INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Timing Analysis of Real-Time Scheduling Policies: A Trajectory Based Model

J.M. Migge, A. Jean-Marie

N° 3561

Novembre 1998

THÈME 1

Rapport de recherche

ISSN 0249-6399

Timing Analysis of Real-Time Scheduling Policies: A Trajectory Based Model

J.M. Migge, A. Jean-Marie

Thème 1 — Réseaux et systèmes
Projet Mistral

Rapport de recherche n° 3561 — Novembre 1998 — 100 pages

Abstract: Timing analysis of real time scheduling policies is concerned with the analysis of response times, because real-time constraints impose that tasks must complete before their deadlines. For the well studied *earliest deadline first* (EDF) and the *fixed priority preemptive* (FPP) policy, results are known under various assumptions about tasks. In this report we propose a mathematical model based on trajectories to represent and analyze the scheduling of recurrent tasks on one processor. We identify generic ideas and concepts such as *majorizing work arrival functions*, which allows to study scheduling policies separately from assumptions on tasks. We develop in particular a common approach for deriving response times bounds under policies that can be defined by priorities which are assigned to *instances* of tasks. Not only FPP and EDF but also the classical *first in first out* or the *last in first out* policies fall in this category. By developing the model as general framework for timing analysis we have also obtained some extensions of existing results about EDF and FPP.

Key-words: real-time scheduling, timing analysis, earliest deadline first, fixed preemptive priorities

Analyse de faisabilité sous contraintes temps-réel: Un modèle à base de trajectoires

Résumé : L'étude de politiques d'ordonnancement temps-réel concerne l'analyse de temps de réponse de tâches afin de déterminer si leurs dates d'échéance sont toujours respectées. Pour l'ordonnancement à *échéance la plus proche en premier* (EDF) et celui à *priorités préemptives fixes*, des résultats sont connus pour diverses hypothèses sur les tâches. Dans ce rapport, nous proposons un modèle mathématique basé sur le concept de trajectoire, permettant de décrire et d'analyser l'ordonnancement de tâches récurrentes sur un processeur. Nous identifions des idées et des concepts génériques comme les *fonctions d'arrivée de travail majorantes* qui permettent d'analyser les politiques de manière indépendante des hypothèses faites sur les tâches. Nous développons en particulier une approche commune pour l'analyse de politiques qui attribuent à toute instance d'une tâche une certaine priorité. Cette classe de politiques englobe non seulement EDF et FPP, mais aussi *premier arrivé, premier servi* (FIFO) et *dernier arrivé, dernier servi* (LIFO). En développant le modèle comme cadre général pour l'analyse de temps de réponse, nous avons aussi obtenu des extensions de résultats concernant EDF et FPP.

Mots-clés : ordonnancement, temps-réel, faisabilité

Contents

List of Figures

List of Tables

1 Introduction

Scheduling is concerned with the problem of organizing the execution of resource consuming jobs under constraints. In the context of real-time systems, these jobs are recurrent invocations of tasks and the main resource is a processing unit that must be shared under the constraint that all invocations of tasks complete their execution before their deadline, i.e. are *feasible*. The feasibility constraint is originated in the fact that in real-time systems a result must not only be logically correct, but also be produced within a certain time span, given by a deadline. Deadlines are supposed to be such that feasibility guarantees the safety and physical integrity of the real-world system. Typical examples are the on-board computer of an airplane or the command and control system of a chemical plant.

The aim of a *timing analysis* is to prove off-line that tasks always meet their deadlines at run-time. A naive approach would consist in attempting to simulate all possible evolutions of the system and to check the feasibility for each execution. But in general there are infinitely many different evolutions, which makes this approach only useful for estimating the average behavior of the system. On the other hand, efficient timing analysis could be seen as a kind of simulation which is aimed at testing if under worst conditions task still meet their deadlines. Ideally, the timing analysis examines only those scenarios where tasks have their worst response time. It depends however on task types and the policy how many different such scenarios exist and have to be analyzed. If tasks are periodic and the policy is based on fixed preemptive priorities, then there is exactly one scenario, the *critical instant* [22]. But if tasks have offset constraints [32] or if their execution times follow a certain pattern as for multiframe tasks [24], then the worst case response time can occur during several different scenarios and it is not possible to know in which, unless examining each of them individually. This diminishes the efficiency of the timing analysis, but satisfactory results can be obtain using conservative approximations if necessary. However, for a policy such as Dual Priorities [11], no efficient timing analysis is known because the underlying mechanism makes it difficult to determine a restricted set of scenarios where worst response times occur.

