Feasibility Intervals for Fixed-Priority Real-Time Scheduling on Uniform Multiprocessors

Liliana Cucu*
Université Libre de Bruxelles, C.P. 212
50 Avenue Franklin D. Roosevelt
1050 Brussels, Belgium
liliana.cucu@ulb.ac.be

Joël Goossens
Université Libre de Bruxelles, C.P. 212
50 Avenue Franklin D. Roosevelt
1050 Brussels, Belgium
joel.goossens@ulb.ac.be

Abstract

In this paper we study the global scheduling of periodic task systems upon uniform multiprocessor platforms. We first show two very general properties which are well-known for uniprocessor platforms and which remain for multiprocessor one: (i) under few and not so restrictive assumptions, we show that any feasible schedules of periodic task system are periodic from some point and (ii) for the specific case of synchronous periodic task systems, we show that the schedule repeats from the origin. We then present our main result: any feasible schedules of asynchronous periodic task sets using a fixed-priority scheduler are periodic from a specific point. Moreover, we characterize that point and we provide a feasibility interval for those systems.

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 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 very predictable manner. These jobs have upper bounds upon their worst-case execution requirements, and associated deadlines. In this work, we consider periodic task systems, each periodic task τ_i generates jobs at each integer multiple of its period T_i and the jobs must be executed for at most C_i time units and completed by its relative deadline D_i. The first job of a task τ_i is released at time O_i (the task offset). If there is a time instant at which jobs of all tasks are released synchronously, the system is said to be *synchronous*; otherwise the system is said to be *asynchronous*.

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 *feasible*. More formal definitions of these notions are given in Section 2.

Uniprocessor real-time systems are well studied since the seminal paper of Liu and Layland [1], the literature presenting scheduling algorithms and feasibility tests for this case is tremendous. 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 between 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., different jobs of an individual task may execute upon different processors) as well as job migration: an individual job that is preempted may resume execution upon a processor different from the one upon which it had been executing prior to preemption.

Scheduling theorists distinguish between at least three kinds of multiprocessor machines:

Identical parallel machines These are multiprocessors in which all the processors are identical, in the sense that they have the same computing power.

Uniform parallel machines By contrast, each processor in a uniform parallel machine is characterized by its own computing capacity, a job that executes on processor p_i of computing capacity 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_{ij} associated with each

*Postdoctoral Researcher FNRS

job-processor pair, a job J_i that executes on processor p_j for t time units completes $s_{ij} \times t$ units of execution.

Related research. The problem of scheduling periodic task systems on identical parallel processors was originally proposed in [2]. Dhall and Liu [3] showed that the *Rate Monotonic* (RM) and *Earliest Deadline First* (EDF) do not provide a feasible schedule for arbitrary low processor utilization. Moreover it has been showed that no on-line[1] scheduling algorithm can be optimal [4]. Nevertheless optimal algorithms based on PFair scheduling [5] were proposed for identical parallel processors. PFair schedules are likely to contain a large number of job preemptions and context-switches. For some applications, this is not an issue; for others, however, the overhead resulting from too many preemptions may prove unacceptable.

A better understanding of hard real-time scheduling on *identical multiprocessors* was provided through [6, 7, 8, 9, 10]. Recent results concern uniform multiprocessors [7, 11, 12, 13, 14]. These papers presented results on feasibility tests or improved algorithms in order to increase processors utilization. It may be noticed that in recent papers, the fact that a feasible schedule repeats or that a feasibility interval exists is mentioned but not proved (as underlined by Baker in [15]). We know that real-time multiprocessor scheduling problems are typically not solved by applying straightforward extensions of techniques used for solving similar uniprocessor problems (see [18] for such examples). For that reason we shall rigorously define feasibility intervals and related properties for uniform multiprocessors in this paper.

In [16], a first step in order to prove that a feasible schedule repeats was given for the case of *two* identical processors. The author gave the earliest instant from which the schedule repeats. The work presented in [16] is actually a generalization of a result obtained for uniprocessor case in [17]. We shall compare the contribution of this work in regard to [16] Section 4.1.

