a time-instant t_0 and an interval length γ such that $\sigma(t) = \sigma(t + \gamma), \forall t \geq t_0$. For instance, in Figure 1 the schedule is periodic from $t = 5$ with a period equal to 6.

The first part of this section provides periodicity results for a general scheduling algorithm class: deterministic, memoryless and work-conserving schedulers. The second part of this section provides periodicity results for synchronous periodic task systems. The third and the fourth parts of this section present periodicity results for task-level fixed-priority scheduling algorithms for constrained and arbitrary deadline systems, respectively.

3.1. Periodicity of deterministic, memoryless and work-conserving scheduling algorithms

We show that feasible schedules of periodic task systems obtained using deterministic, memoryless and work-conserving algorithms are periodic starting from some point. Moreover, we prove that the schedule repeats with a period equal to P for a subclass of such schedulers. Based on that property, we provide two interesting corollaries for preemptive task-level fixed-priority algorithms (Corollary 4) and for preemptive deterministic EDF [2] (Corollary 5).

We first present two basic results in order to prove Theorem 3.

Lemma 1. For any deterministic and memoryless algorithm A, if an asynchronous arbitrary deadline system τ is A-schedulable on m unrelated processor platform π, then

[2] i.e., the method chosen to break deadline ties must be deterministic.

the feasible schedule of τ on π obtained using A is periodic with a period *divisible* by P.

Proof. First note that from any time instant $t_0 \geq O_{\max}$ all tasks are released, and the configuration $\theta_i(t_0)$ of each task is a triple of finite integers (α, β, γ) with $\alpha \in \{0, 1, \ldots, \lceil \frac{D_i}{T_i} \rceil\}$, $0 \leq \beta < \max_{1 \leq i \leq n}(T_i \times \lceil \frac{D_i}{T_i} \rceil)$ and $0 \leq \gamma < \max_{1 \leq i \leq n} C_i$. Therefore, there is a finite number of different system states, hence we can find two distinct instants t_1 and t_2 ($t_2 > t_1 \geq t_0$) with the same state of the system ($\theta(t_1) = \theta(t_2)$). The schedule repeats from that instant with a period dividing $t_2 - t_1$, since the scheduler is deterministic and memoryless.

Note that, since the tasks are periodic, the arrival pattern of jobs repeats with a period equal to P from $O_{\max}$.

We now prove by contradiction that $t_2 - t_1$ is necessarily a multiple of P. We suppose that $\exists k_1 < k_2 \in \mathbb{N}$ such that $t_i = O_{\max} + k_i P + \Delta_i, \forall i \in \{1, 2\}$ with $\Delta_1 \neq \Delta_2$, $\Delta_1, \Delta_2 \in [0, P)$ and $\theta(t_1) = \theta(t_2)$. This implies that there are tasks for which the time elapsed since the last activation at t_1 and the time elapsed since the last activation at t_2 are not equal. But this is in contradiction with the fact that $\theta(t_1) = \theta(t_2)$. Consequently, Δ_1 must be equal to Δ_2 and, thus, we have $t_2 - t_1 = (k_2 - k_1)P$. □

For a sub-class of schedulers, we will show that the period of the schedule is P, but first a definition (inspired from [11]):

Definition 11 (Request-dependent scheduler)**.** A scheduler is said to be *request-dependent* when $\forall i, j, k, \ell : J_i^{k+h_i} > J_j^{\ell+h_j}$ if and only if $J_i^k > J_j^\ell$, where $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.

Note that Definition 11 (request-dependent) is stronger than Definition 7 (job-level fixed-priority) ; informally speaking Definition 11 requires that the same total order is used *each* hyper-period between "corresponding" jobs (J_i^k "corresponds" to $J_i^{k+h_i}$). For instance in Example 1 for the schedule of Figure 1 we have $J_1^1 > J_2^1$ and $J_1^3 > J_2^3$ because the hyper-period is 6.

The next lemma extends results given for arbitrary deadline task systems in the *uniprocessor* case (see [8], p. 55 for details).

Lemma 2. For any preemptive, job-level fixed-priority and request-dependent algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O_{max}$ and k such that $R_i^k \leq t \leq R_i^k + D_i$, if there is no deadline missed up to time $t + P$, then $\epsilon_i^k(t) \geq \epsilon_i^{k+h_i}(t + P)$ with $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.

Proof. Note first that the function $\epsilon_i^k(\cdot)$ is a non-decreasing discrete[3] step function with $0 \leq \epsilon_i^k(t) \leq C_i, \forall t$ and $\epsilon_i^k(R_i^k) = 0 = \epsilon_i^{k+h_i}(R_i^{k+h_i}), \forall k$.

Note, also that, since the tasks are periodic, the arrival pattern of jobs repeats with a period equal to P from O_{max}.