Timing analysis is usually expressed as feasibility tests, which are sufficient and/or necessary, depending on the policy and assumptions made about tasks. These assumptions define models for real-world tasks. It appears that the simpler the model compared to the actual properties of the real-world tasks, the worse these tests perform, rejecting actually feasible sets. The reason is that for real-time systems, tasks models must be build on conservative approximations of the properties of a real-world task, in order to be able to guarantee feasibility. In the literature, simple initial models have been extended little by little, to account more accurately for the properties of tasks. In this document we will somehow proceed the other way round. The idea is to construct a *mathematical model* of a processing unit that executes recurrent invocations of a set of tasks according to some scheduling policy. The aim is to provide a unified framework for deriving timing analysis.

The model is as general as possible to include virtually any kind of scheduling policy and kind of task. It is based on the concept of *point process*, see [3], which is a stochastic model for discrete event systems. We do not consider any stochastic assumptions (in this document), but use the related concept of trajectory space to model all possible evolutions of the system. By analogy, we call it *(scheduled) task process*. The basic entities are instances of tasks, with their attributes. To represent the effects of the scheduling policy, we define a *priority assignment* at the level of instances and a *scheduling function*. Together with *work arrival functions* and the *workloads*, we are able to exactly describe the state of the system at every moment and to write equations for execution ends and response times.

The model as such allows a wide range of behaviors, for which only general or generic properties can be stated. We define then in particular the class of scheduling policies determined by *time independent priority assignments*, which includes not only real-time scheduling policies such as Fixed Preemptive Priorities (FPP), and Earliest Deadline First (EDF) but also First In First Out (FIFO) and Last In First Out (LIFO). In the analysis of the class, we derive a generic result about response time bounds and the related feasibility condition, which we apply then to all four policies. Proceeding this way, we identify common ideas that characterize the timing analysis of these policies. In particular, we show that the analysis can be done independently of specific assumptions about the task, and is only based on the their characterization in terms of *majorizing work arrival functions.* In other words, we separate the analysis of task behaviors from the analysis of policies, which reduces the difficulty of deriving timing analysis. With our approach, we have also obtained some specific results in timing analysis. An overview can be found in Section 5.

The document is organized as follows. In Section 2 the basic model and related concepts are introduced. Section 3 regroups the definition, within the model, of different task types that have been considered in the literature; their characterization in terms of majorizing work arrival functions is derived. Section 4 is concerned with those scheduling policies which can be defined by time independent priority assignments. Apart from their general timing analysis in the context of the model, a subsection discusses improvements in computational complexity and tradeoffs between complexity and accuracy of response time bounds. In the last section we give an overview of the specific timing analysis results that have obtained with our approach. Furthermore we draw conclusions and give an outlook on further extensions of the model.

2 Framework

2.1 The task process

In this section we define the concept of *task process.* Its purpose is to describe the possible evolution of the demands of tasks that are to be scheduled. We also introduce the concept of work arrival function to formalize these demands. The scheduling itself is represented by the concept of *scheduled task process*, defined in the subsequent Section 2.2.

2.1.1 Recurrent tasks

The system to be modeled consists in a finite set of *recurrent tasks* $\mathcal{T} = \{\tau_1, \dots, \tau_m\}$ and a processing unit that executes the successive *instances* or *invocations* of these tasks. We denote by $\tau_{k,n}$ the n^{th} instance of task τ_k and by $A_{k,n} \in \mathbb{R}_+$, with $n \in \mathbb{N}$, the corresponding *activation* or *release* time. For each task τ_k, the arrivals are assumed to be indexed such that:

$$0 \leqslant A_{k,0} \leqslant A_{k,1} \leqslant \dots \leqslant A_{k,n-1} \leqslant A_{k,n} \leqslant A_{k,n+1} \leqslant \dots \tag{2.1}$$

i.e. in the order of their occurrence, with some given order among synchronous arrivals. The time between two successive arrivals, the *inter arrival time* or *cycle time* is denoted $T_{k,n} \overset{\mathsf{def}}{=} A_{k,n+1} - A_{k,n}$. We assume that the sequence of invocations of a task has no *point of accumulation*, i.e. for each $\tau_k \in \mathcal{T}$ there are a finite number of activations in any interval of finite length.