This research. In this paper, we show that any feasible schedules of a periodic task system on uniform parallel processors repeat from some point, extending this way results from uniprocessor case [19] to multiprocessor one. For synchronous periodic task systems on uniform parallel processors we show that the schedule repeats from the origin. Then we give a feasibility interval for the case of global fixed-priority scheduling[2] of these systems. Then, we prove a more precise and useful result, the main contribution of this paper: any feasible schedules of global fixed-priority scheduling of asynchronous periodic task systems on uniform parallel processors are periodic from a specific point (or possibly before). We also characterize that point and we provide a feasibility interval for those systems. Moreover, we provide an additional contribution: we show that priority-driven algorithms are predictable on *uniform* multiprocessors.

Organization. This paper is organized as follows. In Section 2, we introduce our model of computation and we show that priority-driven algorithms are predictable on uniform platforms (Section 2.1). In Section 3, we show that any feasible schedules of periodic task systems are periodic from some point. In Section 3.1, we consider the specific case of synchronous periodic task systems. In Section 4, we present our main result: a feasibility interval for asynchronous periodic task sets using global fixed-priority scheduling and we conclude in Section 5.

[1] the characteristics of jobs need not be known prior to their release times.

[2] the priorities are assigned to the tasks beforehand, at run-time each request inherits of its task priority and remains constant.

2 Definitions and assumptions

We consider the scheduling of periodic task systems. A system τ is composed by n periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, each task is characterized by a period T_i, a relative deadline D_i, a worst-case execution time 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 a worst-case execution requirement of C_i units, and a hard deadline at time-instant $O_i + (k-1)T_i + D_i$.

We shall 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 constraint between the deadline and the period.

In some cases, we shall consider the more general problem of scheduling a set of jobs, each job $J_j = (r_j, e_j, d_j)$ is characterized by an release time r_j, an execution time e_i 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 periodic system is said to be *synchronous* if there is an instant at which jobs of all tasks are released synchronously, i.e. $\exists t, k_1, k_2, \ldots k_n$ such that $\forall i : t = O_i + k_i T_i$ (see [14] for details). Without loss of generality, we assume that $O_i = 0, \forall i$, for synchronous systems. Otherwise the system is said to be *asynchronous*.

A task becomes active from its release time to its completion.

We denote by $\tau^{(i)} \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_i\}$, by $O_{\max} \stackrel{\text{def}}{=} \max\{O_1, O_2, \ldots, O_n\}$, by $P_i \stackrel{\text{def}}{=} \text{lcm}\{T_1, T_2, \ldots, T_i\}$ and $P \stackrel{\text{def}}{=} P_n$.

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

We consider in this paper m uniform processors, and without loss of generality, the set of processors $\{p_1, \ldots, p_m\}$ is considered ordered in the decreasing order of their computing capacity. We consider in this paper a discrete model, i.e., the characteristics of the tasks are integers. Moreover, we assume that the instants at which

the scheduler makes decisions are equidistant.[3]

We now formalize the notions of the state of the system and the schedule.

Definition 1 (State of the system $\theta(t)$) *For any constrained deadline system $\tau = \{\tau_1, \ldots, \tau_n\}$ we define the* state $\theta(t)$ *of the system τ at instant t as $\theta : \mathbb{N} \rightarrow (\{-1, 0, 1\} \times \mathbb{N}^2)^n$ with $\theta(t) \stackrel{\text{def}}{=} (\theta_1(t), \theta_2(t), \ldots, \theta_n(t))$ where*

$$\theta_i(t) \stackrel{\text{def}}{=} \begin{cases} (-1, t_1, 0), & \text{if no job of task } \tau_i \text{ was activated before or at } t \text{ and it remains } t_1 \text{ time units until the first activation of } \tau_i. \text{ (We have } 0 < t_1 \leq O_i\text{);} \\ (0, t_2, 0), & \text{if at least one job of } \tau_i \text{ was already activated before } t\text{, but there is no active job of } \tau_i \text{ at } t. \text{ The time elapsed since its last activation is } t_2. \text{ (We have } 0 < t_2 < T_i\text{);} \\ (1, t_3, t_4), & \text{if there is an active job of } \tau_i \text{ at instant } t \text{ the time elapsed since its last action is } t_3 \text{ and } t_4 \text{ units were already executed. (We have } 0 \leq t_3 < T_i \text{ and } 0 \leq t_4 < C_i\text{.)} \end{cases}$$