The proof is made by contradiction. Thus, we assume that a *first* time-instant t exists such that there are j and k with $R_j^k \leq t \leq R_j^k + D_j$ and $\epsilon_j^k(t) < \epsilon_j^{k+h_j}(t + P)$. This assumption implies that:

1. either there is a time-instant t' with $R_j^k \leq t' < t$ such that $J_j^{k+h_j}$ is scheduled at $t' + P$ while J_j^k is not scheduled at t'. We obtain that there is at least one job $J_\ell^{k_\ell+h_\ell}$ of a task τ_ℓ with $\ell \in \{1, 2, \ldots, n\}$ that is not scheduled at $t' + P$, while $J_\ell^{k_\ell}$ is scheduled at t' ($h_\ell \stackrel{\text{def}}{=} \frac{P}{T_\ell}$). This implies (since the tasks are scheduled according to a request-dependent algorithm, the arrival pattern of jobs repeats with a period equal to P from O_{max} and there is no deadline missed) that $J_\ell^{k_\ell}$ has not finished its execution before t', but $J_\ell^{k_\ell+h_\ell}$ has finished its execution before $t' + P$. Therefore, we have $\epsilon_\ell^{k_\ell+h_\ell}(t' + P) = C_\ell$, which implies that $\epsilon_\ell^{k_\ell}(t') < \epsilon_\ell^{k_\ell+h_\ell}(t' + P)$. This last relation is in contradiction with the fact that t is the first such time instant.
2. or there is at least one time-instant $t'' + P < t + P$ when $J_j^{k+h_j}$ is scheduled on a faster processor than the processor on which J_j^k is scheduled at t''. This implies (since the tasks are scheduled according to a request-dependent algorithm) that there is, at least, one job $J_\ell^{k_\ell+h_\ell}$ of a task τ_ℓ with $\ell \in \{1, 2, \ldots, n\}$ that is not scheduled at $t'' + P$, while $J_\ell^{k_\ell}$ is scheduled at t'' ($h_\ell \stackrel{\text{def}}{=} \frac{P}{T_\ell}$). As proved in the first case, this situation leads us to a contradiction with our assumption.

□

[3]We assume that the time unit chosen is small enough such that all execution requirements/deadlines are natural numbers.

For the task set of Example 1 and the schedule of Figure 1, we have from Lemma 2 that for $t = 6$ and $k = 1$, $\epsilon_3^1(6) = 3 = \epsilon_3^2(12)$ and no deadline missed up to 12.

Theorem 3. For any preemptive job-level fixed-priority and request-dependent algorithm A and any A-schedulable asynchronous arbitrary deadline system τ on m unrelated processors the schedule is periodic with a period equal to P.

Proof. By Lemma 1 we have $\exists t_i = O_{\max} + k_i P + d, \forall i \in \{1, 2\}$ with $0 \leq d < P$ such that $\theta(t_1) = \theta(t_2)$. We know also that the arrivals of task jobs repeat with a period equal to P from $O_{\max}$. Therefore, for all time-instants $t_1 + kP$, $\forall k < k_2 - k_1$ (i.e. $t_1 + kP < t_2$), the time elapsed since the last activation at $t_1 + kP$ is the same for all tasks. Moreover since $\theta(t_1) = \theta(t_2)$ we have $\epsilon_i^{\ell_i}(t_1) = \epsilon_i^{\ell_i + \frac{(k_2 - k_1)P}{T_i}}(t_2)$ with $\ell_i \stackrel{\text{def}}{=} \lceil \frac{t_1 - O_i}{T_i} \rceil$, $\forall i$. But by Lemma 2 we also have $\epsilon_i^{\ell_i}(t_1) \geq \epsilon_i^{\ell_i + \frac{P}{T_i}}(t_1 + P) \geq \cdots \geq \epsilon_i^{\ell_i + \frac{(k_2 - k_1)P}{T_i}}(t_2), \forall i$. Consequently we obtain $\theta(t_1) \geq \theta(t_1 + P) \geq \cdots \geq \theta(t_2)$ and $\theta(t_1) = \theta(t_2)$ which [4] implies that $\theta(t_1) = \theta(t_1 + P) = \cdots = \theta(t_2)$. □

Corollary 4. For any preemptive task-level fixed-priority algorithm A, if an asynchronous arbitrary deadline system τ is A-schedulable on m unrelated processors, then the schedule is periodic with a period equal to P.

Proof. The result is a direct consequence of Theorem 3, since task-level fixed-priority algorithms are job-level fixed-priority and request-dependent schedulers. □

Corollary 5. A feasible schedule obtained using deterministic request-dependent global EDF on m unrelated processors of an asynchronous arbitrary deadline system τ is periodic with a period equal to P.

Proof. The result is a direct consequence of Theorem 3, since EDF is a job-level fixed-priority scheduler. □

[4] where $\theta(t) \geq \theta(t')$ means that, for each task, the system state at time t and t' are identical regarding n_1 and t_2, moreover if $n_1 > 0$ we must have that the time elapsed for the oldest active request of the task at time t is not shorter that the time elapsed for the oldest active request of the task at time t'.

3.2. The particular case of synchronous arbitrary deadline periodic systems

In this section, we deal with the periodicity of feasible schedules of *synchronous* arbitrary deadline periodic systems. Using the results obtained for deterministic, memoryless and work-conserving algorithms, we study synchronous arbitrary deadline periodic systems and the periodicity of feasible schedules of these systems under preemptive task-level fixed-priority scheduling algorithms.

In the following, and without loss of generality, we consider the tasks ordered in decreasing order of their priorities $\tau_1 > \tau_2 > \cdots > \tau_n$.