$$\forall\, t_1 \leqslant t_2 \in \mathbb{R}_+ \;:\; \qquad \sum_{n \in \mathbb{N}} \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} < \infty. \tag{2.2}$$

Since each task has by definition infinitely many instances, (2.2) implies that tasks really are recurrent in the sense that for any time $t \in \mathbb{R}_+$ there exists a release of each task after t.

Let us denote $\mathbb{A} \subset (\mathbb{R}_+^{\mathbb{N}})^m$ the m-dimensional set of sequences satisfying (2.1) and (2.2). With a release $A_{k,n}$ is associated its *execution time* $C_{k,n} \in \mathbb{R}_+$, the amount of work to be done to complete the instance and its *absolute deadline* $D_{k,n}$, the time before which the task should complete its execution. It is related to the activation time by the *relative deadline* denoted $\overline{D}_{k,n}$:

$$D_{k,n} \overset{\mathsf{def}}{=} A_{k,n} + \overline{D}_{k,n}.$$

It is reasonable to assume that $\overline{D}_{k,n} \geqslant 0$ because otherwise, a task may be activated after its deadline.

We call $\mathbb{T} \overset{\text{def}}{=} \mathbb{A} \times (\mathbb{R}_+^{\mathbb{N}})^m \times (\mathbb{R}_+^{\mathbb{N}})^m$ the *set of task sequences.* The elements of $\mathbb{T}$ represent all possible ways in which instances of m recurrent tasks can be activated and request the processing unit in the time interval $[0, \infty[$.

The behavior of a real-world task set is modeled by a subset of $\mathbb{T}$, i.e. a certain set of task sequences that represent all possible behaviors of the tasks. To be able to identify in $\mathbb{T}$ the task sequences that correspond to the modeled system, we introduce an index-set Ω, which might be countable or not. We use a mapping

$$(A, C, D) : \ \Omega \longrightarrow \mathbb{T} : \ \omega \longmapsto (a, c, d) \tag{2.3}$$

to pick out the concerned task sequences from $\mathbb{T}$. Notice that we denote the function by (A, C, D), to remind that it is composed of three components "activation sequences", "execution time sequences" and "deadline sequences". An element $\omega \in \Omega$ is called *trajectory* or *history* of the system. In a similar way as a random variable represents the different possible outcomes of a random experiment, (A, C, D) represents the different possible evolutions that correspond to the modeled system.

Definition 2.1 *A mapping (A, C, D) from a set Ω into the set of task sequences $\mathbb{T}$ is called a* task process.

Notice that in the terminology of *point processes*, activation times are the *points* and execution times and deadlines are *marks.*

We should for example write $A_{k,n}(\omega)$ for the n^{th} release of task τ_k on trajectory ω, but in order to avoid cumbersome notations, we will simply write $A_{k,n}$, unless making appear ω explicitly is needed.

As with random variables we write in small letters a particular realization of the task process. A task sequence in $\mathbb{T}$, that is an arrival time sequence together with an execution time sequence will be denoted (a, c, d). In order to keep notations sparse, we will avoid as far as possible to address particular realizations. Definitions and properties are stated in terms of task processes - written with capital letters - under the convention that they are meant for all trajectories. But trajectories are explicitly written where this convention leads to ambiguities or if this is required for any other reason.
Remark: Assumptions (2.1) and (2.2) are aimed to be as less restrictive as possible to ensure on one hand that the mathematical concept of *recurrent task* does indeed have the properties that one would expect it to have, and on the other hand that it is possible to define within the resulting framework the types of task that are needed to model real-world systems.