Definition 2 (Schedule $\sigma(t)$) *For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any ordered set of m processors $\{p_1, \ldots, p_m\}$ we define the* schedule $\sigma(t)$ *of system τ at instant t as $\sigma : \mathbb{N} \rightarrow \{0, 1, \ldots, n\}^m$ where*
$\sigma(t) \stackrel{\text{def}}{=} (\sigma_1(t), \sigma_2(t), \ldots, \sigma_m(t))$ *with*

$$\sigma_j(t) \stackrel{\text{def}}{=} \begin{cases} 0, & \text{if there is no task scheduled on } p_j \text{ at instant } t\text{;} \\ i, & \text{if } \tau_i \text{ is scheduled on } p_j \text{ at instant } t. \end{cases}$$

$\forall 1 \leq j \leq m$.

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

Notice that Definition 2 can be extended trivially to the scheduling of a set of jobs.

The scheduling algorithms considered in this paper are *deterministic* with the following definition:

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

Moreover, we shall 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.

[3] e.g., all natural integers if C_i/s_j is an natural integer $\forall i, j$.

Definition 4 (Memoryless algorithms) *A scheduling algorithm is said to be* memoryless *if the scheduling decision made by it 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:

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

In uniprocessor (or identical multiprocessor) scheduling, a *work-conserving* algorithm is defined to be the one that never idles a processor while there is at least one active job. For uniform multiprocessors we adopt the following definition.

Definition 5 (Work-conserving algorithms) *An uniform multiprocessor scheduling algorithm is* work-conserving *if:*

- *no processor is idled while there are active jobs;*
- *if at some time instant there are fewer than m active jobs (m denotes the number of uniform processors) then the active jobs are executed upon the fastest processors;*
- *higher priority jobs are executed on faster processors.*

2.1 Preliminary results

In order to dispose of a feasibility interval and since the task parameter C_i is only an *upper* bound of the actual execution time of the task requests, we need to show that work-conserving and priority-driven algorithms are *predictable* on *uniform* platforms. As far as we know the latter property is not considered in the literature and we shall fill the gap. But first we need some additional definitions and properties.

Definition 6 (Priority-driven algorithms) *A scheduling algorithm is a* priority-driven *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.*

It may be noticed that the order defined on the set of jobs is not necessarily total. Indeed, we only need to compare the priorities of simultaneous active jobs.

In this section, we consider the scheduling of sets of ℓ jobs (e.g., the set $J = \{J_1, \ldots, J_\ell\}$), and without loss of generality we consider jobs in decreasing order of priorities ($J_1 > J_2 > \cdots > J_\ell$). 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 by J_i^+ the job defined from job J_i as follows: $J_i^+ \stackrel{\text{def}}{=} (r_i, e_i^+, d_i)$. Similarly, J_i^- is the job defined from J_i as follows: $J_i^- = (r_i, e_i^-, d_i)$. 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 $\{J_1^+, \ldots, J_i^+\}$. Notice that the schedule of an ordered set of jobs using a work-conserving and priority-driven 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 7 (Predictable algorithms) *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 feasible* $J_+^{(i)}$ *sets of jobs.*

In [20] the authors showed that work-conserving priority-driven algorithms are predictable on identical processors. The proof presented is not convincing (the proof is based on the total sum of execution times without taking into account the hypothesis that task parallelism is forbidden), we shall show here a more general result: the case of *uniform* work-conserving and priority-driven algorithms. We introduce the notion of *availability of the processors* for a set of jobs.

