Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms

Liliana Cucu-Grosjean a, *, 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

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 modeled 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 $t_i$ 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 \( \pi_i \) 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 \( s_{ij} \) associated with each job-processor pair, a job \( J_i \) that is executed on processor \( \pi_i \) for \( t \) time units completes \( s_{ij} \times t \) units of execution. This kind of architecture is a model for dedicated processors (e.g., if \( s_{ij} = 0 \) then \( \pi_i \) cannot execute job \( J_j \)).

1.1. 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., Carpenter et al. [3] present 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 post 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.

1.2. 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 \( \epsilon(l) \) cm \((T_1, T_2, \ldots, T_n)\), which is unfortunately also the case of most schedulability tests for simpler uniprocessor 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].

1.3. 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 \( \tau \) is composed of \( n \) periodic tasks \( T_1, T_2, \ldots, T_n \), 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 \( T_j \)) 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 \( D_i < T_i \), \( \forall i \) and arbitrary deadline systems where there is no relation between the deadlines and the period. The periods 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., \( \exists T_1, T_2, \ldots, T_n \) such that \( T_1 = O_i + kT_i \) (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 \( T_i^{(l)} \) \( T_i^{(l)} = T_1, \ldots, T_l \), by \( \mathcal{O}_j^{(l)} \) \( \mathcal{O}_j^{(l)} = \{O_1, \ldots, O_l\} \), by \( \mathcal{O}_j^{(l)} \) \( \mathcal{O}_j^{(l)} = \{O_1, \ldots, O_l\} \), by \( \mathcal{P}_j^{(l)} \) \( \mathcal{P}_j^{(l)} = \{P_0, P_1, \ldots, P_l \} \) cm \((T_1, \ldots, T_l) \) \((0 < l \leq n) \) and by \( P_j^{(l)} = P_j \) by \( P_j^{(l)} = P_j \). In the following, the quantity \( P \) is called the task set hyper-period.

We consider multiprocessor platforms \( \pi \) composed of \( m \) unrelated processors (or one of its particular cases: uniform and identical platforms): \([T_1, T_2, \ldots, T_m]\). Execution rates \( s_{ij} \) are associated with each task-processor pair. A task \( T_j \) that is executed on processor \( \pi_i \) for \( t \) time units completes \( s_{ij} \times t \) units of execution. For each task \( T_j \) we assume that the associated set of processors \( \pi_{n_1} > \pi_{n_2} > \cdots > \pi_{n_n} \) are ordered in a decreasing order of the execution rates relative to the task: \( s_{n_1} \geq s_{n_2} \geq \cdots \geq s_{n_n} \). 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 \( T_j \) is \( \pi_{n_1} \), i.e., the first processor of the ordered set associated with the task. Moreover, for a task \( T_j \) in the following we consider that a processor \( \pi_a \) is faster than \( \pi_a \) (relative to its associated set of processors) if \( s_{n} > s_{n} \) even if we have \( s_{n} = s_{n} \). For the processor-task pair \((\pi_i, \pi_j)\) if \( s_{ij} \neq 0 \) then \( \pi_i \) is said to be an eligible processor for \( \pi_j \). 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 $\tau$ be a system of three periodic tasks

\begin{align*}
\tau_1 & : O_1 = 0, C_1 = 2, D_1 = 6, T_1 = 6 \\
\tau_2 & : O_2 = 5, C_2 = 4, D_2 = 6, T_2 = 6 \\
\tau_3 & : O_3 = 4, C_3 = 5, D_3 = 7, T_3 = 6
\end{align*}

Let $\pi$ be a multiprocessor platform of two unrelated processors $\pi_1$ and $\pi_2$. We have $S_11 = 1, S_12 = 2, S_21 = 2, S_22 = 1, S_31 = 2$ and $S_32 = 1$. Note that according to our definition above and regarding task $\tau_1$ we have that $S_2 > \pi_1$.

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 at time-instant $t$ as $\theta : N \rightarrow (\mathbb{Z} \times N \times N)^n$ with $\theta(t) \equiv (\theta_1(t), \theta_2(t), \ldots, \theta_n(t))$ where

$$
\begin{cases}
(-1,x,0), & \text{if no job of task } \tau_i \text{ was activated before or at } t. \\
(0,y,u), & \text{otherwise.}
\end{cases}
$$

In this case, $x$ time units remain until the first activation of $\tau_i$. We have $0 < x < 0$.

