CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Feasibility Intervals for Homogeneous Multicores, Asynchronous Periodic Tasks, and FJP Schedulers

Vincent Nelis

Patrick Meumeu Yomsi

Joel Goossens

CISTER-TR-131005

Version:

Date: 10/28/2013

Feasibility Intervals for Homogeneous Multicores, Asynchronous Periodic Tasks, and FJP Schedulers

Vincent Nelis, Patrick Meumeu Yomsi, Joel Goossens

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: nelis@isep.ipp.pt, pamyo@isep.ipp.pt, joel.goossens@ulb.ac.be

http://www.cister.isep.ipp.pt

Abstract

We address the problem of scheduling asynchronous periodic real-time tasks on homogeneous multicore platforms using a global and Fixed Job-level Priority (FJP) scheduler, e.g., global-EDF (global Earliest Deadline First). We establish a finite interval of time such that, if no task deadline is missed while scheduling only the jobs released within this interval, then no task deadline will ever be missed at run time. This kind of interval is referred to as “feasibility interval” and allows for sufficient and necessary schedulability analyses.

© CISTER Research Unit
www.cister.isep.ipp.pt

Feasibility Intervals for Homogeneous Multicores, Asynchronous Periodic Tasks, and FJP Schedulers*

Vincent Nelis
CISTER/INESC-TEC
ISEP, Polytechnic Institute of Porto
nelis@isep.ipp.pt

Patrick Meumeu Yomsi
CISTER/INESC-TEC
ISEP, Polytechnic Institute of Porto
pamyo@isep.ipp.pt

Joël Goossens
PARTS Research Center
Université Libre de Bruxelles (ULB)
joel.goossens@ulb.ac.be

ABSTRACT

We address the problem of scheduling asynchronous periodic real-time tasks on *homogeneous* multicore platforms using a global and Fixed Job-level Priority (FJP) scheduler, e.g., global-EDF (global Earliest Deadline First). We establish a finite interval of time such that, if no task deadline is missed while scheduling only the jobs released within this interval, then no task deadline will ever be missed at run time. This kind of interval is referred to as "feasibility interval" and allows for *sufficient and necessary* schedulability analyses.

1. INTRODUCTION

Many of the applications in the embedded systems arena have stringent timing requirements (the system is then referred to as "real-time" embedded system). Among these real-time systems, *hard* real-time systems are those for which violating one of these timing requirements can entail severe consequences, e.g., it can damage the system, lead to substantial economic loss, or even threaten human lives. Before these hard real-time systems can actually be deployed and marketed, they have to be *certified*: it has to be guaranteed at design time, that every "task" of the system will always meet its timing requirements at run time. To this end, the research community has designed many scheduling algorithms over the last decades, together with associated "schedulability analysis" techniques, that enable certification experts to provide these required guarantees. An abundant literature focusing on cost-effective certification techniques is available, regarding both unicore and multicore architectures. Essentially, there are two different methodologies to assert the so-called *schedulability* of a given set of tasks.

The first methodology consists in using *schedulability tests*. In its simplest form, a schedulability test is a *mathematical* condition such that, if the condition is satisfied then the system is asserted schedulable, i.e., all the task deadlines will always be met at run-time. In the same vein, there also exist *online* schedulability analysis techniques called "*admission tests*". These tests are used at *run-time* whenever a task is released to figure out whether the task can be accommodated and scheduled with the rest of the currently executing workload without violating any timing requirement. In case the incoming task fails the admission test, it can be simply ignored and dropped out, or there can be some feedback mechanisms implemented. Typically, these mechanisms send a feedback to a dedicated controller that modifies the system configuration accordingly, i.e., it regulates the parameters of the other tasks in order to make room for the incoming task.

The second methodology to certify the schedulability of a task system consists in *simulating the execution of the tasks* until a time-instant t such that, if no deadline is missed while scheduling only the tasks released within the time interval $[0, t)$ then no deadline will ever be missed during the run time. In the literature, such time intervals $[0, t)$ are often referred to as "*feasibility intervals*" and are the focus of this work. More precisely, we focus on *cyclic* feasibility interval formally defined as below.

Definition 1 (Cyclic feasibility interval). *Given a scheduling algorithm, a set of tasks and a computing platform (uni- or multi-cores), a* feasibility interval *is a* finite *interval of time $[t_0, t_1)$ such that, if no task deadline is missed while scheduling only the tasks released within $[t_0, t_1)$ then no deadline will ever be missed at run time. Further, a feasibility interval is said to be cyclic if and only if there exists a time-instant $t \in [t_0, t_1)$ such that the entire schedule within $[t, t_1)$ is repeated from time t_1 onward. That is, if $\Delta = t_1 - t$ the schedule in the time intervals $[t_1 + k\Delta, t_1 + (k+1)\Delta)$ for all $k = 0, 1, \ldots$ are identical to the schedule in $[t, t_1)$.*

The length of the feasibility intervals established so far can be pointed out as a former obstacle to their development. Typically, their length is either *optimistic* due to extension to multicore platforms of results obtained for unicore platforms [1][1], or *pessimistic* as it depends on the parameters of the tasks [3], which can result in extremely long intervals and thus, substantial simulation and computation time.

*This work was partially supported by the Fonds de la Recherche Scientifique - FNRS under Grant nr. (T.0102.13); by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within projects ref. FCOMP-01-0124-FEDER-022701 (CISTER) and ref. FCOMP-01-0124-FEDER-020447 (REGAIN); by National Funds through FCT and by the EU ARTEMIS JU funding within project ref. ARTEMIS/0003/2012, grant nr. 333053 (CONCERTO); by ERDF, through ON2 - North Portugal Regional Operational Programme, under the National Strategic Reference Framework (NSRF), within project ref. NORTE-07-0124-FEDER-000063 (BEST-CASE).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
RTNS 2013 , October 16 - 18 2013, Sophia Antipolis, France
Copyright 2013 ACM 978-1-4503-2058-0/13/10 ...$15.00.
http://dx.doi.org/10.1145/2516821.2516848.

[1]The results presented in [1] are flawed as pointed out in [2].

The latter drawback has played an important role in the past decades as the computing capabilities of 20-years-old computers were far behind those of today's computers.

Regarding the schedulability tests, most of the tests that have been proposed so far are only *sufficient*, i.e., there exist *schedulable* task systems that fail the tests. This may have heavy consequences in the design process as failing the test implies that the task parameters must be re-estimated and adjusted and/or the computing capacity of the platform must be enhanced. Besides, the authors of [4] have proven that the problem of verifying the schedulability of a task system in a sufficient and necessary fashion is PSPACE-complete, which means that unless PSPACE = NP = P, there will never be a sufficient and necessary test with a polynomial time-complexity[2]. For example, for the popular Global-EDF scheduler [7] and homogeneous multicore platforms (in which all the cores have the same computational capabilities and are interchangeable), to the best of our knowledge there is no sufficient and *necessary* schedulability test established so far, but the one obtained by Baker and Cirinei in 2007 [3] (and improved later by Lindström et al.in [4]). This test is based on solving a state reachability problem in a finite automaton and is of unacceptably high computational complexity as it performs an *exhaustive* search within a very large state-space. It is thus not practical except for very small task sets. Even though the approach adopted in [4] (which is based on techniques developed by the formal verification community and branch-and-bound alike algorithms) substantially reduces the number of states to be explored compared to [3], it has the same worst-case performance, regrettably. We refer the interested reader to [7,8] for a recent state of the art on the Global-EDF schedulability tests. Note that cyclic feasibility intervals inherently allow for sufficient and necessary schedulability analysis.

Contribution and organization of the paper. Recently, some essential results have paved the way for considering simulating the schedule of the task set to conclude on its schedulability. For example, the authors of [9] proposed an upper-bound on the length of the feasibility interval (and thus on the simulation time) for a given task set, assuming a Fixed Task-level Priority (FTP) scheduler. Then, Courbin et al.extended this analysis to parallel real-time tasks in [10]. However, this analysis does not extend to Fixed Job-level Priority (FJP) schedulers which assign a constant priority to each job upon its release (different jobs of the same task may have different priorities) and the aim of this work is to fill this gap: we provide an upper-bound on the length of the feasibility interval, assuming global FJP schedulers, asynchronous constrained-deadline periodic tasks and homogeneous multicore platforms (see the next section for the details on the model).

2. MODEL OF COMPUTATION

In our model, all the timing parameters and time-instants at which events occur are assumed to be non-negative integers, i.e., they are multiples of some irreducible time interval (e.g., the "clock tick", the smallest "user-defined" indivisible core time unit).