Definition 8 (Availability of the processors $A(J,t)$) *For any ordered set of jobs J and any ordered set of m uniform processors $\{p_1, \ldots, p_m\}$, we define the* availability of the processors *$A(J,t)$ of the set of jobs J at instant t as the set of available processors:* $A(J,t) \stackrel{\text{def}}{=} \{j \mid \sigma_j(t) = 0\} \subseteq \{1, \ldots, m\}$*, where σ is the schedule of J.*

Lemma 1 *For a feasible ordered set of jobs J (using the priority-driven and work-conserving schedule) and an ordered set of uniform processors $\{p_1, \ldots, p_m\}$, we have that $A(J_+^{(i)}, t) \subseteq A(J^{(i)}, t)$, for all t and all i. That is, at any time instant the processors available in $\sigma_+^{(i)}$ are also available in $\sigma^{(i)}$. (We consider that the sets of jobs are ordered in the same decreasing order of the priorities, i.e., $J_1 > J_2 > \cdots > J_\ell$ and $J_1^+ > J_2^+ > \cdots > J_\ell^+$.)*

Proof. The proof is made by induction by ℓ (the number of jobs). Our inductive hypothesis is the following: $A(J_+^{(k)}, t) \subseteq A(J^{(k)}, t)$, for all t and $1 \leq k \leq i$.

The property is true in the base case since $A(J_+^{(1)}, t) \subseteq A(J^{(1)}, t)$, for all t. Indeed, $S(J^{(1)}) = S(J_+^{(1)})$ and they are both scheduled on the fastest processor, but $J_+^{(1)}$ will be executed for the same or a larger amount of time than $J^{(1)}$.

We shall show now that $A(J_+^{(i+1)}, t) \subseteq A(J^{(i+1)}, t)$, for all t.

Since the jobs in $J^{(i)}$ have higher priority than J_{i+1}, then the scheduling of J_{i+1} will not interfere with higher priority jobs which are already scheduled. Similarly J_{i+1}^+ will not interfere with higher priority jobs of $J_+^{(i)}$ which are already scheduled. Therefore, we may build the schedule $\sigma^{(i+1)}$ from $\sigma^{(i)}$, such that the jobs $J_1, J_2, \ldots, J_i$, are scheduled at the very same instants and on the very same processors as they were in $\sigma^{(i)}$. Similarly, we may build $\sigma_+^{(i+1)}$ from $\sigma_+^{(i)}$.

Notice that $A(J^{(i+1)}, t)$ will contain the same available processors as $A(J^{(i)}, t)$ for all t except the time instants at which J_{i+1} is scheduled, and similarly $A(J_+^{(i+1)}, t)$ will contain the same available processors as $A(J_+^{(i)}, t)$ for all t except the time instants at which J_{i+1}^+ is scheduled. From the inductive hypothesis we have that $A(J_+^{(i)}, t) \subseteq A(J^{(i)}, t)$, for all t and consequently, the job J_{i+1} can be scheduled at least at the very same instants and on the very same processors than J_{i+1}^+, but the job J_{i+1} may also progress at the very same instant on faster processors or during additional time instants (since we consider work-conserving scheduling). Combined with the fact that $e_i \leq e_i^+$ the property follows. □

Theorem 2 *Work-conserving and priority-driven algorithms are predictable on uniform platforms.*

Proof. For a feasible ordered set J of ℓ jobs and an ordered set of uniform processors $\{p_1, \ldots, p_m\}$, we have to show that $S(J_-^{(i)}) \leq S(J^{(i)}) \leq S(J_+^{(i)})$ and $F(J_-^{(i)}) \leq F(J^{(i)}) \leq F(J_+^{(i)})$, for all $1 \leq i \leq \ell$. (The sets of jobs are ordered in the same decreasing order of the priorities, i.e., $J_1^- > J_2^- > \cdots > J_\ell^-$, $J_1 > J_2 > \cdots > J_\ell$ and $J_1^+ > J_2^+ > \cdots > J_\ell^+$.)

The proof is made by induction by ℓ (the number of jobs). We show the second part of each inequality, i.e. $S(J^{(i)}) \leq S(J_+^{(i)})$ and $F(J^{(i)}) \leq F(J_+^{(i)})$, for all $1 \leq i \leq \ell$. The proof of the first part of inequalities is similar.

Our inductive hypothesis is the following: $S(J^{(k)}) \leq S(J_+^{(k)})$ and $F(J^{(k)}) \leq F(J_+^{(k)})$, for all $1 \leq k \leq i$.