Lemma 6. For any preemptive task-level fixed-priority algorithm A and for any synchronous arbitrary deadline system τ on m unrelated processors, if no deadline is missed in the time interval $[0, P)$ and if $\theta(0) = \theta(P)$, then the schedule of τ is periodic with a period P that begins at time-instant 0.

Proof. Since at time-instants 0 and P the system is in the same state, i.e. $\theta(0) = \theta(P)$, then at time-instants 0 and P a preemptive task-level fixed-priority algorithm will make the same scheduling decision and the scheduled repeats from 0 with a period equal to P. □

Theorem 7. For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ on m unrelated processors, if all deadlines are met in $[0, P)$ and $\theta(0) \neq \theta(P)$, then τ is not A-schedulable.

Proof. In the following, we denote by $\sigma^{(i)}$ the schedule of the task subset $\tau^{(i)}$.

Since $\theta(0) \neq \theta(P)$, there is at least one task with more than one active job at P. We define $\ell \in \{1, 2, \ldots, n\}$ to be the smallest task index such that τ_ℓ has at least two active jobs at P_ℓ, i.e., ℓ is the smallest task index such that $\theta(0) \neq \theta(P_\ell)$. In order to prove the property we will prove that τ_ℓ will miss a deadline.

By definition of ℓ we have $\theta(0) = \theta(P_{\ell-1})$ (at least for the schedule $\sigma^{(\ell-1)}$) and by Lemma 6 we have that the time instants, such that at least one processor is available, are periodic with a period $P_{\ell-1}$, i.e., the schedule $\sigma^{(\ell-1)}$ obtained by considering only the task subset $\tau^{(\ell-1)}$ is periodic with a period $P_{\ell-1}$. Moreover, since P_ℓ is a multiple of $P_{\ell-1}$, we know that the schedule $\sigma^{(\ell-1)}$ is periodic with a period P_ℓ. Therefore, in

each time interval $[k \cdot P_\ell, (k+1)P_\ell)$ with $k \geq 0$ after scheduling $\tau_1, \tau_2, \ldots, \tau_{\ell-1}$ there is the same number t_ℓ of time instants such that at least one processor is available and where τ_ℓ could be scheduled on its fastest processor among the available processors. We denote by $x_j, \forall j \in \{1, 2, \ldots, m\}$ the number (on t_ℓ) of time instants when processor π_j is available and it is the fastest processor for τ_ℓ. Therefore, we have $(\sum_{j=1}^{m} x_j) = t_\ell$ and τ_ℓ completes $\sum_{j=1}^{m} x_j s_{\ell,j}$ units of execution in the interval $[k \cdot P_\ell, (k+1)P_\ell)$.

At time-instant P_ℓ, since task parallelism is forbidden, there are $\frac{P_\ell}{T_\ell} C_\ell - \sum_{j=1}^{m} x_j s_{\ell,j} > 0$ remaining units for execution of τ_ℓ and, consequently, at each time-instant $k \cdot P_\ell$ there will be $k \cdot (\frac{P_\ell}{T_\ell} C_\ell - \sum_{j=1}^{m} x_j s_{\ell,j})$ remaining units for execution of τ_ℓ. Consequently, we can find $k_\ell = \left\lceil \frac{D_\ell}{P_\ell \cdot C_\ell / T_\ell - \sum_{j=1}^{m} x_j s_{\ell,j}} \right\rceil$ such that the job activated at $k_\ell P_\ell$ will miss its deadline since it cannot be scheduled before older jobs of τ_ℓ and there are $k_\ell(\frac{P_\ell}{T_\ell} \cdot C_\ell - \sum_{j=1}^{m} x_j s_{\ell,j}) \geq D_\ell$ remaining units for execution of τ_ℓ at $k_\ell P_\ell$.

Since we consider task-level fixed-priority scheduling, then the tasks τ_i with $i > \ell$ will not interfere with the higher priority tasks already scheduled, particularly with τ_ℓ that misses its deadline, and consequently the system is not A-schedulable. □

Corollary 8. For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ on m unrelated processors, if τ is A-schedulable, then the schedule of τ obtained using A is periodic with a period P that begins at time-instant 0.

Proof. Since τ is A-schedulable, we know by Theorem 7 that $\theta(0) = \theta(P)$. Moreover, a deterministic and memoryless scheduling algorithm will make the same scheduling decision at those instants. Consequently, the schedule repeats from its origin with a period of P. □

3.3. Task-level fixed-priority scheduling of asynchronous constrained deadline systems

In this section we give another important result: any feasible schedule on m unrelated processors of asynchronous constrained deadline systems, obtained using preemptive task-level fixed-priority algorithms, is periodic from some point (Theorem 9) and we characterize that point.

Without loss of generality, we consider the tasks ordered in a decreasing order of their priorities $\tau_1 > \tau_2 > \cdots > \tau_n$.

Theorem 9. For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous constrained deadline system τ on m unrelated processors, the schedule is periodic with a period P_n from time-instant S_n where S_i is defined inductively as follows:

- $S_1 \stackrel{\text{def}}{=} O_1$;
- $S_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{S_{i-1}-O_i}{T_i} \rceil T_i\}, \forall i \in \{2, 3, \ldots, n\}$.