Task specifications. The workload is modeled by a set τ of n recurrent and *complete* tasks $\tau \stackrel{\text{def}}{=} \{\tau_1, \tau_2, \ldots, \tau_n\}$, where a complete task is defined in [11] as a task for which the first release time, the worst-case execution time, the deadline, and the rate/period are known at system design-time. We model each τ_i by a *constrained-deadline periodic* task characterized by four parameters $\langle O_i, C_i, D_i, T_i \rangle$—an offset O_i, a worst-case execution time C_i, a relative deadline $D_i \leq T_i$ and a period T_i that denotes the *exact* inter-arrival time between two consecutive releases of task τ_i. These parameters are interpreted as follows: during the execution of the system, task τ_i generates a (potentially infinite) sequence of *jobs* $\tau_{i,j}$ (with $j = 1, \ldots, \infty$) released at times $r_{i,j}$ such that $r_{i,j} \stackrel{\text{def}}{=} r_{i,j-1} + T_i$ (with $r_{i,1} \stackrel{\text{def}}{=} O_i$, the release time of the first job of task τ_i), each such job has an execution requirement of at most C_i time units and must complete by its absolute deadline $d_{i,j} \stackrel{\text{def}}{=} r_{i,j} + D_i$. Hereafter, we call an "instance" any finite or infinite collection of jobs and we call a "legal instance of τ" any instance that could be generated according to the parameters of the tasks in τ.

Without any loss of generality, we assume that $O_i \geq 0$, $\forall i \in [1, n]$, and we denote by $O_{\max}$ the maximal value among all task offsets, i.e., $O_{\max} \stackrel{\text{def}}{=} \max_{i=1}^{n}\{O_i\}$. Also, we denote by P the *hyper-period* of the task set, which is defined as the *least common multiple* of all tasks periods: $P \stackrel{\text{def}}{=} \text{lcm}\{T_1, T_2, \ldots, T_n\}$. If τ is synchronous (i.e., $O_i = O_j \; \forall i, j \in [1, n]$) then it has been proven in [9] that (i) it can be assumed without loss of generality that $O_i = 0 \; \forall i \in [1, n]$ and (ii) $[0, P)$ is a feasibility interval for both FTP and FJP schedulers. Otherwise, if τ is not synchronous (i.e., it is asynchronous with the meaning that $\exists i, j \in [1, n]$ with $i \neq j$ and $O_i \neq O_j$), then no feasibility interval is known for FJP schedulers and the main contribution of this paper is to fill this gap.

The following two definitions introduce two key concepts defined at run-time within a system schedule, namely, the task execution status and the system configuration.

DEFINITION 2 (EXECUTION STATUS $e_i(t)$). *The execution status $e_i(t)$ of task τ_i at time t is the number of time units during which its last released job has executed, from its release time up to time t. The execution status $e_i(t)$ is undefined for all $t < O_i$ since τ_i does not have any "last released job" before instant O_i.*

DEFINITION 3 (CONFIGURATION $C_{\mathcal{S}}(\tau, t)$). *Let $\mathcal{S}$ be the schedule of a task set τ. The* configuration *$C_{\mathcal{S}}(\tau, t)$ of $\mathcal{S}$ at time t is the n-tuple $(e_1(t), e_2(t), \ldots, e_n(t))$. If $t < O_{\max}$ then $C_{\mathcal{S}}(\tau, t)$ is undefined.*

According to Definitions 2 and 3, it holds $\forall \tau_i, 1 \leq i \leq n$, and time-instant $t \geq O_i$ that $0 \leq e_i(t) \leq C_i$ and $C_{\mathcal{S}}(\tau, t)$ is defined if and only if $t \geq O_{\max}$. If $t \geq O_{\max}$ and $t' \geq O_{\max}$ are two time-instants such that $t \neq t'$, then we denote by $C_{\mathcal{S}}(\tau, t) \succeq C_{\mathcal{S}}(\tau, t')$ the fact that $e_i(t) \geq e_i(t'), \forall i \in [1, n]$.

We assume that the tasks are independent, i.e., there is no communication, no precedence constraint and no shared resource (besides the cores) between them. Also, we assume that parallel execution of jobs is forbidden at run-time, i.e., no job can execute on more than one core at a time.

Regarding the jobs, a job $\tau_{i,j}$ is said to be *active* at time t if and only if $r_{i,j} \leq t$ and $\tau_{i,j}$ is not completed yet, i.e., $e_i(t) < C_i$. Further, an active job is said to be *running* at time t if it has been allocated to a core and is being executed. Otherwise, the active job is said to be *ready* and is pending in the ready queue of the operating system.

Platform specifications. We consider a multicore platform $\pi \stackrel{\text{def}}{=} \{\pi_1, \pi_2, \ldots, \pi_m\}$ comprising m *homogeneous* cores,

[2]It must be noted that some NP problems can be solved both exactly and "most of the time" efficiently. SAT-solvers [5,6] are a good example used in real applications such as package dependency solving in some Linux distributions.

where "homogeneous" means that all the cores have the same computational capabilities and are interchangeable. In the literature, homogeneous platforms are particular case of uniform platforms where all the cores have the same speed, and uniform cores are particular case of unrelated platforms where the execution rate $s_{i,j}$ of each core π_j is the same for all the tasks τ_i executing on this core. This inclusive relation between these models will aid us later in this work.

Scheduler specifications. We consider a fully-preemptive scheduling scheme in which a running job can be interrupted at *any* discrete time-instant and have its execution resumed later on the same core as, or a different core from, the one on which it was executing prior to the interruption.

We consider that the tasks are globally scheduled by using a FJP scheduler (e.g., global-EDF). Formally, a Fixed Job-level Priority (FJP) scheduler assigns a constant priority to each job upon its release and different jobs of the same task may have different priorities. From now on, we always assume an implementation of a FJP scheduler which is *deterministic*, *work-conserving* and *request-dependent* according to the definitions given in [9]. Informally speaking, these three requirements ensure a periodic schedule, in which the same total priority-order is used between jobs within each hyper-period.

LEMMA 1 (PERIODICITY – FROM THEOREM 3 IN [9]). *Let A denote any preemptive* FJP *and request-dependent scheduling algorithm. For any asynchronous constrained deadline task set τ which is guaranteed to meet all the deadlines when scheduled by A on m homogeneous cores, the schedule is periodic with a period of length P.*

From the statement of Lemma 1, the open question that needs to be addressed is the following: *When does the periodicity of the schedule start*? Providing an answer to this question will clearly help us in determining the duration of the simulation process on the one hand and decide on the schedulability of the system on the other hand.

3. PRIMARY RESULTS

Before we present the main contribution of this paper, we shall introduce the following definitions and results. First, let us introduce the concepts of "*sustainability*" (as presented in [12] and later generalized in [13]), "*sustainability with respect to the execution requirement parameters (sust-C)*", and the notion of worst-case instance denoted by τ^{worst}.

DEFINITION 4 (SUSTAINABLE TEST [12]). *A schedulability test for a scheduling policy is* sustainable *if any task set deemed schedulable by the test remains schedulable when the parameters of one or more individual job[s] are changed in any, some, or all of the following ways: (i) decreased execution requirements; (ii) later arrival times; (iii) smaller jitter*[3]*; and (iv) larger relative deadlines.*

DEFINITION 5 (SUST-C). *A scheduler $\mathcal{A}$ is said to be* sustainable[4] *(with respect to execution requirements) if the $\mathcal{A}$-feasibility*[5] *of a set of tasks implies the $\mathcal{A}$-feasibility of another set of tasks with identical release times, relative deadlines and periods, but* smaller *execution requirements.*

[3]The time that elapses between the arrival of the job at the core and the earliest instant at which the job may start executing.

[4]Sometimes this property is referred to as *predictability* in the literature.

[5]All tasks meet all their deadlines when scheduled using $\mathcal{A}$.

LEMMA 2 (SUSTAINABILITY [14]). *Any work-conserving and* FJP *scheduler is* sustainable *(with respect to execution requirements) upon homogeneous multicore platforms.*

Let τ^{worst} be the instance of τ in which all jobs execute for their WCETs. According to Definition 5, if τ^{worst} is schedulable on the targeted platform then any other instance of τ in which jobs execute for less than their WCETs will also be successfully scheduled on this platform.

Since homogeneous platforms are particular case of unrelated platforms where the execution rates $s_{i,j}$ are identical for all cores π_j and all tasks τ_i, we can take advantage of the following lemma taken from [9].

LEMMA 3 (MONOTONICITY – FROM LEMMA 2 IN [9]). *For any preemptive,* FJP *and request-dependent algorithm A and any asynchronous arbitrary deadline*[6] *task set τ on m unrelated cores, we have: for each task τ_i and for any time-instant $t \geq O_{\max}+P$, if no deadline is missed within $[t-P, t)$ then*

$$e_i(t) \leq e_i(t-P) \tag{1}$$

COROLLARY 1. *If a task set τ is schedulable on a given platform π then the process of simulating the schedule S^{worst} of the instance τ^{worst} will always reach a time-instant t at which $C_{\mathcal{S}^{\text{worst}}}(\tau, t) = C_{\mathcal{S}^{\text{worst}}}(\tau, t-P)$.*

PROOF. From Definition 2 (execution status), we know that in any schedule it holds for each task τ_i and for all time-instants $t \geq O_i$ that $0 \leq e_i(t) \leq C_i$. From Lemma 3, we also know that *at each multiple of the hyper-period*, it holds true that either

1. $C_{\mathcal{S}}(\tau, t) = C_{\mathcal{S}}(\tau, t-P)$: all the task execution statuses are identical to the ones defined one hyper-period earlier, i.e., $e_i(t) = e_i(t-P)$, $\forall \tau_i$, and thus the entire schedule within $[t-P, t]$ starts repeating from this time-instant t onward, or
2. $C_{\mathcal{S}}(\tau, t) \prec C_{\mathcal{S}}(\tau, t-P)$: at least one of the execution statuses is lower than the one defined one hyper-period earlier, i.e., there exists at least one task τ_i for which $e_i(t) < e_i(t-P)$.