The property is true in the base case since $S(J^{(1)}) = S(J_+^{(1)})$ and $F(J^{(1)}) \leq F(J_+^{(1)})$.

We shall show now that $S(J^{(i+1)}) \leq S(J_+^{(i+1)})$ and $F(J^{(i+1)}) \leq F(J_+^{(i+1)})$.

Since the jobs in $J^{(i)}$ have higher priority than J_{i+1} then the scheduling of J_{i+1} will not interfere with higher priority jobs which are already scheduled. Similarly J_{i+1}^+ will not interfere with higher priority jobs of $J_+^{(i)}$ which are already scheduled. Therefore, we may build $\sigma^{(i+1)}$ from $\sigma^{(i)}$, respectively $\sigma_+^{(i+1)}$ from $\sigma_+^{(i)}$, such that the jobs $J_1, J_2, \ldots, J_i$, respectively the jobs $J_1^+, J_2^+, \ldots, J_i^+$, are scheduled at the very same instants and on the very same processors as they were in $\sigma^{(i)}$, respectively in $\sigma_+^{(i)}$. The job J_{i+1} can be scheduled only when the processors are available in $\sigma^{(i)}$, consequently at those time instants $t_0 \geq r_{i+1}$ for which the availability of processors $A(J^{(i)}, t) \neq \emptyset$. Similarly J_{i+1}^+ may be scheduled at those time instants $t_0^+ \geq r_{i+1}$ for which the availability of processors $A(J_+^{(i)}, t) \neq \emptyset$. From the inductive hypothesis we know that higher priority jobs complete sooner (or at the

same time) consequently $t_0 \leq t_0^+$, and J_{i+1} begins its execution in $\sigma^{(i+1)}$ sooner or at the same instant than J_{i+1}^+ in $\sigma_+^{(i+1)}$, i.e. $S(J^{(i+1)}) \leq S(J_+^{(i+1)})$. It follows from Lemma 1 that from time t_0 the job J_{i+1} can be scheduled at least at the very same instants and on the very same processors than J_{i+1}^+, but the job J_{i+1} may also progress at the very same instants on faster processors or during additional time instants (since we consider work-conserving scheduling). Consequently, $F(J^{(i+1)}) \leq F(J_+^{(i+1)})$. □

3 Periodicity of deterministic and memoryless scheduling algorithms

In this section, we shall show that feasible schedules of periodic task systems obtained using deterministic and memoryless algorithm are periodic from some point (Theorem 3), assuming that the execution times of each task is constant. With the latter assumption, we obtain two interesting applications of Theorem 3: preemptive fixed-priority algorithms (Corollary 4) and deterministic EDF[4] (Corollary 5). But first, we formalize the notion of feasibility interval.

Definition 9 (Feasibility interval) *For any task system $\tau = \{\tau_1, \ldots, \tau_n\}$ and any ordered set of m processors $\{p_1, \ldots, p_m\}$, the* feasibility 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.*

Theorem 3 *For any deterministic and memoryless algorithm A, if an asynchronous constrained deadline system τ is A-feasible, then the A-feasible schedule of τ on m uniform processors is finally periodic, i.e. from some point the schedule repeats. (Assuming that the execution time of each task is constant.)*

Proof. First notice that from $t_0 \geq O_{\max}$ all tasks are released, and the configuration $\theta_i(t)$ of each task is a triple of finite integers (α, β, γ) with $\alpha \in \{0, 1\}, 0 \leq \beta < \max_{1 \leq i \leq n} T_i$ and $0 \leq \gamma < \max_{1 \leq i \leq n} C_i$. Therefore there is a finite number of possible combinations and we can find two instants t_1 and t_2 ($t_2 > t_1 \geq t_0$) with the same state of the system. The schedule repeats from that instant with a period dividing $t_2 - t_1$, since the scheduler is deterministic and memoryless. □

Notice that non-preemptive scheduling algorithms are not memoryless[5] since at instant t the scheduler must know which task[s] was scheduled at instant $t-1$ to make a scheduling decision and to avoid preemption.