For every job $\tau_i$ we have the following property:

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

Not 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 $\sigma_i(t)$ of system at time-instant $t$ as $\sigma_i : N \rightarrow (0,1,\ldots,n)^m$ where $\sigma_i(t) \equiv \sigma_{\tau_1}(t), \sigma_{\tau_2}(t), \ldots, \sigma_{\tau_m}(t)$ with

$$
\sigma_{\tau_i}(t) \equiv \begin{cases}
0, & \text{if there is no task scheduled on } \pi_i, \\
\text{at time-instant } t, & \forall 1 \leq j \leq m. \\
i, & \text{if task } \tau_i \text{ is scheduled on } \pi_j \text{ at time-instant } t.
\end{cases}
$$

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

A system $\pi$ 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 $\pi$ on a multiprocessor platform to meet its deadlines, then the system $\pi$ is said to be A-schedulable.

Note that for a feasible schedule from Definition 1 we have $0 \leq y \leq \frac{O_i}{C_i}, 0 \leq z < T_i \cdot \frac{D_i}{C_i}$ 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., $\exists \pi_i \neq \pi_j \in \{1, \ldots, m\}$ and $t \in \mathbb{N}$ such that $\sigma_{\tau_i}(t) = \sigma_{\tau_j}(t) = 0$.

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 $\pi_i$. 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 $\pi_i$ can be idle and a job $J_i$ can be active at the same time if and only if $S_\tau_i = 0$.

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:

$$
\forall t > t_0 \text{ such that } \theta(t_0) = \theta(t_2) \text{ then } \sigma_i(t_1) = \sigma_i(t_2).
$$

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 $J_j$ has higher priority than $J_i$ at some time-instant, then $J_j$ always has higher priority than $J_i$.

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 $J_i$ of a task $\tau_i$). For any task $\tau_i$, we define $J_i(t)$ to be the $i$th job of task $\tau_i$, which becomes active at time-instant $R_i^6(t) = O_i + (k-1)T_i$. For two tasks $\tau_i$ and $\tau_j$, by $J_i(t) > J_j(t)$ we mean that the job $J_i(t)$ has a higher priority than the job $J_j(t)$.

Definition 9 (Executed time $e_i^j(t)$ for a job $J_i^j$). For any task $\tau_i$, we define $e_i^j(t)$ to be the amount of time already executed for $J_i^j$ in the interval $[R_i^6(t), t]$.

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 $\tau$ at time-instant $t$ as the set of available processors $a(t) \equiv \{j : a_j(t) = 0\} \subseteq \{1, \ldots, m\}$.

In Example 1 for a preemptive task-level fixed-priority algorithm with $\tau_1$ the highest priority task and $\tau_2$ the lowest priority task, we obtain the schedule provided in Fig. 1. In this figure the execution on the first processor is indicated by black boxes. The releases of jobs are indicated by $|$ and the deadlines by $\tau$. For instance we have $\sigma_2(2) = (-1,3,0), \sigma_3(3) = (3,0,0)$ and $\sigma_3(6) = (2,1,3,0)$. Moreover $\sigma(6) = (2,1)$ and $\sigma(7) = (3,0)$. Concern-
that each job of the same task (say, \( t_s \)) becomes active at \( R_s^1 = 6 \) and this job has a higher priority than the job \( j_s^2 \) of task \( t_2 \). For this schedule \( a(1) = \{ 1 \} \) and \( a(6) = 8 \).

### 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, \( t_s \)) has the same execution requirement (C). 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 \( \gamma \)) of a schedule \( \sigma \), we mean there is a time-instant \( t_0 \) and an interval length \( \gamma \) such that \( \sigma(t) = \sigma(t + \gamma) \), for \( t \geq t_0 \). For instance, in Fig. 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 sub-class 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 (Corollary 5).

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

**Lemma 1.** For any deterministic and memoryless algorithm \( A \) and any asynchronous arbitrary deadline system \( \tau \) on \( m \) unrelated processors, if then the feasible schedule of \( \tau \) on \( \prod \) obtained using \( A \) is periodic with a period divisible by \( P \).