As a consequence of this corollary, starting from any time-instant $t \geq O_{\max}$ and iterating from one hyper-period to the next one (i.e., $\forall t' = t + k \times P$ with $k \in \mathbb{N}^+$), the system configuration $C_{\mathcal{S}}(\tau, t')$ does not "increase" in the sense that none of its components $e_i(t')$ increases. Hence, $C_{\mathcal{S}}(\tau, t')$ can be seen as a counter for which at each multiple of the hyper-period, either all its components $e_i(t')$ are identical to the $e_i(t'-P)$ determined one hyper-period earlier (and the schedule starts repeating from this instant t' onward), or at least one of its components is decremented, i.e., $\exists j \in [1, n]$ such that $e_j(t') < e_j(t'-P)$. Therefore, since the task execution statuses are non-negative integers (and because the task set is assumed to be schedulable), we know that the schedule will eventually reach a time-instant $t' = t + k \times P$ with $k \in \mathbb{N}^+$ such that $e_i(t') = e_i(t'-P)$, $\forall \tau_i$. □

From this corollary, we can easily derive a first cyclic feasibility interval.

LEMMA 4. *A cyclic feasibility interval is given by*

$$\mathrm{I}_{\text{strfwd}} \stackrel{\text{def}}{=} \left[0, O_{\max} + \left(\sum_{i=1}^{n} C_i + 1\right) \times P\right] \tag{2}$$

[6]There is no constraint on the relative deadline of each task. It might be either smaller than, equal to or larger than the task's period.

PROOF. From Corollary 1, starting from $O_{\max}$ (which is the earliest time-instant in any schedule at which all the execution statuses are defined), the longest time it may take to reach an instant t' such that $e_i(t') = e_i(t'-P)$ $(\forall \tau_i)$ is given by the following scenario: the configuration/counter $C_{\mathcal{S}}(\tau, t)$ at time $t = O_{\max}$ is such that $e_i(O_{\max}) = C_i$ $(\forall \tau_i)$ and at each multiple of the hyper-period, only one task execution status is decremented. Under this scenario, it takes $\sum_{i=1}^{n} C_i$ hyper-periods to reach the configuration $(0, 0, \ldots, 0)$ after which the schedule has to repeat (assuming that the system is schedulable). Consequently, if no deadline has been missed until time $O_{\max} + \left(\sum_{i=1}^{n} C_i\right) \times P + P$, we can safely conclude on the system schedulability. The term "$+P$" stems from the fact that, once the configuration $(0, \ldots, 0)$ has been reached, the simulation process requires one more hyper-period to conclude on the schedulability, because either a deadline will be missed within this last hyper-period, or the configuration $(0, \ldots, 0)$ will repeat after this hyper-period as the tasks execution statuses are non negative integers by definition. □

If no deadline is missed while simulating the schedule S^{worst} of the instance τ^{worst} *within* I_{strfwd}, then we are guaranteed that no deadline will ever be missed while scheduling S^{worst}. From Lemma 2, this latter statement implies that the schedule of any other legal instance of τ will *never* miss a deadline and finally, the whole system can be asserted schedulable. The simulation of S^{worst} provides an *exact* schedulability test, in the sense that succeeding in meeting the deadlines of all the jobs during the simulation is a *sufficient* and *necessary* condition for the system schedulability. Indeed, *sufficient*: a positive outcome guarantees that all deadlines are *always* met, and *necessary*: the failure of the test may lead to a deadline miss at some point during the execution of the system.

Since I_{strfwd} seems rather a trivial interval and simulating a schedule within such an interval may be of very high computational complexity (as shown in Section 7), we will refer to this method as the *naive* solution and we will present tighter intervals in the following sections.

4. A TIGHTER FEASIBILITY INTERVAL: THE COUNTING FACTOR $K(T)$

The whole concept underlying our feasibility analysis is based on counting the maximum number of times that the simulation process can iterate, from one hyper-period to the next one without missing any deadline, until reaching a time-instant at which *two consecutive configurations separated by P time units are identical*. In this section, we determine this number of iterations in a formal way. Specifically:

DEFINITION 6 (COUNTING FACTOR $K(t)$). *Consider a given task system τ, platform π, and scheduler A. If the schedule of τ is simulated from time $t \geq O_{\max}$, the* counting factor $K(t)$ *denotes the maximum number of hyper-periods the simulation process (starting from the configuration at time t) can iterate until it reaches a configuration at which it starts repeating, i.e. a configuration identical to the one defined P time units later.*

Given the definition of the counting factor, feasibility intervals can be written as $[0, t + K(t) \times P + P]$, for any $t \geq O_{\max}$, and our objective in the next sections is to find the time-instant t_0 which minimizes the length of this interval, i.e.,

$$t_0 = \min_{t \geq O_{\max}} \{t + K(t) \times P + P\} \tag{3}$$

As for the naive solution, the term "$+P$" stems from the fact that, once $K(t)$ is determined, the simulation process needs one extra hyper-period to conclude on the schedulability.

4.1 Computation of $K(t)$ based on the individual tasks execution status

In this subsection, we propose a first approach to compute the counting factor $K(t)$, $\forall t$, based on the execution status $e_i(t)$ of each individual task τ_i. To do so, let us start by introducing few notions and prerequisites.

DEFINITION 7 (MIN- AND MAX- EXECUTION STATUS). *For a given task τ_i and time-instant t, we respectively denote by $e_i^{\min}(t)$ and $e_i^{\max}(t)$ a* lower- *and* upper-bound *on its execution status $e_i(t)$, assuming that no deadline has been missed from time 0 to time t.*

DEFINITION 8 (MIN- AND MAX- CONFIGURATION). *For a given task set τ and time-instant t, we define the* minimum *and* maximum configurations *at time t in the schedule S of τ as follows:*

$$C_S^{\min}(\tau, t) \stackrel{\text{def}}{=} (e_1^{\min}(t), e_2^{\min}(t), \ldots, e_n^{\min}(t))$$
$$C_S^{\max}(\tau, t) \stackrel{\text{def}}{=} (e_1^{\max}(t), e_2^{\max}(t), \ldots, e_n^{\max}(t))$$

DEFINITION 9 (PERIODIC BOUND). *For any task set τ, we say that a lower-bound $e_i^{\min}(t)$ (resp. an upper-bound $e_i^{\max}(t)$) is* periodic *if and only if it holds that $e_i^{\min}(t) = e_i^{\min}(t+P)$ (resp. $e_i^{\max}(t) = e_i^{\max}(t+P)$), for all tasks $\tau_i \in \tau$ and instant $t \geq O_{\max}$ during the simulation of τ.*

In the previous section, we mentioned that the simulation process of our naive solution continues as long as no deadline is missed and no "cycle" is detected in the schedule. Given the definitions above, we can now be more precise about the duration of this process, i.e., about the value of the counting factor $K(t)$. At any time $t \geq O_{\max}$ in the schedule $\mathcal{S}^{\text{worst}}$ of τ^{worst}, we can make the following two observations:

Obs. 1) As long as no deadline is missed and no cycle is detected, the execution status $e_i(t)$ $(i \in \{0, 1, \ldots, n\})$ of *at least* one task will be decreased at each multiple of the hyper-period, starting from time-instant t (from Corollary 1).

Obs. 2) If the bounds $e_i^{\min}(t)$ and $e_i^{\max}(t)$ are periodic (i.e., if they satisfy Definition 9) then we know that at each time-instant $t' = t + k \times P$ for all $k \in \mathbb{N}^+$, the execution status $e_i(t')$ of each task τ_i will be bounded from below by $e_i^{\min}(t') = e_i^{\min}(t)$ and from above by $e_i^{\max}(t') = e_i^{\max}(t)$.