[4] by deterministic EDF we mean that ambiguous situations are solved deterministically.

[5] at least with our definitions (Definition 1 in particular).

Corollary 4 *For any preemptive fixed-priority algorithm A, if an asynchronous constrained deadline system τ is A-feasible, then the A-feasible schedule of τ on m uniform processors is finally periodic. (Assuming that the execution time of each task is constant.)*

Proof. The result is a direct consequence of Theorem 3, since a preemptive fixed-priority algorithm is deterministic and memoryless. □

Corollary 5 *A feasible schedule obtained using deterministic global* EDF *on m multiprocessor platforms of an asynchronous constrained deadline system τ is finally periodic. (Assuming that the execution time of each task is constant.)*

Proof. The result is a direct consequence of Theorem 3, since deterministic EDF is deterministic and memoryless. □

Notice that the scheduler may intentionally idle one or several processors in order to satisfy some constraints of the system, i.e., the scheduler may be not work-conserving. In other words, Theorem 3 remains if the scheduler idles the processor[s] in a deterministic and memoryless way.

3.1 The particular case of synchronous periodic systems

In this section we deal with the particular case of synchronous periodic task systems and we show the periodicity of feasible schedules obtained using, as in Section 3, deterministic and memoryless algorithms.

Theorem 6 *For any deterministic and memoryless algorithm A, if a synchronous constrained deadline system τ is A-feasible, then the A-feasible schedule of τ on m uniform processors is periodic with a period P that begins at instant 0. (Assuming that the execution time of each task is constant.)*

Proof. Since τ is a synchronous periodic system, all tasks become active at instants 0 and P. Moreover, since τ is a A-feasible constrained deadline system, all jobs occurred strictly before instant P have finished their execution before or at instant P. Therefore at instants 0 and P the system is in the same state, i.e. $\theta(0) = \theta(P)$, and a deterministic and memoryless scheduling algorithm will make the same scheduling decision. The schedule repeats with a period equal to P.

□

An interesting special case of Theorem 6 is the following:

Corollary 7 *A feasible schedule obtained using deterministic global* EDF *of a synchronous constrained deadline system τ on m identical or uniform processors is periodic with a period P that begins at instant 0. (Assuming that the execution time of each task is constant.)*

Now we have the material to define a feasibility interval for synchronous constrained deadline systems.

Corollary 8 *For any preemptive fixed-priority algorithm A and any A-feasible synchronous constrained deadline system τ on m uniform processors, the interval $[0, P)$ is a feasibility interval.*

Proof. The result is a direct consequence of Theorem 6 and Theorem 2, since fixed-priority schedulers are priority-driven. □

Notice that we do not assume here that the task execution times are constant, the task parameter C_i represents the *worst-case* execution time.

4 Fixed-priority scheduling of asynchronous systems

In this section we give our main result: any feasible schedules on m uniform processors of asynchronous constrained deadline systems, obtained using preemptive fixed-priority algorithms, are periodic from some point (Theorem 9), assuming that the execution time of each task is constant. Moreover, we define a feasibility interval for this case (Corollary 11).

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

We introduce now 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 ordered set of m processors $\{p_1, \ldots, p_m\}$ we define the* availability of the processors $a(t)$ *of system τ at instant t as the set of available processors $a(t) \stackrel{\text{def}}{=} \{j \mid \sigma_j(t) = 0\} \subseteq \{1, \ldots, m\}$.*

Theorem 9 *For any preemptive fixed-priority algorithm A, if an asynchronous constrained deadline system τ is A-feasible, then the A-feasible schedule of τ on m uniform processors is periodic with a period P from instant S_n where S_i is defined inductively as follows:*

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

(Assuming that the execution time of each task is constant.)

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 instant $P_1 = T_1$ the previous request of τ_1 has finished its execution and the schedule repeats.

We shall now show that any A-feasible schedule of $\tau^{(i+1)}$ is 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:

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

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}, then 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 very same instants and on the very same processors as they were in $\sigma^{(i)}$. We apply now the induction step: for all $t \geq S_i$ in $\sigma^{(i)}$ we have $a^{(i)}(t) = a^{(i)}(t + P_i)$ the availability of the processors repeats. Notice that at those instants t and $t + P_i$ the available processors (if any) are the same. Consequently at only these instants task τ_{i+1} *may* be executed. Notice that the scheduler can decide to leave one or several processor(s) to be idle intentionally in a deterministic and memoryless way.