Proof. The proof is made by induction by n (the number of tasks). We denote by $\sigma^{(i)}$ the schedule obtained by considering only the task subset $\tau^{(i)}$, the first higher priority i tasks $\{\tau_1, \ldots, \tau_i\}$, and by $a^{(i)}$, the corresponding availability of the processors. Our inductive hypothesis is the following: the schedule $\sigma^{(k)}$ is periodic from S_k with a period P_k for all $1 \leq k \leq i$.

The property is true in the base case: $\sigma^{(1)}$ is periodic from $S_1 = O_1$ with period P_1, for $\tau^{(1)} = \{\tau_1\}$: since we consider constrained deadline systems, at time-instant $P_1 = T_1$, the previous request of τ_1 has finished its execution and the schedule repeats.

We will now prove that any feasible schedules of $\tau^{(i+1)}$ are periodic with period P_{i+1} from S_{i+1}.

Since $\sigma^{(i)}$ is periodic with a period P_i from S_i, the following equation is verified:

$$\sigma^{(i)}(t) = \sigma^{(i)}(t + P_i), \forall t \geq S_i. \tag{1}$$

We denote by $S_{i+1} \stackrel{\text{def}}{=} \max\{O_{i+1}, O_{i+1} + \lceil \frac{S_i - O_{i+1}}{T_{i+1}} \rceil T_{i+1}\}$ the first request of τ_{i+1} not before S_i.

Since the tasks in $\tau^{(i)}$ have higher priority than τ_{i+1}, the scheduling of τ_{i+1} will not interfere with higher priority tasks which are already scheduled. Therefore, we may build $\sigma^{(i+1)}$ from $\sigma^{(i)}$ such that the tasks $\tau_1, \tau_2, \ldots, \tau_i$ are scheduled at the same instants and on the same processors as they were in $\sigma^{(i)}$. We now apply the induction step: for all $t \geq S_i$ in $\sigma^{(i)}$ we have $a^{(i)}(t) = a^{(i)}(t + P_i)$, thus the availability of the processors

repeats. Task τ_{i+1} *may* be executed only at these instants. Note that at the instants t and $t + P_i$ the available processors (if any) are the same.

The instants t with $S_{i+1} \leq t < S_{i+1} + P_{i+1}$, where τ_{i+1} may be executed in $\sigma^{(i+1)}$, are periodic with period $P_{i+1} = \text{lcm}\{P_i, T_{i+1}\}$. Moreover, since the system is schedulable and we consider constrained deadlines, the only active request of τ_{i+1} at S_{i+1} (respectively at $S_{i+1} + P_{i+1}$) is the one activated at S_{i+1} (respectively at $S_{i+1} + P_{i+1}$). Consequently, the instants at which the task-level fixed-priority algorithm A schedules τ_{i+1} are periodic with period P_{i+1}. Therefore, the schedule $\sigma^{(i+1)}$ repeats from S_{i+1} with a period equal to P_{i+1} and the property is true for all $1 \leq k \leq n$, in particular for $k = n$: $\sigma^{(n)}$ is periodic with a period equal to P from S_n and the property follows. □

Example 2. Let τ be a system of three periodic tasks

	O_i	C_i	D_i	T_i
τ_1	0	2	6	6
τ_2	5	4	6	6
τ_3	4	5	6	6

Let π be a multiprocessor platform of 2 unrelated processors π_1 and π_2. We have $s_{1,1} = 1, s_{1,2} = 2, s_{2,1} = 2, s_{2,2} = 1, s_{3,1} = 2$ and $s_{3,2} = 1$.

Using a task-level fixed-priority scheduling algorithm with τ_1 the highest priority, we obtain the schedule provided in Figure 2. For this schedule and the notations of Theorem 9, we have $S_1 = 0$, $S_2 = 5$ and $S_3 = 10$. In Figure 2 the execution on the first processor is indicated by black boxes.

3.4. Task-level fixed-priority scheduling of asynchronous arbitrary deadline systems

In this section we present another important result: any feasible schedule on m unrelated processors of asynchronous arbitrary deadline systems, obtained using preemptive task-level fixed-priority algorithms, is periodic from some point (Theorem 13).

Corollary 10. For any preemptive task-level fixed-priority algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O_i$ and k such that $R_i^k \leq t \leq R_i^k + D_i$, if there is no deadline missed up to time $t + P$, then $\epsilon_i^k(t) \geq \epsilon_i^{k+h_i}(t + P)$ with $h_i \stackrel{\text{def}}{=} \frac{P}{T_i}$.