As a consequence of these two observations, starting from any time-instant $t \geq O_{\max}$ and iterating from one hyper-period to the next one (i.e., $\forall t' = t + k \times P$ with $k \in \mathbb{N}^+$), the system configuration $C_{\mathcal{S}}(\tau, t')$ converges not toward the zero-configuration $(0, 0, \ldots, 0)$ as claimed in the proof of Lemma 4, but rather towards its minimum configuration $C_S^{\min}(t)$ $(= C_S^{\min}(t'))$. By using a similar reasoning as in the proof of Lemma 4, starting from a time $t \geq O_{\max}$ the longest time it may take to reach an instant t' such that $e_i(t') = e_i(t'-P)$ $(\forall \tau_i)$ is given by the following scenario: *the configuration at time t is $C_{\mathcal{S}}(\tau, t) = C_S^{\max}(\tau, t)$ and at each multiple of the hyper-period, the execution status of only one task is decremented.* Within this scenario, it may take up to $K(t) = \left(\sum_{i=1}^{n} e_i^{\max}(t) - \sum_{i=1}^{n} e_i^{\min}(t)\right)$ hyper-periods to reach the configuration $C_S^{\min}(\tau, t)$, after which the schedule will start repeating or a deadline will be missed in the

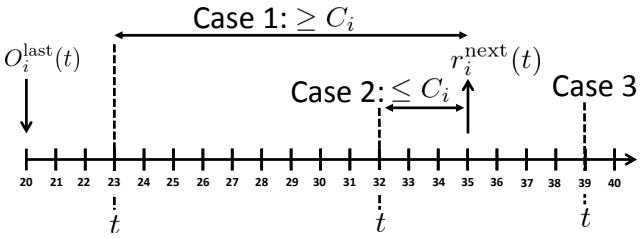

Figure 1: Illustration of the three different cases, assuming $\tau_i = \langle O_i = 0, C_i = 5, D_i = 15, T_i = 20\rangle$.

subsequent hyper-period. Since we are interested in finding the time-instant $t_0 \geq O_{\max}$ leading to the *shortest* feasibility interval, Equation (3) can be rewritten as

$$t_0 = \min_{t \geq O_{\max}} \left\{ t + \left(\sum_{i=1}^{n} e_i^{\max}(t) - \sum_{i=1}^{n} e_i^{\min}(t) \right) \times P + P \right\}$$

Moreover, if the bounds are periodic then for all instants $t \in [O_{\max}, O_{\max} + P]$ and all $t' = t + k \times P$ with $k \in \mathbb{N}$, we have $\sum_{i=1}^{n} e_i^{\max}(t) - \sum_{i=1}^{n} e_i^{\min}(t) = \sum_{i=1}^{n} e_i^{\max}(t') - \sum_{i=1}^{n} e_i^{\min}(t')$. Therefore, the instant t_0 leading to the shortest feasibility interval can be searched within $O_{\max}$ and $O_{\max} + P$ only, i.e.,

$$t_0 = \min_{O_{\max} \leq t < O_{\max}+P} \left\{ t + \left(\sum_{i=1}^{n} e_i^{\max}(t) - \sum_{i=1}^{n} e_i^{\min}(t) \right) \times P + P \right\} \quad (4)$$

It is worthwhile to understand the importance of deriving periodic bounds, as it considerably reduces the research space for t_0. Note that writing feasibility intervals as a function of lower- and upper-bounds on the task execution statuses provides a wide and general framework: determining these two periodic bounds $e_i^{\max}(t)$ and $e_i^{\min}(t)$ (for each τ_i) is a problem that can be addressed independently of the computation of a feasibility interval and it can potentially be refined and optimized for a specific scheduling algorithm. However, this work focuses on FJP schedulers in general and the following lemmas provide generic lower- and upper-bounds.

LEMMA 5. *For each task τ_i and for any time-instant $t \geq O_{\max}$, if there is no deadline missed from time 0 to time t, then $e_i(t) \leq e_i^{\max}(t)$ where*

$$e_i^{\max}(t) = \min(C_i, t - O_i^{\text{last}}(t)) \quad (5)$$

$$\textit{and} \quad O_i^{\text{last}}(t) \stackrel{\text{def}}{=} O_i + \left\lfloor \frac{t - O_i}{T_i} \right\rfloor \times T_i \quad (6)$$

PROOF. Let $O_i^{\text{last}}(t)$ denote the *last* release time of task τ_i that occurs before (or at) time t. It can easily be shown that this instant $O_i^{\text{last}}(t)$ is given by Equation (6). Besides, we know that $O_i^{\text{last}}(t)$ is defined at time t as we assumed $t \geq O_{\max}$, and thus τ_i has released at least one job (before or) at time t. In any feasible schedule, it is obvious that the execution status $e_i(t)$ of every task τ_i cannot be greater than the difference between the current instant t and the instant $O_i^{\text{last}}(t)$ of its last release. This comes from the fact that a task cannot have executed for longer than the time for which it has been released. Moreover we have $e_i(t) \leq C_i$ by the definition of an execution status. That is, for a given time-instant t and task τ_i, we have $e_i(t) \leq \min(C_i, t - O_i^{\text{last}}(t))$. □

Figure 1 illustrates Lemma 5, where the last release of task τ_i occurred at time 20 and we consider the current time to be $t = 23$. As explained in the proof above, between time-instants 20 and 23, τ_i cannot have executed for more than $23 - 20 = 3$ time units.

LEMMA 6. *The upper-bound $e_i^{\max}(t)$ defined in Lemma 5 is periodic.*

PROOF. For any two time-instants t and t' such that $t' = t + kP$, $k \in \mathbb{N}^+$, we have

$$\begin{aligned} O_i^{\text{last}}(t') &\stackrel{\text{def}}{=} O_i + \left\lfloor \frac{t' - O_i}{T_i} \right\rfloor \times T_i \\ &= O_i + \left\lfloor \frac{t - O_i}{T_i} + \frac{kP}{T_i} \right\rfloor \times T_i \\ &= O_i + \left\lfloor \frac{t - O_i}{T_i} \right\rfloor \times T_i + kP \text{ as } P \stackrel{\text{def}}{=} \text{lcm}\{T_1, T_2, \ldots, T_n\} \\ &= O_i^{\text{last}}(t) + kP \end{aligned}$$

And thus

$$\begin{aligned} e_i^{\max}(t') &\stackrel{\text{def}}{=} \min(C_i, t' - O_i^{\text{last}}(t')) \\ &= \min(C_i, t + kP - O_i^{\text{last}}(t) - kP) \\ &= \min(C_i, t - O_i^{\text{last}}(t)) \\ &= e_i^{\max}(t) \end{aligned}$$

The lemma follows. □

Let us now focus on the lower-bounds $e_i^{\min}(t)$, for all τ_i and t. Given the four parameters $\langle O_i, C_i, D_i, T_i\rangle$ of each task τ_i, there exists some methods such as the ones introduced in [15, 16] and [17] that can be used to *upper-bound* the *worst-case response time* (WCRT) — this bound is noted R_i for task τ_i in the remainder of this paper and is assumed to be known. This quantity R_i is defined as an *upper-bound* on the time that may elapse between the release of any job of τ_i and the instant it completes execution. Note that R_i can be computed directly from the four parameters of all the tasks in the system.

LEMMA 7. *For each task τ_i and for any time-instant $t \geq O_{\max}$, if there is no deadline missed from time 0 to time t, then $e_i(t) \geq e_i^{\min}(t)$ where*

$$e_i^{\min}(t) = \begin{cases} \max(0, C_i - (r_i^{\text{next}}(t) - t)) & \textit{if } r_i^{\text{next}}(t) \geq t \\ C_i & \textit{otherwise} \end{cases} \quad (7)$$

$$\textit{and} \quad r_i^{\text{next}}(t) \stackrel{\text{def}}{=} O_i + \left\lfloor \frac{t - O_i}{T_i} \right\rfloor \times T_i + R_i \quad (8)$$

PROOF. Let $r_i^{\text{next}}(t)$ denote an upper-bound on the completion time of the *last* released job of task τ_i at time t, i.e., $r_i^{\text{next}}(t) \stackrel{\text{def}}{=} O_i^{\text{last}}(t) + R_i$ where $O_i^{\text{last}}(t)$ is defined as in Lemma 5 and R_i is an upper-bound on the response time of τ_i. Depending on t, three cases may arise for a feasible schedule (see Figure 1):
Case 1: $r_i^{\text{next}}(t) \geq t$ and $r_i^{\text{next}}(t) - t \geq C_i$. Task τ_i could execute entirely after time t, hence yielding $e_i^{\min}(t) = 0$.
Case 2: $r_i^{\text{next}}(t) \geq t$ and $r_i^{\text{next}}(t) - t < C_i$. Task τ_i must have executed for at least $C_i - (r_i^{\text{next}}(t) - t)$ time units at time t, otherwise τ_i would complete after time $r_i^{\text{next}}(t)$ (which is in contradiction with the definition of $r_i^{\text{next}}(t)$). Cases 1 and 2 lead to the first piece of Expression (7).
Case 3: $r_i^{\text{next}}(t) < t$. By definition of $r_i^{\text{next}}(t)$, task τ_i has executed for C_i time units at time t.
The lemma follows. □

LEMMA 8. *The lower-bound $e_i^{\min}(t)$ defined in Lemma 7 is periodic.*

PROOF. For any two time-instants t and t' such that $t' = t + kP$, $k \in \mathbb{N}^+$, we have

$$\begin{aligned} r_i^{\text{next}}(t') &\stackrel{\text{def}}{=} O_i + \left\lfloor \frac{t' - O_i}{T_i} \right\rfloor \times T_i + R_i \\ &= O_i + \left\lfloor \frac{t - O_i}{T_i} + \frac{kP}{T_i} \right\rfloor \times T_i + R_i \\ &= O_i + \left\lfloor \frac{t - O_i}{T_i} \right\rfloor \times T_i + kP + R_i \; as \; P \stackrel{\text{def}}{=} \text{lcm}\{T_i\}_{1 \le i \le n} \\ &= r_i^{\text{next}}(t) + kP \end{aligned}$$