**Proof.** First note that from any time instant \( t_0 \geq O_{\text{max}} \) all tasks are released, and the configuration \( \theta(t_0) \) of each task is a triple of finite integers \( (x, \beta, \gamma) \) with \( x \in \{0, 1, \ldots, \frac{O_{\text{max}}}{T_i} \} \), \( 0 < \beta < \max_{t < t_0} (T_i \times \frac{O_{\text{max}}}{T_i}) \) and \( 0 < \gamma < \max_{t < t_0} (O_{\text{max}}) \).

Therefore, there is a finite number of different system states, hence we can find two distinct instants \( t_1 \) and \( t_2 \) \( (t_2 > t_1 > t_0) \) 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 \( O_{\text{max}} \).

We now prove by contradiction that \( t_2 - t_1 \) is necessarily a multiple of \( P \). We suppose that \( R_i < R_j \) 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, \( \gamma \) must be equal to \( \Delta_2 \) and, thus, we have \( t_2 - t_1 = (k_2 - k_1)P \). \( \square \)

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 \( \forall i, j, k \in \mathbb{N} \) such that \( R_i^j > R_k^j \) if and only if \( h_i^j > h_k^j \).

Note that Definition 11 (request-dependent) is stronger than Definition 7 (job-level fixed-priority). Informally speaking Definition 11 requires that the total order used is each hyper-period with the corresponding jobs (job “corresponds” to \( R_i^j \)). For instance in Example 1 for the schedule of Fig. 1 we have \( J_1^1 > J_2^1 \) and \( J_2^1 > J_2^3 \) because the hyper-period is 6.

The next lemma extends results given for arbitrary deadline task systems in the uniprocessor case ([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 \( \tau \) on \( m \) unrelated processors, we have: for each task \( \tau_s \), for any time-instant \( t \geq O_{\text{max}} \) and \( k \) such that \( R_s^k \leq t < R_s^{k+1} + D_s \), if there is no deadline missed up to time \( t + P \), then \( \theta(t) = \theta(t + P) \) with \( h_i^j = \frac{h_i^j}{P} \).

**Proof.** Note first that the function \( \theta(t) \) is a non-decreasing discrete step function with \( \theta(t) \in C \) for all \( t \) and \( \theta(t^k) = 0 = \theta(t^{k+1}) \) (C).

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

The proof is made by contradiction. Thus, we assume that a first time-instant \( t \) exists such that there is \( j \) and \( k \) with \( R_s^k \leq t < R_s^{k+1} + D_s \) and \( \theta(t) \neq \theta(t + P) \). This assumption implies that:

1. either there is a time-instant \( t' \) with \( R_s^k \leq t' < t \) such that \( J_i^j \) is scheduled at \( t' + P \) while \( J_j^k \) is not scheduled at \( t' \). We obtain that there is at least one job \( J_i^j \) of the task \( \tau_s \) with \( t \in \{1, 2, \ldots, n\} \) that is not scheduled at \( t' + P \), while \( J_j^k \) is scheduled at \( t'(h_i^j = \frac{P}{h_i^j}) \). 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 \( O_{\text{max}} \) and there is no deadline missed) that \( J_j^k \) has not finished its execution before \( t' \), but \( J_i^j \) has finished its execution before \( t' + P \). Therefore, we have \( \theta(t') = \theta(t' + P) = C \), which implies that \( \theta(t') < \theta(t' + P) \). 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 \( t' + P < t < t + P \) when \( J_i^j \) 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 \( J_i^j \) of the task \( \tau_s \) with \( t \in \{1, 2, \ldots, n\} \) that is not scheduled at \( t' + P \), while \( J_j^k \) is scheduled at \( t'(h_i^j = \frac{P}{h_i^j}) \) as proved in the first case, this situation leads us to a contradiction with our assumption. \( \square \)

---

1 The method chosen to break deadline ties must be deterministic.

2 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 Fig. 1, we have from Lemma 2 that for \( t = 6 \) and \( k = 1 \), \( \epsilon_1^1(6) = 3 = \epsilon_2^2(12) \) 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 \( \tau \) on \( m \) unrelated processors the schedule is periodic with a period equal to \( P \).