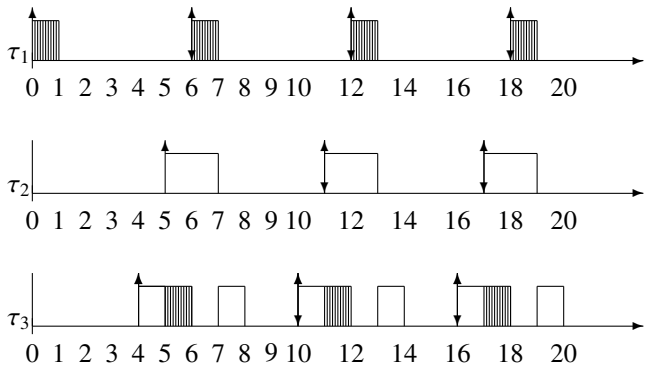

Figure 2: Schedule (obtained using a task-level fixed-priority scheduling) for the task set of Example 2

Proof. This result is a direct consequence of Lemma 2 since preemptive task-level fixed-priority algorithms are job-level fixed-priority and request-dependent schedulers. □

In the following corollary we use the notation $\theta_i(t) = (\alpha_i(t), \beta_i(t), \gamma_i(t)), \forall \tau_i$.

Corollary 11. For any preemptive task-level fixed-priority algorithm A and any asynchronous arbitrary deadline system τ on m unrelated processors, we have: for each task τ_i, for any time-instant $t \geq O^i_{\max}$, if there is no deadline missed up to time $t + P$, then either $(\alpha_i(t) < \alpha_i(t + P))$ or $[\alpha_i(t) = \alpha_i(t + P)$ and $\gamma_i(t) \geq \gamma_i(t + P)]$.

Proof. Note that $0 \leq \alpha_i(t) = n_i(t) - m_i(t)$ where $n_i(t)$ is the number of jobs activated before or at t, and $m_i(t)$ is the number of jobs that have completed their execution before or at t. We have $n_i(t+P) = n_i(t) + \frac{P}{T_i}$ and by Corollary 10 we obtain $m_i(t+P) \leq m_i(t) + \frac{P}{T_i}$. Consequently $\alpha_i(t + P) \geq \alpha_i(t)$, and if $\alpha_i(t) = \alpha_i(t + P)$ then $m_i(t + P) = m_i(t) + \frac{P}{T_i}$, and $\gamma_i(t) = \epsilon^{m_i(t)+1}(t) \geq \epsilon_i^{m_i(t)+1+\frac{P}{T_i}}(t + P) = \gamma_i(t + P)$. The last inequality $(\epsilon^{m_i(t)+1}(t) \geq \epsilon_i^{m_i(t)+1+\frac{P}{T_i}}(t + P))$ is a direct consequence of Corollary 10. □

Lemma 12. For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous arbitrary deadline system τ of n tasks on m unrelated processors, let $\sigma^{(i)}$ the schedule obtained by considering only the task subset $\tau^{(i)}$. Moreover, let the set of $\{\widehat{S}_1, \cdots, \widehat{S}_n\}$ be defined inductively as follows:

- $\widehat{S}_1 \stackrel{\text{def}}{=} O_1$
- $\widehat{S}_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{\widehat{S}_{i-1} - O_i}{T_i} \rceil T_i\} + P_i, \qquad (i > 1)$

If $\theta_{i+1}(\widehat{S}_{i+1}) \neq \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$, then $\nexists t \in [\widehat{S}_{i+1}, \widehat{S}_{i+1} + P_{i+1})$ such that at t there is at least one available (and eligible) processor for τ_{i+1} in $\sigma^{(i)}$ and no job of τ_{i+1} is scheduled at t in $\sigma^{(i+1)}$.

Proof. The proof is made by contradiction.
If there is such a time-instant $t' \in [\widehat{S}_{i+1}, \widehat{S}_{i+1} + P_{i+1})$ then $\alpha_{i+1}(t') = 0$. Thus, by Corollary 11 ($\tau^{(i+1)}$ being schedulable and $t' - P_{i+1} \geq O^i_{\max}$ by construction) we have $\alpha_{i+1}(t' - P_{i+1}) = 0$ (because $\alpha_{i+1}(t' - P_{i+1})$ cannot be smaller than 0) and $\gamma_{i+1}(t') = \gamma_{i+1}(t' - P_{i+1})$. This implies that $\theta_{i+1}(t') = \theta_{i+1}(t' - P_{i+1})$ because $\beta_{i+1}(t')$ is obviously equal to $\beta_{i+1}(t' - P_{i+1})$. Note that $t' - P_{i+1} \geq \widehat{S}_i$ since $t' - P_{i+1} \geq \widehat{S}_{i+1} - P_{i+1} \geq \widehat{S}_i$. From the fact that $t' - P_{i+1} \geq \widehat{S}_i$, $\theta_{i+1}(t') = \theta_{i+1}(t' - P_{i+1})$, we obtain that $\theta_{i+1}(\widehat{S}_{i+1}) = \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$, which is a contradiction with the hypothesis of the lemma. □

Theorem 13. For any preemptive task-level fixed-priority algorithm A and any A-schedulable asynchronous arbitrary deadline system τ on m unrelated processors, the schedule is periodic with a period P_n from time-instant $\widehat{S}_n$ where $\widehat{S}_n$ are defined inductively as follows:

- $\widehat{S}_1 \stackrel{\text{def}}{=} O_1$
- $\widehat{S}_i \stackrel{\text{def}}{=} \max\{O_i, O_i + \lceil \frac{\widehat{S}_{i-1} - O_i}{T_i} \rceil T_i\} + P_i, \qquad (i > 1)$

Proof. The proof is made by induction with n (the number of tasks). We denote by $\sigma^{(i)}$ the schedule obtained by considering only the task subset $\tau^{(i)}$, the first higher priority i tasks $\{\tau_1, \ldots, \tau_i\}$, and by $a^{(i)}$, the corresponding availability of the processors. Our inductive hypothesis is the following: the schedule $\sigma^{(k)}$ is periodic from $\widehat{S}_k$ with a period P_k, for all $1 \leq k \leq i$.