And thus

$$\begin{aligned} e_i^{\min}(t') &\stackrel{\text{def}}{=} \begin{cases} \max(0, C_i - (r_i^{\text{next}}(t') - t')) & \text{if } r_i^{\text{next}}(t') \ge t' \\ C_i & \text{otherwise} \end{cases} \\ &= \begin{cases} \max(0, C_i - (r_i^{\text{next}}(t) + kP - (t + kP))) & \text{if } r_i^{\text{next}}(t) \ge t \\ C_i & \text{otherwise} \end{cases} \\ &= \begin{cases} \max(0, C_i - (r_i^{\text{next}}(t) - t)) & \text{if } r_i^{\text{next}}(t) \ge t \\ C_i & \text{otherwise} \end{cases} \\ &= e_i^{\min}(t) \end{aligned}$$

The lemma follows. □

Following this computation of $e_i^{\max}(t)$ and $e_i^{\min}(t)$, if no deadline has been missed from 0 up to any time t during the simulation process, then the shortest feasibility interval is given by I_{impr} where

$$\mathrm{I}_{\text{impr}} \stackrel{\text{def}}{=} \left[0, t_{\text{impr}}^{\text{up}}\right] \tag{9}$$

and

$$t_{\text{impr}}^{\text{up}} = \min_{O_{\max} \le t < O_{\max} + P} \left\{ t + \left(\sum_{i=1}^{n} e_i^{\max}(t) - \sum_{i=1}^{n} e_i^{\min}(t) \right) \times P + P \right\} \tag{10}$$

Expression (10) introduces a significant pessimism: $K(t)$ is built upon the conservative assumption that at each multiple of the hyper-period the execution status of *only one task* is decremented. To overcome this limitation, the next subsection provides another method for deriving $K(t)$. Instead of considering the execution status of the tasks, this second method is built upon an estimation of the workload that has been executed up to time t; Thus, it takes into consideration the computing capability of the platform.

4.2 Computation of $K(t)$ based on the previously executed workload

Because the execution status $e_i(t)$ of each individual task τ_i is a *non-increasing step-case function* from one hyper-period to the next one as proven in Lemma 3, the cumulative amount of time "$\sum_{i=1}^{n} e_i(t)$" executed by *all* the tasks in the system is also a *non-increasing step-case function* from one hyper-period to the next one. As such, if $\sum_{i=1}^{n} e_i(t+k\cdot P) = \sum_{i=1}^{n} e_i(t+(k+1)\cdot P)$ for any $k \in \mathbb{N}$, then it also holds by construction that $e_i(t+k\cdot P) = e_i(t+(k+1)\cdot P)$ for each individual task τ_i. Consequently, the intuitive idea behind this second approach is to take advantage of the total computing capability provided by the platform for the computation of $K(t)$. Rather than decreasing the execution status "$e_i(t)$" of each individual task as previously, this second technique decrements the cumulative amount of time "$\sum_{i=1}^{n} e_i(t)$" executed by *all* the tasks since their last release.

DEFINITION 10 ($E(t)$). *The* cumulative *amount of time executed by* all *tasks since their last release times up to time-instant t is defined as: $E(t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} e_i(t)$ if $t \ge O_{\max}$, and $E(t)$ is undefined otherwise.*

Algorithm 1: Computation of $E^{\max}(t)$.

```
Input  : time-instant t ≥ O_max
Output: E^max(t)
{α_1, α_2, ..., α_p} ← set of p events occurring before time t ;
1  rem_budget ← α_1.wcet ;
2  cumul_budget ← rem_budget ;
3  cumul_alloc ← 0 ;
4  indic_dead ← 1 ;
5  indic_budg ← 1 ;
   foreach (j = 2 to p) do
       if (α_j.time > α_{j-1}.time) then
6          nb_proc_max ← min(m, indic_budg, indic_dead) ;
7          alloc ←
           min(rem_budget, nb_proc_max ×(α_j.time −α_{j-1}.time));
           cumul_alloc ← cumul_alloc + alloc ;
           if (cumul_alloc == cumul_budget) then
8              indic_budg ← 0;
           end
           rem_budget ← rem_budget − alloc ;
       end
       if (α_j. type == release) then
9          indic_dead ← indic_dead +1 ;
10         indic_budg ← indic_budg +1 ;
           rem_budget ← rem_budget +α_j.wcet ;
           cumul_budget ← cumul_budget +α_j.wcet ;
       else
11         indic_dead ← indic_dead −1;
       end
   end
12 nb_proc_max ← min(m, indic_budg, indic_dead) ;
   alloc ← min(rem_budget, nb_proc_max ×(t − α_p.time));
13 cumul_alloc ← cumul_alloc + alloc ;
   return cumul_alloc ;
```

COROLLARY 2 (FROM LEMMA 3 — MONOTONICITY). *For any preemptive, FJP and request-dependent algorithm A and any asynchronous constrained-deadline task set τ on m homogeneous cores, we have: for any time-instant $t \ge O_{\max}$, if there is no deadline missed up to time $t + P$, then $E(t) \ge E(t + P)$.*

PROOF. The proof is a direct consequence of Lemma 3. Since $\forall t \ge O_{\max}$ and $\forall \tau_i$, we have $e_i(t) \ge e_i(t + P)$, it obviously holds that $\sum_{i=1}^{n} e_i(t) \ge \sum_{i=1}^{n} e_i(t + P)$. □

Given a current time-instant t, we attach to each task τ_i two different "*events*": (1) one occurring at the time of its last job release and (2) another one at the deadline of this last released job. Among the $2n$ events obtained, we keep only those occurring before time t and we denote the j^{th} *task event* by α_j. Based on this notion of task event, Algorithm 1 provides an *upper-bound* on $E(t)$ by performing some computation at each *task events* occurring *before time* t. We use the following notations:

- "p" is the number of task events occurring *before* t,
- "α_j.time" is the time at which the event α_j occurs,
- "α_j.type" is the type of the event (release or deadline),
- "α_j.wcet" is the WCET of the task generating α_j.

The task events are sorted by non-decreasing order of occurring time, i.e., $\forall j \in [2, p]$: $\alpha_{j-1}.\text{time} \le \alpha_j.\text{time}$. By definition, there are at most 2 events per task and the first event is always of type " job release". Algorithm 1 goes through each event, from α_1 to α_p and finally from α_p to time t, and its objective is to schedule as many execution units as possible within each time interval, using only the workload of the last released job of each task at time t. However, Algorithm 1 sometimes *allows* job parallelism while scheduling the jobs, and the amount of work that it executes within $[\alpha_1.\text{time}, t]$ is therefore always greater than or equal to the amount of work that can actually be executed *without* job parallelism, hence providing an upper-bound on $E(t)$.

The variables of this algorithm have the following interpretation (assuming that the algorithm is at the j^{th} iteration with $j \in [2, p]$, i.e., at time α_j.time):

"cumul_alloc" records the amount of execution units that have already been *executed* from time α_1.time to time α_j.time. This quantity is initially set to 0 at line 3 and is returned at the end of the algorithm as $E^{\max}(t)$.

"rem_budget" records the amount of execution units *available* to be scheduled within $[\alpha_{j-1}.\text{time}, \alpha_j.\text{time}]$. This variable is initially set to α_1.wcet (at line 1) since the very first event is a job release and thus α_1.wcet execution units can be scheduled within $[\alpha_1.\text{time}, \alpha_2.\text{time}]$.

"cumul_budget" records the amount of execution units that have been *available* to be scheduled from the beginning, i.e., from α_1.time, to the current time α_j.time. This variable is initially set to rem_budget at line 2.

"indic_dead" and "indic_budg" are two different indicators that provide an upper-bound on the number of jobs active at time α_j.time. "indic_dead" is based on the number of releases and deadlines that occurred from time α_1.time to time α_j.time. It is initially set to 1 at line 4 (to count the release occurring at time α_1.time) and then, it is *incremented* on each job release (line 9) and *decremented* on each job deadline (line 11). "indic_budg" is based on the difference between the amount cumul_budget of execution units that have been *available* to be scheduled up to the current time α_j.time, and the amount cumul_alloc of execution units that have been *executed* by the algorithm up to that time. As for indic_dead, the indicator indic_budg is incremented on each job release (line 10), but it is never decremented, rather it is reset to 0 whenever all the execution units available for execution have been scheduled, i.e., whenever cumul_alloc = cumul_budget (line 8). Note that indic_budg gives an upper-bound on the number of active jobs *only* under the assumption that job parallelism is allowed.

"nb_proc_max" is an upper-bound on the number of cores that can be used within $[\alpha_{j-1}.\text{time}, \alpha_j.\text{time}]$. This upper-bound is computed as the minimum between the number m of cores, and the maximum number of jobs that can be active at time α_{j-1}.time, that is, $\min(\text{indic\_dead}, \text{indic\_budg})$ (line 6).

"alloc" records the amount of execution units that the algorithm schedules in $[\alpha_{j-1}.\text{time}, \alpha_j.\text{time}]$, considering only the last released job of each task at time t. This quantity is computed at line 7 as the minimum between

1. the amount rem_budget of execution units available to be scheduled in $[\alpha_{j-1}.\text{time}, \alpha_j.\text{time}]$, and
2. the maximum amount of execution units that can be scheduled in that interval, which is given by $\text{nb\_proc\_max} \times (\alpha_j.\text{time} - \alpha_{j-1}.\text{time})$.