**Proof.** By Lemma 1 we have \( \exists t = O_{\text{max}} + kP + d, \forall i \in \{1, 2\} \) with \( 0 < d < P \) 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 \( O_{\text{max}} \). Therefore, for all time-instants \( t_1 + kP, \forall k < k_1 - k_1 \) (ie. \( t_1 + kP < t_2 \)), 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 \( \epsilon_1^i(t_1) = \epsilon_1^i(t_2) \) with \( \epsilon_1^i \overset{\text{def}}{=} t_1 - c_{i1} \), \( \forall i \). But by Lemma 2 we also have \( \epsilon_1^i(t_1) \geq \epsilon_1^1(t_1 + P) \geq \cdots \geq \epsilon_1^i(t_2), \forall i \). 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 implies that \( \theta(t_1) = \theta(t_1 + P) = \cdots = \theta(t_2) \). \( \square \)

**Corollary 4.** For any preemptive task-level fixed-priority algorithm A, if an asynchronous arbitrary deadline system \( \tau \) 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. \( \square \)

**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 \( t_1 > t_2 > \cdots > t_n \).

**Lemma 6.** For any preemptive task-level fixed-priority algorithm A and for any synchronous arbitrary deadline system \( \tau \) 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 \( \tau \) is periodic with a period \( P \) that begins at time-instant 0. \( ^{3} \)

**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 \( P \). \( \square \)

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

**Proof.** In the following, we denote by \( \sigma^0 \) the schedule of the task subset \( \tau^0 \).

Since \( \theta(0) = \theta(P) \), 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 \( t_\ell \) has at least two active jobs at \( P \), i.e. \( \ell \) is the smallest task index such that \( \theta(0) \neq \theta(P) \). In order to prove the property we will prove that \( t_\ell \) will miss a deadline.

By definition of \( \ell \) 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 \( P_{\ell - 1} \) is a multiple of \( P_{\ell - 1} \), we know that the schedule \( \sigma^{(\ell - 1)} \) is periodic with a period \( P_{\ell} \). Therefore, in each time interval \([kP, (k + 1)P] \) with \( k \geq 0 \) after scheduling \( t_1, t_2, \ldots, t_{\ell - 1} \) there is the same number of time instants such that at least one processor is available and where \( t_\ell \) could be scheduled on its fastest processor among the available processors. We denote by \( \eta_{ij}, \forall j \in \{1, 2, \ldots, m\} \) the number (on \( t_j \)) of time instants when processor \( p_i \) is available and it is the fastest processor for \( t_j \). Therefore, we have \( \sum_{j=1}^{m} \eta_{ij} = t_j \) and \( t_j \) completes \( \sum_{j=1}^{m} \eta_{ij} \) units of execution in the interval \([kP, (k + 1)P] \).

At time-instant \( P_{\ell} \), since task parallelism is forbidden, there are \( \sum_{j=1}^{m} C_j - \sum_{j=1}^{m} \eta_{ij} \) remaining units for execution of \( t_\ell \) and, consequently, at each time-instant \( kP \) there will be \( \sum_{j=1}^{m} C_j - \sum_{j=1}^{m} \eta_{ij} \) remaining units for execution of \( t_\ell \). Consequently, we can find \( k_1 = \frac{D_{\ell} - \sum_{j=1}^{m} \eta_{ij}}{P_{\ell} - C_{\ell} - \sum_{j=1}^{m} \eta_{ij}} \) such that the job activated at \( kP \), will miss its deadline since it cannot be scheduled before older jobs of \( t_\ell \) and there are \( k_1 \) units of execution for \( t_\ell \) at \( kP \).