The property is true in the base case: $\sigma^{(1)}$ is periodic from $\widehat{S}_1 = O_1$ with period $P_1 = T_1$, for $\tau^{(1)} = \{\tau_1\}$: since we consider schedulable systems, at time-instant $P_1 + O_1 = T_1 + O_1$ the previous job of τ_1 has finished its execution ($C_1 \leq T_1$) and the schedule repeats.

We will now prove that any feasible schedule of $\tau^{(i+1)}$ is periodic with period P_{i+1} from $\widehat{S}_{i+1}$.

Since $\sigma^{(i)}$ is periodic with a period P_i from $\widehat{S}_i$ the following equation is verified:

$$\sigma^{(i)}(t) = \sigma^{(i)}(t + P_i), \forall t \geq \widehat{S}_i. \tag{2}$$

We denote by $\widehat{S}_{i+1} \stackrel{\text{def}}{=} \max\{O_{i+1}, O_{i+1}+\lceil \frac{\widehat{S}_i - O_{i+1}}{T_{i+1}} \rceil T_{i+1}\} + P_{i+1}$ the time-instant obtained by adding P_{i+1} to the time-instant which corresponds to the first activation of τ_{i+1} after $\widehat{S}_i$.

Since the tasks in $\tau^{(i)}$ have higher priority than τ_{i+1}, the scheduling of τ_{i+1} will not interfere with higher priority tasks which are already scheduled. Therefore, we may build $\sigma^{(i+1)}$ from $\sigma^{(i)}$ such that the tasks $\tau_1, \tau_2, \ldots, \tau_i$ are scheduled at the same instants and on the same processors as they were in $\sigma^{(i)}$. We apply now the induction step: for all $t \geq \widehat{S}_i$ in $\sigma^{(i)}$ we have $a^{(i)}(t) = a^{(i)}(t + P_i)$, i.e., the availability of the processors repeats. Note that at the instants t and $t + P_i$ the available processors for τ_{i+1} (if any) are the same. Hence only at these instants task τ_{i+1} *may* be executed in the time interval $[\widehat{S}_{i+1}, \widehat{S}_{i+1} + P_{i+1})$.

The instants t such that $\widehat{S}_{i+1} \leq t < \widehat{S}_{i+1} + P_{i+1}$, where τ_{i+1} may be executed in $\sigma^{(i+1)}$, are periodic with period P_{i+1}, since P_{i+1} is a multiple of P_i and $\widehat{S}_{i+1} \geq \widehat{S}_i$.

We prove now that the system is in the same state at time-instants $\widehat{S}_{i+1}$ and $\widehat{S}_{i+1} + P_{i+1}$. The proof is made by contradiction. Therefore, we suppose that $\theta(\widehat{S}_{i+1}) \neq \theta(\widehat{S}_{i+1} + P_{i+1})$. Since from the inductive hypothesis we have $\theta_k(\widehat{S}_{i+1}) = \theta_k(\widehat{S}_{i+1} + P_{i+1})$ for $1 \leq k \leq i$, then our assumption implies that only $\theta_{i+1}(\widehat{S}_{i+1}) \neq \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$.

Now we prove that our assumption $\theta_{i+1}(\widehat{S}_{i+1}) \neq \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$ is wrong.

From $\theta_{i+1}(\widehat{S}_{i+1}) \neq \theta_{i+1}(\widehat{S}_{i+1} + P_{i+1})$ we have by Corollary 11 that either there are fewer active jobs at $\widehat{S}_{i+1}$ than at $\widehat{S}_{i+1} + P_{i+1}$, or if there is the same number of active jobs of $\widehat{S}_{i+1}$ then the oldest active job at $\widehat{S}_{i+1}$ was executed for more time units than the oldest active at $\widehat{S}_{i+1} + P_{i+1}$. Therefore and because of Lemma 12, there are not sufficient time instants when at least one processor is available to schedule all the jobs activated for τ_{i+1} in the time interval $[\widehat{S}_{i+1}, \widehat{S}_{i+1} + P_{i+1})$. We obtain that the system is not schedulable, which is in contradiction with our assumption of τ being schedulable.

Consequently, $\theta(\widehat{S}_{i+1}) = \theta(\widehat{S}_{i+1} + P_{i+1})$. Moreover, by definition of $\widehat{S}_{i+1}$ (which corresponds to an activation of τ_{i+1}), the task activations repeat from $\widehat{S}_{i+1}$ which proves the property. □

For instance for the task set of Example 1 and the schedule of Figure 1, we have $\widehat{S}_1 = 0$, $\widehat{S}_2 = 11$ and $\widehat{S}_{i+1} = 22$.

4. Exact schedulability tests

In the previous sections, we assumed that the execution requirement of each task is constant while the designer actually only knows an upper bound on the actual execution requirement, i.e., the worst case execution time (WCET). Consequently, we need tests that are *robust* relatively to the variation of the execution times up to a worst case value. More precisely, we need *predictable* schedulers on the considered platforms.