LEMMA 9. *For any preemptive,* FJP *and request-dependent scheduler A, and for any periodic, asynchronous constrained-deadline task set τ scheduled by A upon m homogeneous cores, we have: for any time-instant $t \geq O_{\max}$, if there is no deadline miss up to time t, then $E(t) \leq E^{\max}(t)$ where $E^{\max}(t)$ is obtained by Algorithm 1.*

PROOF. The lemma is obtained from the construction of Algorithm 1, and from the pessimism that it introduces during this computation of alloc. □

EXAMPLE 1. *Consider the four tasks $\tau_i = \langle O_i, C_i, D_i, T_i \rangle$ with characteristics: $\tau_1 = \langle 9, 9, 20, 20 \rangle$, $\tau_2 = \langle 5, 5, 7, 20 \rangle$, $\tau_3 = \langle 3, 3, 6, 20 \rangle$ and $\tau_4 = \langle 0, 4, 8, 20 \rangle$. Let us assume that these four tasks are scheduled on two homogeneous cores π_1 and π_2 and let us compute $E^{\max}(15)$ using Algorithm 1. Figure 2 depicts the schedule which is assumed by the algorithm. The up and down arrows represent the release and deadline of every task, respectively. As we can see, job parallelism is permitted in the computed schedule, hence bringing some pessimism in the computation of $E^{\max}(15)$. However, the parallelism is sometimes avoided thanks to the indicators "*indic_budg*" and "*indic_dead*". For example, when $\tau_{2,1}$ is released at time 5,* indic_budg *is reset*[7] *to 0 at line 21 and then immediately incremented to 1 at line 29 to count the release of $\tau_{2,1}$. This operation prevents $\tau_{2,1}$ from being executed on more than one core during the time interval $[5, 9]$. Analogously at time 12, the indicator "*indic_dead*" prevents $\tau_{1,1}$ from being executed on multiple cores, because the difference between the number of releases and the number of deadlines that occur in the time interval $[0, 12]$ is $3 - 2 = 1$ and the algorithm deduces from this information that there cannot be more than 1 active job at that time.*

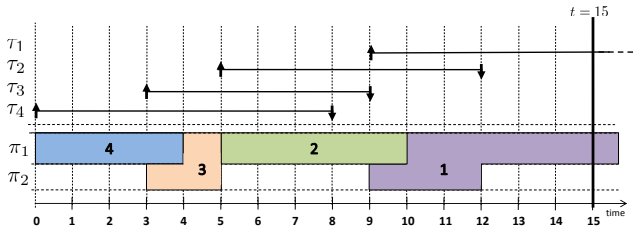

Figure 2: Schedule computed by Algorithm 1, considering the task set of Example 1.

Algorithm 2 provides a *lower-bound* on $E(t)$. The main idea to minimize the workload executed before time t is to maximize the workload executed after time t. Once this maximum is determined, the searched minimum can then be obtained by subtracting the obtained maximum from the total workload $\sum_{i=1}^{n} C_i$. As such, the problem of finding the minimum workload executed before time t can be reduced to the reverse problem of finding the maximum workload executed after time t. This is why Algorithm 2 proceeds almost the same way as Algorithm 1, except that it iterates backward, from the latest event back to time t. The main difference between Algorithms 1 and 2 resides in the task events. Firstly, the task events correspond only to job deadlines as the last job release of each task occurs *by definition* before (or at) time t. That is, only deadline-type events are considered, which in turn further simplifies the algorithm since the maximum number of potentially active jobs can be estimated only by using the indicator indic_budg. Secondly, the set $\{\alpha_1, \alpha_2, \ldots, \alpha_p\}$ is sorted by decreasing order of occurring time, i.e., $\forall j \in [1, p-1]$: $\alpha_j.\text{time} \geq \alpha_{j+1}.\text{time}$.

LEMMA 10. *For any preemptive, FJP and request-dependent scheduler A, and for any periodic and asynchronous constrained-deadline task set τ scheduled by A on m homogeneous* CPU*s, we have: for any time-instant $t \geq O_{\max}$, if there is no deadline missed up to the latest deadline of all the jobs released prior to time t, then $E(t) \geq E^{\min}(t)$ where $E^{\min}(t)$ is obtained using Algorithm 2.*

PROOF. Algorithm 2 is similar to Algorithm 1 and is safe-by-construction. □

[7] because the amount cumul_alloc of execution units that has been scheduled at time 5 is equal to the amount cumul_budget of work generated since time $\alpha_1.\text{time} = 0$.

Algorithm 2: Computation of $E^{\min}(t)$.

Input : time-instant $t \geq O_{\max}$
Output: $E^{\min}(t)$
$\{\alpha_1, \alpha_2, \ldots, \alpha_p\} \leftarrow$ set of p events (deadlines) occurring *after* time t ;
1 rem_budget $\leftarrow \alpha_1$.wcet ;
2 cumul_budget $\leftarrow$ rem_budget ;
3 cumul_alloc $\leftarrow 0$;
4 indic_budg $\leftarrow 1$;
foreach *($j = 2$ up to p)* **do**
  alloc $\leftarrow$ $\min(\text{rem\_budget}, \min(m, \text{indic\_budg}) \times (\alpha_{j-1}.\text{time} - \alpha_j.\text{time})$;
  cumul_alloc $\leftarrow$ cumul_alloc + alloc ;
  rem_budget $\leftarrow$ rem_budget − alloc ;
  if *(*cumul_alloc == cumul_budget*)* **then** indic_budg $\leftarrow 0$;
  rem_budget $\leftarrow$ rem_budget $+\alpha_j$.wcet ;
  cumul_budget $\leftarrow$ cumul_budget $+\alpha_j$.wcet ;
  indic_budg $\leftarrow$ indic_budg $+1$;
end
alloc $\leftarrow \min(\text{rem\_budget}, \min(m, \text{indic\_budg}) \times (\alpha_p.\text{time} - t)$;
cumul_alloc $\leftarrow$ cumul_alloc + alloc ;
return $\left(\sum_{j=1}^{n} C_i - \text{cumul\_alloc}\right)$;

As a conclusion, if no deadline has been missed until the latest deadline of all the jobs released prior to time t then we can safely conclude on the system schedulability if no deadline is missed until time $t + K_2(t) \times P + P$ where

$$K_2(t) \stackrel{\text{def}}{=} \left(E^{\max}(t) - E^{\min}(t)\right) \tag{11}$$

Finally, since the tasks are periodic they have the same release pattern at every multiple of the hyper-period. As a consequence Algorithm 1 produces the same output at every time $t' = t + kP$, for any $t \geq O_{\max}$ and $k \in \mathbb{N}$ (and the same is true for Algorithm 2). That is, Algorithms 1 and 2 provide periodic bounds and w.r.t. Equation (3), the shortest feasibility interval is given by $\mathrm{I}_{\text{impr2}}$ where

$$\mathrm{I}_{\text{impr2}} \stackrel{\text{def}}{=} \left[0, t^{\text{up}}_{\text{impr2}}\right] \tag{12}$$

and

$$t^{\text{up}}_{\text{impr2}} = \min_{O_{\max} \leq t < O_{\max}+P} \left\{t + \left(E^{\max}(t) - E^{\min}(t)\right) P + P\right\} \tag{13}$$

5. SOME TRICKS TO FURTHER TIGHTEN THE FEASIBILITY INTERVAL

A simple way to derive the shortest feasibility interval from all the methods presented in this paper is to combine all these results together. Such a combination results in the following interval:

$$t^{\text{up}}_{\text{best}} = \min_{O_{\max} \leq t < O_{\max}+P} \{t + (\mathrm{UB}(t) - \mathrm{LB}(t))\, P + P\} \tag{14}$$

where $(\mathrm{UB}(t) - \mathrm{LB}(t))$ is the counting factor at time t and:

$$\mathrm{UB}(t) \stackrel{\text{def}}{=} \min\left(E^{\max}(t), \sum_{i=1}^{n} e_i^{\max}(t)\right) \tag{15}$$

$$\mathrm{LB}(t) \stackrel{\text{def}}{=} \max\left(E^{\min}(t), \sum_{i=1}^{n} e_i^{\min}(t)\right) \tag{16}$$

Since the four terms "$e_i^{\min}(t)$", "$e_i^{\max}(t)$", "$E^{\min}(t)$", and "$E^{\max}(t)$" are computed based on the task parameters, their order of magnitude is likely to be the same as the order of magnitude of these parameters. This can be observed in the following example. Let us consider a two-processors platform and a task set τ composed of three tasks whose parameters are listed in Table 1. The schedule of these three tasks by Global-EDF is depicted in Figure 3 from time 0 to 300. From that picture, the schedule will repeat from time $O_{\max} + P = 290$ onward. That is, the schedule repeats after one hyper-period, starting from $O_{\max}$.