Since we consider task-level fixed-priority scheduling, then the tasks \( t_\ell \) with \( \ell > \ell \) will not interfere with the higher priority tasks already scheduled, particularly with \( t_\ell \) that misses its deadline, and consequently the system is not A-schedulable. \( \square \)

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

**Proof.** Since \( \tau \) 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 \( P \). \( \square \)

**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 $\tau$ on $m$ unrelated processors, the schedule is periodic with a period $P_n$ from time-instant $S_n$ where $S_n$ is defined inductively as follows:

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

**Proof.** The proof is made by induction on $n$ (the number of tasks).

We denote by $\sigma^0$ the schedule obtained by considering only the task subset $t^0(\tau)$, the first higher priority $i$ tasks $\{t_1, \ldots, t_i\}$, and by $\sigma^0_i$, the corresponding availability of the processors. Our inductive hypothesis is the following: the schedule $\sigma^k$ is periodic from $S_k$ with a period $P_i$ 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 $t^1(\tau) = \{t_1\}$; since we consider constrained deadline systems, at time-instant $P_1 = T_1$, the previous request of $t_1$ has finished its execution and the schedule repeats.

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

Since $\sigma^0$ is periodic with a period $P_i$ from $S_i$, the following equation is verified:

$$\sigma^0(t) = \sigma^0(t + P_i), \quad \forall t \geq S_i.$$ (1)

We denote by $S_{i+1}^n \overset{\text{def}}{=} \max\{O_{i+1}, O_{i+1} + \lfloor \frac{S_i - S_{i+1}}{C_{i+1}} \rfloor T_{i+1}\}$ the first request of $t_{i+1}$ not before $S_i$.

Since the tasks in $t^0$ have higher priority than $t_{i+1}$, the scheduling of $t_{i+1}$ will not interfere with higher priority tasks which are already scheduled. Therefore, we may build $\sigma^i$ from $\sigma^0$ such that the tasks $t_1, t_2, \ldots, t_i$ are scheduled at the same instants and on the same processors as they were in $\sigma^0$. We now apply the induction step: for all $t > S_i$ in $\sigma^0$ we have $\sigma^0(t) = \sigma^0(t + P_i)$, thus the availability of the processors repeats. Task $t_{i+1}$ 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 $t_{i+1}$ may be executed in $\sigma^i$, are periodic with period $P_{i+1}$ from $S_{i+1}$ (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 $t_{i+1}$ are periodic with period $P_{i+1}$. Therefore, the schedule $\sigma^i$ 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 $P_n$ from $S_n$ and the property follows. $\square$

**Example 2.** Let $\tau$ be a system of three periodic tasks

<table>
<thead>
<tr>
<th>$O_i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$T_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>0</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>$t_2$</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>$t_3$</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Let $\pi$ be a multiprocessor platform of 2 unrelated processors $P_1$ and $P_2$. We have $s_1 = 1$, $s_2 = 2$, $s_2 = 5$, $s_2 = 1$, $s_3 = 2$ and $s_5 = 1$.

Using a task-level fixed-priority scheduling algorithm with $\tau_1$ the highest priority, we obtain the schedule provided in Fig. 2. For this schedule and the notations of Theorem 9, we have $S_1 = 0$, $S_2 = 5$ and $S_3 = 10$. In Fig. 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 $\tau$ on $m$ unrelated processors, we have: for each task $t_i$, 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 $t + P_i$, then $\epsilon^i(t) \geq \epsilon^i(t + P_i)$ with $h_i \overset{\text{def}}{=} \frac{R^i_k}{C_i}$.

**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. $\square$

**Corollary 11.** For any preemptive task-level fixed-priority algorithm $A$ and any asynchronous arbitrary deadline system $\tau$ on $m$ unrelated processors, we have: for each task $t_i$, for any time-instant $t \geq \max\{O_i, O_i + \lfloor \frac{S_i - S_{i+1}}{C_{i+1}} \rfloor T_{i+1}\}$, if there is no deadline missed up to time $t + P_i$, then either $\epsilon^i(t) < \epsilon^i(t + P_i)$ or $\epsilon^i(t + P_i)$. Consequenlty $\epsilon^i(t + P_i) \geq \epsilon^i(t)$ and if $s_i(t) = \epsilon^i(t + P_i)$ then $s_i(t + P_i) = s_i(t + P_i)$ and $\gamma(t) = \gamma(t + P_i) = \gamma(t + P_i)$. The last inequality ( $\epsilon^i(t + P_i) \geq \epsilon^i(t + P_i)$ ) is a direct consequence of Corollary 10. $\square$

**Lemma 12.** For any preemptive task-level fixed-priority algorithm $A$ and any $A$-schedulable asynchronous arbitrary deadline system $\tau$ on $n$ tasks on $m$ unrelated processors, let $\sigma^0$ the schedule obtained by considering only the task subset $t^0$. Moreover, let the set of $\{S_1, \ldots, S_n\}$ be defined inductively as follows:

- $S_1 \overset{\text{def}}{=} O_1$;
- $S_i \overset{\text{def}}{=} \max\{O_i, O_i + \lfloor \frac{S_i - S_{i-1}}{C_i} \rfloor T_i\} + P_i$. ($i > 1$)

**Fig. 2.** Schedule (obtained using a task-level fixed-priority scheduling) for the task set of Example 2.
If \( \theta_{t+1}(\tilde{S}_{i+1}) = \theta_{t+1}(\tilde{S}_{i+1} + P_{i+1}) \), then \( t \in \tilde{S}_{i+1} = S_{i+1} + P_{i+1} \) such that at \( t \) there is at least one available (and eligible) processor for \( \tau_{i+1} \) in \( \sigma(0) \) and no job of \( \tau_{i+1} \) is scheduled at \( t \) in \( \sigma(t) \).

**Proof.** The proof is made by contradiction.

If there is such a time-instant \( t^* \in [\tilde{S}_{i+1}, \tilde{S}_{i+1} + P_{i+1}] \) then \( \alpha_{t^*}(t^*) = 0 \). Thus, by Corollary 11 it is contradictory with the hypothesis of the lemma.

**Definition 13.** (Schedulability interval) For any \( m \)-processor system \( \tau \), and any set of m processors \( \{p_1, \ldots, p_m\} \), 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.

**4.4. Basic results**

In this section, we consider the scheduling of sets of job \( J^f_i \) (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 \( J_1 \) can be any value in the interval \( [e^r_i, e^d_i] \) and we denote with \( J_1^f \) the job defined as \( J_1^f \). The associated execution rates of \( J_1^f \) are equal to \( s_{i+1} \) which we denote by \( f_{j_i} \) the set of the first i higher priority jobs. We denote also by \( f_{J_1} \) the set \( \{J_1, \ldots, J_3\} \) and by \( f_{J_1}^f \) the set \( \{J_1^f, \ldots, J_3^f\} \). Note that the schedule of an ordered set of jobs using a work-conserving and job-level fixed-priority algorithm is unique. Let \( S(J_1) \) be the time-instant at which the lowest priority job of \( J_1 \) 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)** A scheduling algorithm is said to be predictable if \( S(J_1) = S_{J_1} \leq S_{J_1}^f \) and \( F(J_1) = F_{J_1}^f \) for all \( 1 \leq i \leq \ell \) and for all predictable scheduling algorithms.

**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 $\tau$ on unrelated multiprocessors, $\tau$ 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 $\tau_i$ only the deadlines in the interval $[S_i, S_i + 1 \text{cm}(T_{ij} \leq i)]$ 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 $X_n = S_n$, $X_i = O_i + \sum_{j=1}^{i-1} T_j$ ($i \in \{n - 1, n - 2, \ldots, 1\}$); thus $\tau$ is $A$-schedulable if and only if all deadlines are met in $[X_i, S_n + P]$ and $\theta(S_n) = \theta(S_n + P)$.

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 $\tau$ on $m$ unrelated processors, $\tau$ is $A$-schedulable if and only if all deadlines are met in $(0, S_n + P)$ and $\theta(S_n) = \theta(S_n + P)$, where $S_i$ 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 uniprocessor 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 $O_{\text{max}} + P$ 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 $\tau$
Output: feasible
begin
Schedule (from 0) to $O_{\text{max}}$;
(The function Schedule stops the program and returns false once a deadline is missed)
$s_1 := \theta(O_{\text{max}})$;
Schedule (from $O_{\text{max}}$ to $O_{\text{max}} + P$);
$s_2 := \theta(O_{\text{max}} + P)$;
current-time := $O_{\text{max}} + P$;
While $s_1 \neq s_2$ do
\begin{align*}
&s_1 := s_2; \\
&S\text{chedule} \text{ (from current-time) to current-time+} P; \\
&\text{current-time := current-time+} P; \\
&s_2 := \theta(\text{current-time}); \\
&\text{return true;}
\end{align*}
end

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 $\tau$ 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.

References