In Section 4.1 we remember the predictability results proposed in [6] that are used in this section to provide exact schedulability tests for various kind schedulers and platforms.

First of all, we introduce and formalize the notion of the *schedulability interval* necessary to provide the exact schedulability tests:

Definition 12 (Schedulability interval)**.** For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any set of m processors $\{\pi_1, \ldots, \pi_m\}$, the *schedulability interval* is a finite interval such that if no deadline is missed while considering only requests within this interval then no deadline will ever be missed.

4.1. Basic results

In this section, we consider the scheduling of sets of job $J \stackrel{\text{def}}{=} J_1, J_2, J_3 \ldots$, (finite or infinite set of jobs) and without loss of generality we consider jobs in a decreasing order of priorities ($J_1 > J_2 > J_3 > \cdots$). We suppose that the execution times of each job J_i can be any value in the interval $[e_i^-, e_i^+]$ and we denote with J_i^+ the job defined as $J_i^+ \stackrel{\text{def}}{=} (r_i, e_i^+, d_i)$. The associated execution rates of J_i^+ are equal to $s_{i,j}, \forall j$. Similarly, J_i^- is the job defined from J_i as follows: $J_i^- = (r_i, e_i^-, d_i)$. Similarly, the associated execution rates of J_i^- are equal to $s_{i,j}, \forall j$. We denote by $J^{(i)}$ the set of the

first i higher priority jobs. We denote also by $J_-^{(i)}$ the set $\{J_1^-, \ldots, J_i^-\}$ and by $J_+^{(i)}$ the set $\{J_1^+, \ldots, J_i^+\}$. Note that the schedule of an ordered set of jobs using a work-conserving and job-level fixed-priority algorithm is unique. Let $S(J)$ be the time-instant at which the lowest priority job of J begins its execution in the schedule. Similarly, let $F(J)$ be the time-instant at which the lowest priority job of J completes its execution in the schedule.

Definition 13 (Predictable algorithms [6])**.** A scheduling algorithm is said to be *predictable* if $S(J_-^{(i)}) \leq S(J^{(i)}) \leq S(J_+^{(i)})$ and $F(J_-^{(i)}) \leq F(J^{(i)}) \leq F(J_+^{(i)})$, for all $1 \leq i \leq \ell$ and for all schedulable $J_+^{(i)}$ sets of jobs.

Theorem 14 ([6])**.** Job-level fixed-priority algorithms are predictable on unrelated platforms.

4.2. Asynchronous constrained deadline systems and task-level fixed-priority schedulers

Now we have the material to define an exact schedulability test for asynchronous constrained deadline periodic systems.

Corollary 15. For any preemptive task-level fixed-priority algorithm A and for any asynchronous constrained deadline system τ on unrelated multiprocessors, τ is A-schedulable if all deadlines are met in $[0, S_n + P)$ and $\theta(S_n) = \theta(S_n + P)$, where S_i is defined inductively in Theorem 9. Moreover, for every task τ_i only the deadlines in the interval $[S_i, S_i + \text{lcm}\{T_j \mid j \leq i\})$ have to be checked.

Proof. Corollary 15 is a direct consequence of Theorem 9 and Theorem 14, since task-level fixed-priority algorithms are job-level fixed-priority schedulers. □

The schedulability test given by Corollary 15 may be improved as it was done in the uniprocessor case [10]. The uniprocessor proof is also true for multiprocessor platforms since it does not depend on the number of processors, nor on the kind of platforms but on the availability of the processors.

Theorem 16 ([10])**.** Let X_i be inductively defined by $X_n = S_n, X_i = O_i + \lfloor \frac{X_{i+1}-O_i}{T_i} \rfloor T_i$ ($i \in \{n-1, n-2, \ldots, 1\}$; thus τ is A-schedulable if and only if all deadlines are met in $[X_1, S_n + P)$ and if $\theta(S_n) = \theta(S_n + P)$.

4.3. Asynchronous arbitrary deadline systems and task-level fixed-priority policies

Now we have the material to define an exact schedulability test for asynchronous arbitrary deadline periodic systems.

Corollary 17. For any preemptive task-level fixed-priority algorithm A and for any asynchronous arbitrary deadline system τ on m unrelated processors, τ is A-schedulable if and only if all deadlines are met in $[0, \widehat{S}_n + P)$ and $\theta(\widehat{S}_n) = \theta(\widehat{S}_n + P)$, where $\widehat{S}_i$ are defined inductively in Theorem 13.

Proof. Corollary 17 is a direct consequence of Theorem 13 and Theorem 14, since task-level fixed-priority algorithms are job-level fixed-priority schedulers. □

Note that the length of our (schedulability) interval is proportional to P (the least common multiple of the periods) which is unfortunately also the case of most schedulability intervals for the *simpler uni*processor scheduling problem (and for identical platforms or simpler task models). In practice, the periods are often *harmonics*, which limits the term P.