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} since P_{i+1} is a multiple of P_i. Moreover since the system is feasible 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 deterministic and memoryless 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 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 period equal to P from S_n and the property follows. □

An interesting application of Theorem 9 concerns identical processors:

Corollary 10 *For any preemptive fixed-priority algorithm A, if an asynchronous constrained deadline system τ is A-feasible on m identical processors, then the A-feasible schedule of τ is periodic with a period P from instant S_n where S_i are defined inductively as defined in Theorem 9. (Assuming that the execution time of each task is constant.)*

Now we have the material to define a feasibility interval for asynchronous constrained deadline periodic systems.

Corollary 11 *For any preemptive work-conserving fixed-priority algorithm A and for any A-feasible asynchronous constrained deadline system τ on m uniform processors,*

$[0, S_n + P)$ is a feasibility interval where S_i are defined inductively in Theorem 9. Moreover, for every task τ_i one only has to check the deadlines in the interval $[S_i, S_i + \text{lcm}\{T_j | j \leq i\})$.

Proof. The Corollary 11 is a direct consequence of Corollary 10 and Theorem 2, since fixed-priority algorithms are priority-driven. □

The feasibility interval given by Corollary 11 may be improved as it was done in the uniprocessor case [19]. In that paper, the authors introduced several intermediary results proved by induction in order to give the following theorem:

Theorem 12 *[19] Let X_i be inductively defined by $X_n = S_n, X_i = O_i + \lfloor \frac{X_{i+1} - O_i}{T_i} \rfloor T_i$ ($i \in \{n-1, n-2, \cdots, 1\}$; then $[X_1, S_n + P]$ is a feasibility interval.*

Following the proof of Theorem 9, we can extend the reasoning of Theorem 12 to the multiprocessor case, since it does not depend on the number of processors nor on the kind of platforms but on the availability of the processors.

Theorem 13 *For any A-feasible system τ on m processors, $[X_1, S_n + P]$ is a feasibility interval where X_i are defined inductively in Theorem 12.*

4.1 Comparison with earlier work

In this section we compare our results to those presented in [16]. We show that, even if the instant X_1 (Theorem 12) is not necessary the earliest instant from which the schedule repeats, our results are more general, i.e., they are given for $\mathbf{m}$ uniform processors and constrained deadline task systems.

In [16] preemptive fixed-priority scheduling of asynchronous implicit deadline task systems on two identical processors is considered. The authors extended results obtained for uniprocessor case [17] to the case of *two* processors and they showed that any feasible schedules repeat immediately after a time instant called *the last acyclic idle time*. The last acyclic idle time t_c is defined as the last time instant when at least one processor idles which does not repeat with a period equal to P. No upper bound is provided for t_c, but the authors suggest that $O_{\max} + P$ might be such a bound as it was for the uniprocessor case.

As the authors mention, the results obtained in [16] do not remain for scheduling algorithms which are not work-conserving, e.g. algorithm introducing intentionally idle times in a deterministic and memoryless way.

Moreover from the applicability point of view, in order to verify the feasibility of a periodic task system, the last acyclic idle time must be found. It implies that one needs to observe the schedule at least from instant $O_{\max}$ to instant $O_{\max} + 2P$, if we suppose that the bound remains the same as for the uniprocessor case. The length of the interval that must be observed is $\simeq 2P$ and it is longer than the interval that must be observed in our case ($\simeq P$). Indeed in order to make the distinction between cyclic and acyclic idle times, one must observe an interval of length P, t_c cannot appear before $O_{\max}$ and the upper bound on t_c is $O_{\max} + P$. Once the last idle time t_0 was found, the interval $[t_0, t_0 + P)$ is studied in order to determine if the system is feasible. Notice that the complexity of our algorithm allowing to find the explicit form of the feasibility intervals is $\mathcal{O}(n)$ (i.e., the computations of S_n and X_1). While Geniet's algorithm needs to consider a full hyper-period P.