Task set τ					
	O_i	C_i	D_i	T_i	R_i
τ_1	50	90	120	120	100
τ_2	30	60	80	80	70
τ_3	0	10	120	120	100

$\Leftrightarrow$

Task set τ'					
	O'_i	C'_i	D'_i	T'_i	R'_i
τ_1	5	9	12	12	10
τ_2	3	6	8	8	7
τ_3	0	1	12	12	10

Table 1: Parameters of equivalent task sets τ and τ'

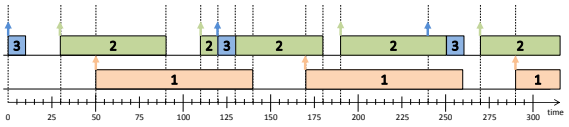

Figure 3: Schedule of τ from time 0 to 300 on a two-processors platform, using G-EDF.

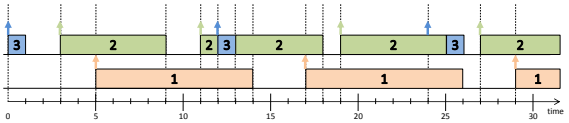

Figure 4: Schedule of τ' from time 0 to 30 on a two-processors platform, using G-EDF.

By computing the feasibility interval given by Equation (9), it can be showed that the value of $t^{\text{up}}_{\text{impr}}$ that minimizes the length of the interval is obtained at time $t = 100$, with $e_2^{\min}(100) = e_2^{\max}(100) = C_2 = 60$, $e_3^{\min}(100) = e_3^{\max}(100) = C_3 = 10$, and $e_1^{\min}(100) = 40$ and $e_1^{\max}(100) = 50$. The counting factor $K(t)$ at that time $t = 100$ is

$$K(100) = \sum_{i=1}^{n} e_i^{\max}(100) - \sum_{i=1}^{n} e_i^{\min}(100) = 120 - 110 = 10 \tag{17}$$

As we can observe, the task parameters are all multiples of ten, and so are the terms $e_i^{\min}(100)$, $e_i^{\max}(100)$ and $K(100)$. This counting factor leads to $t^{\text{up}}_{\text{impr}} = t + K(t) \times P + P = 100 + 10 \times 240 + 240 = 2740$, and thus the schedulability of the task set τ can be asserted by constructing its schedule from time 0 up to $100 + 11 \times P$.

A second trick to reduce the length of the feasibility interval is to replace the given task set τ for an *equivalent* task set τ' with smaller task parameters. Two task sets τ and τ' are said to be *equivalent* if and only if the schedulability of either of them can be deduced from the schedulability of the other. *Considering FJP schedulers*, it can easily be shown that for all $k \in \mathbb{R}^+$, multiplying all the task parameters by k results in an equivalent task set.

To illustrate that claim, let us consider the task set τ' derived from τ by multiplying all the task parameters by 1/10 (see right table of Table 1). The response time of the tasks become $R'_1 = 10$, $R'_2 = 7$, and $R'_3 = 10$. The schedule of these three tasks is depicted in Figure 4 and we can see that this schedule is exactly the same as the schedule of τ, except that it is shrunk by a factor 10 (hence the equivalence of the two task sets). In the schedule of τ', the value of $t^{\text{up}}_{\text{impr}}$ that minimizes the length of the feasibility interval is obtained at $t = 10$, with $e_2^{\min}(10) = e_2^{\max}(10) = C'_2 = 6$, $e_3^{\min}(10) = e_3^{\max}(10) = C'_3 = 1$, and $e_1^{\min}(10) = 4$ and

$e_1^{\max}(10) = 5$. The counting factor $K'(10)$ is then given by:

$$K'(10) = \sum_{i=1}^{n} e_i^{\max}(10) - \sum_{i=1}^{n} e_i^{\min}(10) = 12 - 11 = 1 = \frac{K(100)}{10} \quad (18)$$

which leads to $t_{\text{impr}}^{\text{up}} = t + K'(t) \times P' + P' = 10 + 1 \times 24 + 24 = 58$. Consequently, the schedulability of τ' (and thus the schedulability of τ) can be asserted by simulating the schedule of τ' from time 0 to $10 + 2 \times P$.

As a general rule, the smaller the task parameters, the shorter the feasibility interval. Therefore, an efficient trick is to first divide the parameters of all the tasks by the greatest common divisor of those parameters.

THEOREM 1 (EXACT SCHEDULABILLITY TEST). *System τ is schedulable if and only if all the job deadlines of τ^{worst} are met in $[0, t_{\text{best}}^{\text{up}}]$, where $t_{\text{best}}^{\text{up}}$ is defined as in Equation* (14).

PROOF. If a deadline is missed in $[0, t_{\text{best}}^{\text{up}}]$, then the system is clearly not schedulable. Otherwise, the schedule of τ^{worst} has passed by all possible system configurations, plus one, without any deadline miss. Hence τ is schedulable. □

6. COMPUTATIONAL COMPLEXITY

In this section we distinguish between (1) the time complexity of computing the length of the feasibility interval $[0, t_{\text{best}}^{\text{up}}]$, and (2) the time complexity of the exact analysis this interval allows for.

The computation of the feasibility interval $[0, t_{\text{best}}^{\text{up}}]$ involves computing $e_i^{\min}(t)$, $e_i^{\max}(t)$, $E^{\min}(t)$, and $E^{\max}(t)$ at all time-instants $t \in [O_{\max}, O_{\max}+P]$ and for all tasks $\tau_i \in \tau$. From Expressions (5) and (7), the computation of $e_i^{\min}(t)$ and $e_i^{\max}(t)$ takes $O(1)$, assuming that R_i is computed beforehand, and from Algorithm 1 and 2 the computation of $E^{\min}(t)$ and $E^{\max}(t)$ takes $O(n \log(n))$ – the computation is linear in $p \leq 2n$ and takes $O(p \log(p))$ due to the sorting of the set of events. Therefore, the overall complexity of computing $t_{\text{best}}^{\text{up}}$ is $O(P \times (n + n + n\log(n) + n\log(n))) = O(P \times n \log(n))$.

Regarding (2), as mentioned in Theorem 1, an exact schedulability test for a given task set τ consists in simulating the execution of its instance τ^{worst} within the interval $[0, t_{\text{best}}^{\text{up}}]$. Since $t_{\text{best}}^{\text{up}}$ is computed based on P and $P \stackrel{\text{def}}{=} \text{lcm}\{T_i\}_{1 \leq i \leq n}$, it follows that the computing complexity of an exact schedulability analysis is a function of the least common multiple of the task periods. Specifically, the number of hyper-periods the simulator must scrutinize is given by the counting factor K (obtained during the computation of $t_{\text{best}}^{\text{up}}$). Since the number of scheduling decisions taken within an hyper-period P is given by $2 \times \sum_{\tau_i \in \tau} \frac{P}{T_i}$ (one decision at each job arrival and another one at each job completion), the total amount of scheduling decisions taken during the simulation is $2K \times \sum_{\tau_i \in \tau} \frac{P}{T_i}$, resulting in a time-complexity of $O(K \times \sum_{\tau_i \in \tau} \frac{P}{T_i})$.

Based on this complexity, one can decide either to use existing sufficient schedulability tests (see [7]) or to simulate the schedule of τ^{worst} if the simulation time appears to be reasonable.

7. SIMULATION RESULTS

This section reports on the lengths of the feasibility intervals obtained in this paper. For simplicity, our simulations are carried out by using periodic and asynchronous *implicit-deadline*[8] tasks and comparisons are performed by simulating the execution of thousands of task sets scheduled by global-EDF. For each generated task set τ, we computed the exact feasibility interval (see below) and the length of the tightest interval $[0, t_{\text{best}}^{\text{up}}]$ that we obtained from Equation (14). Each task set is generated as explained below.

Task generation process: The input to the task-set generator is: a number m of processors, a minimum task utilization $U_{\min}$, a maximum task utilization $U_{\max}$, and a targeted total utilization U_{sum}. Given these inputs, utilization values u_i are uniformly generated within $[U_{\min}, U_{\max}]$ until their sum becomes $\geq U_{\text{sum}} - U_{\max}$. Once this threshold is exceeded, a last task is generated with utilization equal to $U_{\text{sum}} - \sum u_i$. Then, the period of the tasks are generated in such a way that the hyper-period is kept "reasonably small". This was necessary since the method used to derive the exact (i.e. smallest) length of a feasibility interval is highly computationally intensive and its complexity depends on the least common multiple of the task periods. Each task period is generated as follows: we randomly chose a number $a_i \in \{2, 4, 8, 16\}$, a second number $b_i \in \{3, 6, 9, 12\}$ and a third number $c_i \in \{5, 10, 15\}$; The period T_i is then set to $a_i \times b_i \times c_i$. Finally, the WCET of each task is computed based on its utilization and period, $C_i = u_i \times T_i$ (it is rounded up to 1 if ≤ 1) and its offset O_i is uniformly generated within $[1, T_i]$.