4.4. EDF scheduling of asynchronous arbitrary deadline systems

We know by Corollary 5 that any deterministic, request-dependent and feasible EDF schedule is periodic with a period equal to P. Unfortunately, to the best of our knowledge we have no upper bound on the time-instant at which the periodic part of the schedule begins. Examples show that $O_{\max} + P$ is not such a time-instant for EDF on multiprocessors (see [2] for instance). Other examples, show that in some cases the periodic part of the schedule begins after a huge time interval (i.e., many hyper-periods).

However, based on Corollary 5 we will define an *exact* schedulability test under EDF on multiprocessors. The idea illustrated by Algorithm 1 is to build the schedule (by means of simulation) and regularly check if the periodic part of the schedule is reached or not.

Algorithm 1: Exact EDF-schedulability test on multiprocessors

Input: task set τ

Output: feasible

begin

Schedule (from 0) to $O_{\max}$;

{The function **Schedule** stops the program and returns false once a deadline is missed}

$s_1 := \theta(O_{\max})$;

Schedule (from $O_{\max}$) to $O_{\max} + P$;

$s_2 := \theta(O_{\max} + P)$;

current-time := $O_{\max} + P$;

while $s_1 \neq s_2$ **do**

$s_1 := s_2$;

Schedule (from current-time) to current-time + P ;

current-time := current-time + P ;

$s_2 := \theta$(current-time) ;

return *true*;

end

Algorithm	$D_i = T_i$	$D_i \leq T_i$	$D_i ? T_i$
deterministic and memoryless			
task-level fixed-priority			
job-level fixed-priority			

Table 1: Table summarizing our results

4.5. The particular case of synchronous arbitrary deadline periodic systems

In this section we present exact schedulability tests in the particular case of synchronous arbitrary deadline periodic systems.

Corollary 18. For any preemptive task-level fixed-priority algorithm A and any synchronous arbitrary deadline system τ, τ is A-schedulable on m unrelated processors if and only if all deadlines are met in the interval $[0, P)$ and $\theta(0) = \theta(P)$.

Proof. The result is a direct consequence of Corollary 8 and Theorem 14, since task-level fixed-priority schedulers are priority-driven. □

5. Conclusion

In this paper we studied the global scheduling of periodic task systems on heterogeneous multiprocessor platforms. We provided exact schedulability tests based on periodicity properties.

For any asynchronous arbitrary deadline periodic task system and any task-level fixed-priority scheduler (e.g., RM) we characterized an upper bound in the schedule where the periodic part begins. Based on that property we provide schedulability intervals (and consequently exact schedulability tests) for those schedulers.

To the best of our knowledge such an interval is unknown for EDF, a *job*-level fixed-priority scheduler. Fortunately, based on a periodicity property we provide an algorithm which determines (by means of simulation) where the periodicity is already started (if schedulable), this algorithm provides an *exact* schedulability test for EDF on heterogeneous multiprocessors.

[1] Baker, T., Cirinei, M., December 2007. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In: The 10th International Conference on Principles of Distributed Systems. No. 4878. Springer Lecture Notes in Computer Science, pp. 62–75.

[2] Braun, C., Cucu, L., 2007. Negative results on idle intervals and periodicity for multiprocessor scheduling under EDF. In: Junior Researcher Workshop on Real-Time Computing. Institut National Polytechnique de Lorraine, France.

[3] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., Baruah, S., 2005. A categorization of real-time multiprocessor scheduling problems and algorithms. Handbook of Scheduling.

[4] Cucu, L., Goossens, J., 2006. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation, 397–405.

[5] Cucu, L., Goossens, J., 2007. Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. Proceedings of the 10th Design, Automation and Test in Europe, 1635–1640.

[6] Cucu-Grosjean, L., Goossens, J., 2010. Predictability of fixed-job priority schedulers on heterogeneous multiprocessor real-time systems. Information Processing Letters 110 (10), 399–402.

[7] Davis, R.I. and Burns, A., 2009. A survey of hard real-time scheduling algorithms and schedulability analysis techniques for multiprocessor systems. Tech. rep., University of York, ftp://ftp.cs.york.ac.uk/reports/2009/YCS/443/YCS-2009-443.pdf.

[8] Goossens, J., 1999. Scheduling of hard real-time periodic systems with various kinds of deadline and offset constraints. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium.

[9] Goossens, J., 2003. Scheduling of offset free systems. Real-Time Systems: The International Journal of Time-Critical Computing 24 (2), 239–258.

[10] Goossens, J., Devillers, R., 1997. The non-optimality of the monotonic assignments for hard real-time offset free systems. Real-Time Systems: The International Journal of Time-Critical Computing 13 (2), 107–126.

[11] Goossens, J., Devillers, R., 1999. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In: Proceedings of the six International Conference on Real-time Computing Systems and Applications. IEEE Computer Society Press, pp. 54–61.

[12] Goossens, J., Funk, S., Baruah, S., 2002. EDF scheduling on multiprocessors: some (perhaps) counterintuitive observations. Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications, 321–330.

[13] Lehoczky, J., 1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: IEEE Real-Time Systems Symposium. pp. 201–213.

[14] Liu, C., 1969. Scheduling algorithms for multiprocessors in a hard real-time environment. JPL Space Programs Summary 37-60(II), 28–31.

[15] Liu, C., Layland, J., 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM 20 (1), 46–61.