In conclusion our results, concerning the fact that the schedule repeats, are more general than those presented in [16], since they are given for constrained deadline systems on $\mathbf{m}$ uniform processors and can be applied to scheduling algorithms introducing intentionally idle times in a deterministic and memoryless way. Moreover we show that priority-driven algorithms are predictable on uniform multiprocessors and we provide a feasibility test considering that the task parameter C_i is the worst-case execution time while Geniet considers that the execution times are constant.

5 Conclusion and future work

In this work we extended some results concerning the periodicity of feasible schedules and the feasibility intervals from the uniprocessor case to the multiprocessor case. We showed that any feasible schedules of a periodic task system on uniform parallel processors repeat from some point. For synchronous constrained deadline task systems we showed that this point is the origin and we provided feasibility intervals for these systems. Then, we presented the main contribution of this paper: feasibility intervals for asynchronous constrained deadline task systems obtained using preemptive fixed-priority algorithms.

As future work we are interested in obtaining results concerning feasibility intervals for asynchronous constrained deadline task systems obtained using deterministic EDF on multiprocessor platforms.

6 Acknowledgements

We would like to thank an anonymous reviewer for his comments that allowed us to improve some ambiguous definitions and notations.

References

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

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

[3] S.K. Dhall and C.L. Liu. On a real-time scheduling problem. *Operations Research(26)*, pages 127–140, 1978.

[4] K. Hong and J. Leung. On-line scheduling of real-time tasks. *Proceedings of the Real-Time Systems Symposium*, pages 244–250, 1988.

[5] J. Anderson, P. Holman, and A. Srinivasan. Fair scheduling of real-time tasks on multiprocessors. *Handbook of Scheduling*, 2005.

[6] J. Anderson and A. Srinivasan. Early release fair scheduling. *Proceedings of ECRTS*, 2000.

[7] C. Phillips, C. Stein, E. Torng, and Wein J. Optimal time-critical scheduling via resource augmentation. *Proceedings of the twenty-ninth annual ACM Symposium on theory of computing*, pages 140–149, 2000.

[8] B. Andersson, S. Baruah, and J. Jansson. Static-priority scheduling on multiprocessors. *Proceedings of the 22nd IEEE Real-time Systems Symposium*, pages 193–202, 2001.

[9] J. Lopez, M. Garcia, and D. Garcia. Worst-case utilization bound for edf scheduling on real-time multiprocessor systems. *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 25–33, 2000.

[10] S. Baruah and J. Goossens. The static-priority scheduling of periodic task systems upon identical multiprocessor platforms. *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 427–432, 2003.

[11] S. Baruah, J. Goossens, and S. Funk. Robustness results concerning edf scheduling upon uniform multiprocessors. *IEEE Transactions on Computers, 52(9)*, pages 966–970, 2003.

[12] S. Baruah and J. Goossens. Rate-monotonic scheduling on uniform multiprocessors. *IEEE Transactions on Computers, 52(7)*, pages 966–970, 2003.

[13] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-time Systems 25 (2-3)*, pages 187–205, 2003.

[14] J. Goossens. Scheduling of offset free systems. *Real-Time Systems (24)*, pages 239–258, 2003.

[15] T.P. Baker. Comparison of empirical success rates of global vs. partitioned fixed-priority edf scheduling for hard real time. Technical report, Florida State University, Tallahassee, 2005.

[16] A. Choquet-Geniet. Un premier pas vers l'étude de la cyclicité en environnement multiprocesseurs. *Proceedings of the 13th International Conference on Real-Time Systems*, pages 289–302, 2005.

[17] A. Choquet-Geniet and E. Grolleau. Minimal schedulability interval for real-time systems of periodic tasks with offsets. *Journal of Theoretical Computer Science*, pages 117–134, 2003.

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

[19] J. Goossens and R. Devillers. The non-optimality of the monotonic assignments for hard real-time offset free systems. *The Journal of Real-Time Systems*, pages 107–126, 1997.

[20] R. Ha and J.W.S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, 1994.