Computation of the exact interval $[0, t_{\text{exact}}]$: For a given task set τ, we simulate the schedule of τ and we record and store the system configuration $C_{\mathcal{S}}(\tau, t)$ at each and every time-instant t, starting from $t = O_{\max}$. Once $t \geq O_{\max} + P$, every configuration $C_{\mathcal{S}}(\tau, t)$ recorded at time t is further compared against the one recorded at time $t - P$. While $C_{\mathcal{S}}(\tau, t) \neq C_{\mathcal{S}}(\tau, t - P)$, the simulation process continues; In this case the configuration $C_{\mathcal{S}}(\tau, t - P)$ is dropped out (as it is no longer useful) and the configuration $C_{\mathcal{S}}(\tau, t)$ is stored. The simulation process is stopped once $C_{\mathcal{S}}(\tau, t) = C_{\mathcal{S}}(\tau, t - P)$ and $t_{\text{exact}} = t$ is returned. For each generated system τ, we divided the length t_{exact} of the exact interval by $O_{\max} + P$ before plotting the resulting value. We did so in order to depict only values that are not proportional to the length of the hyper-period and the maximum task offset.

We ran several sets of experiments with different values of the four input parameters: $m = 4$, 8, and 16, $U_{\min} = 0.01$, and $U_{\max} = 0.1$, 0.5, and 1. For all combinations of these parameters, we made U_{sum} varying from 0.1 to m with a step of 0.1. Due to the space limitation, all the figures presented here use the parameters $m = 8$, $U_{\min} = 0.01$, $U_{\max} = 1$, and U_{sum} varies from 0.1 to m by step of 0.1.

Experiment 1 (Figure 5): As it can be seen, the interval $[0, t_{\text{best}}^{\text{up}}]$ has the same length as the exact interval for a total utilization less than 3. Beyond 3, the accuracy of $[0, t_{\text{best}}^{\text{up}}]$ starts deteriorating and the our interval is no longer tight. Note that the plotted values for $t_{\text{best}}^{\text{up}}$ in Figure 5 are not the absolute values of $t_{\text{best}}^{\text{up}}$; Rather, it is the values of $t_{\text{best}}^{\text{up}}$ divided by the length of the corresponding exact interval (and this applies to all the other results that are presented in this section). The rational for applying this transformation is that the ratio between the lengths of the intervals is a much more meaningful information. Also, given that the resulting values for the exact interval seem to always be equal to one, it might be thought that the length of this interval is always equal to $O_{\max} + P$. However, there are few cases for which it is not true! (not so many cases though).

Another misinterpretation of the results of Figure 5 is to claim that, since the exact interval is most of the time

[8]For each task $\tau_i = \langle O_i, C_i, D_i, T_i \rangle$ we have $D_i = T_i$.

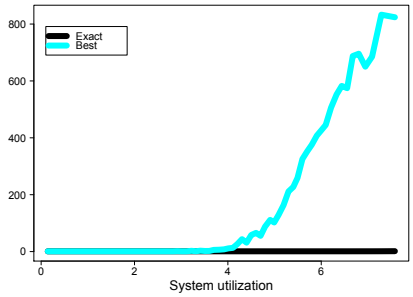

Figure 5: Comparison between our best feasibility interval and the exact one.

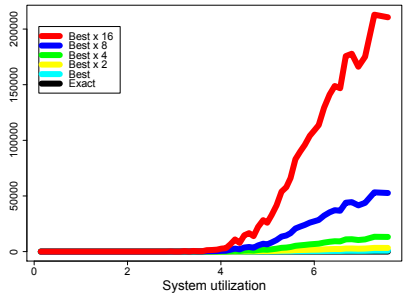

Figure 6: Illustration of the efficiency of the "gcd" trick.

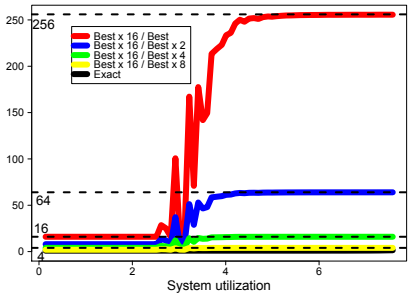

Figure 7: Interesting tendency observed by using of the "gcd" trick.

of length $O_{\max} + P$, it is a wiser option to compute this exact interval rather than an over-approximated one (like $t^{\text{up}}_{\text{best}}$). Here again, our experiments revealed that this intuition is wrong. Computing the exact interval may be extremely memory-intensive as the system configurations must be recorded at every time-instant t during an entire hyper-period. This may result in an excessive number of configurations to be stored and moreover, the computation of this exact interval progresses time unit by time unit, and passing through all the system configurations within an entire hyper-period may take a non-acceptable amount of time. In contrast, though the length of our interval $[0, t^{\text{up}}_{\text{best}}]$ may be a way longer than the exact one, simulating the schedule of the system within that interval does not require to progress so slowly through the schedule. The simulation can instead progress from one "event" to the next one, where an event is either a task release or a job completion.

Experiment 2 (Figure 6): In this second set of experiments, we illustrate the benefit of using the "gcd" trick presented in Section 5. We carried out the same simulations as in Experiment 1, but from each generated task set τ we created four additional task sets τ^2, τ^4, τ^8, and τ^{16}. Each additional task set τ^x is obtained from τ by multiplying all its task parameters by x. Then, we computed the length of our feasibility interval $[0, t^{\text{up}}_{\text{best}}]$ for each of these additional task sets. The results show that: the bigger the task parameters, the longer the interval $[0, t^{\text{up}}_{\text{best}}]$. That is (the other way around), the higher the greatest common divisor of all the task parameters, the higher the benefits of using this technique.

Experiment 3 (Figure 7): We observed an interesting phenomenon while testing the "gcd" trick. Beyond a certain utilization threshold (when the displayed functions seem to stabilize after taking off, around approximately $\frac{m}{2}$), a reduction of all the task parameters by a factor x results in a feasibility interval $[0, t^{\text{up}}_{\text{best}}]$ approximately x^2 times shorter. Surprisingly, and beyond all expectations, this simple trick appeared to be a necessary step to considerably reduce the pessimism of the computed interval. So far, we have not yet found any explanation for this quadratic reduction.

8. CONCLUSION

In this paper, we proposed a feasibility interval for periodic and asynchronous constrained-deadline tasks, FJP schedulers, and homogeneous multicores. We have shown through extensive experiments that the proposed interval is as pessimistic as the utilization of the system increases. However, we propose a simple trick (referred to as the "gcd"-trick in the paper) to considerably reduce this pessimism for highly utilized systems. It is important to stress that such feasibility intervals allow for *exact* schedulability analyses.

9. REFERENCES

[1] J. Y.-T. Leung, "A new algorithm for scheduling periodic real-time tasks," *Algorithmica*, vol. 4, pp. 209–219, 1989.

[2] B. Kalyanasundaram, K. R. Pruhs, and E. Torng, "Errata: A new algorithm for scheduling periodic, real-time tasks," *Algorithmica*, vol. 28, pp. 269–270, 2000.

[3] T. P. Baker and M. Cirinei, "Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks," in *International Conference On Principles Of DIstributed Systems*, 2007, pp. 62–75.

[4] G. Geeraerts, J. Goossens, and M. Lindstrum, "Multiprocessor schedulability of arbitrary-deadline sporadic tasks: Complexity and antichain algorithm," *Real-Time Systems: The International Journal of Time-Critical Computing*, vol. 49, no. 2, pp. 171–218, 2013.

[5] C. P. Gomes, H. Kautz, A. Sabharwal, and B. Selman, "Satisfiability solvers," pp. 89–134.

[6] *Handbook of Knowledge Representation*, ser. Foundations of Artificial Intelligence. Elsevier, 2008, vol. 3.

[7] M. Bertogna and S. Baruah, "Tests for global EDF schedulability analysis," *Journal of Systems Architecture*, vol. 57, no. 5, pp. 487–497, 2011.

[8] D. Muller and M. Werner, "Genealogy of hard real-time preemptive scheduling algorithms for identical multiprocessors," *Central European Journal of Computer Science*, vol. 1, no. 3, pp. 253–265, 2011.

[9] L. Cucu-Grosjean and J. Goossens, "Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms," *Journal of Systems Architecture*, vol. 57, no. 5, pp. 561–569, 2011.

[10] P. Courbin, I. Lupu, and J. Goossens, "Scheduling of hard real-time multi-phase multi-thread periodic tasks," *Real-Time Systems: The International Journal of Time-Critical Computing*, 2013.

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

[12] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *27th IEEE International Real-Time Systems Symposium*, 2006, pp. 159–168.

[13] A. Burns and S. Baruah, "Sustainability in real-time scheduling," *Journal of Computing Science and Engineering*, vol. 2, no. 1, pp. 74–97, 2008.

[14] R. Ha and J. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems," in *14th IEEE International Conference on Distributed Computing Systems*, 1994, pp. 162–171.

[15] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *28th IEEE International Real-Time Systems Symposium*, dec. 2007, pp. 149–160.

[16] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed priority multiprocessor scheduling," in *Real-Time Systems Symposium*. 2009, pp. 387–397.

[17] T. P. Baker and M. Cirinei, "A unified analysis of global edf and fixed-task-priority schedulability of sporadic task systems on multiprocessors." *Journal of Embedded Computing*, vol. 4, no. 2, pp. 55–69, 2011.