2.1.2 Work arrival functions (WAF)

The response time of an instance of a task depends on the concurrent demands of other instances, belonging to the same task or not. Demands become effective after the corresponding release time $A_{k,n}$, meaning that an instance can not be executed before that time. To represent the amount of requested execution time of an instance $\tau_{k,n}$, if there is an effective demand, we introduce

$$S_{k,n}(t_1, t_2) \overset{\text{def}}{=} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \tag{2.4}$$

for any interval $[t_1, t_2[$. We call $S_{k,n}$ the *work arrival function* (WAF for short) of an instance. It measures the amount of work due to instance $\tau_{k,n}$ and arriving during the interval $[t_1, t_2[$. It is a very simple function, that is of minor interest as such, but it serves as building brick for other functions. Let $\mathcal{I}$ be a set of instances (not necessarily belonging to the same task). The corresponding WAF is defined by

$$S_{\mathcal{I}}(t_1, t_2) \overset{\text{def}}{=} \sum_{\tau_{i,j} \in \mathcal{I}} S_{i,j}(t_1, t_2). \tag{2.5}$$

An example is the WAF of a given task τ_k, which corresponds to $\mathcal{I} = \{\tau_{k,n} \mid n \in \mathbb{N}\}$:

$$S_k(t_1, t_2) \overset{\mathsf{def}}{=} \sum_{n \in \mathbb{N}} S_{k,n}(t_1, t_2) = \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]}. \tag{2.6}$$

Notice that this function is piecewise constant, meaning that any interval of finite length can be subdivided into a finite number of subintervals on which the function is constant. This is partially due to the fact that the sequence of activation times is assumed to have no points of accumulation, see (2.2). We will call a piecewise constant function also a *step-function.* A WAF is in particular an increasing step-function in its second variable, see Figure 3 on page 29. Notice that these functions are left-continuous in their second variable which means that an arrival at t_2 is not taken into account by $S_{k,n}(t_1, t_2)$. In order to have possible releases at t_2 taken into account, we do not need to introduce a new function. We can use the right-hand side limits in the second variable. To see this, notice first that $[t_1, t_2] = \bigcap_{\varepsilon > 0} [t_1, t_2 + \varepsilon[$. Thus,

$$C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} \leqslant t_2]} = \lim_{\varepsilon \to 0^+} S_{k,n}(t_1, t_2 + \varepsilon) = S_{k,n}(t_1, t_2^+). \tag{2.7}$$

In a similar way, the right-hand side limits in the first variable implies that possible arrivals at t_1 are not taken into account: $S_{k,n}(t_1^+, t_2)$. Which of the variants is needed, depends on the context. Notice that on contiguous intervals, for example $[t_1, t_2[\cup [t_2, t_3[$, WAF's are additive:

$$S_k(t_1, t_2) + S_k(t_2, t_3) = S_k(t_1, t_3) \qquad \forall\, t_1 \leqslant t_2 \leqslant t_3. \tag{2.8}$$

In connection with feasibility it is useful to consider sets of instances $\mathcal{I}$ that are determined by a deadline d. The *deadline based work arrival function* a given task τ_k, which corresponds to $\mathcal{I} = \{\tau_{k,n} \mid n \in \mathbb{N},\ D_{k,n} \leqslant d\}$ is defined by:

$$S_k(t_1, t_2, d) \overset{\mathsf{def}}{=} \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[t_1 \leqslant A_{k,n} < t_2]} \cdot \mathbb{1}_{[D_{k,n} \leqslant d]}. \tag{2.9}$$

For the analysis of other scheduling policies than those considered in this document, it might be necessary to introduce further kinds of WAF, based on other sets of instances.

2.2 The scheduled task process

The processing capacity or speed of the processing unit is shared by the instances according to a given scheduling policy. In this section we introduce the *scheduling function*, which is an exact representation of the way in which the policy attributes the processing unit to tasks. We also define the resulting responses times of instances of a task.

2.2.1 The scheduling function

Since we only consider one processing unit at the same time, we can choose a unit capacity and assume that execution times and other characteristics are expressed in the corresponding time unit, to simplify the notations. The effect of the policy is "materialized" by the *scheduling function* Π that gives for every instance $\tau_{k,n}$ the part

$$\Pi_{k,n}(t) \in [0, 1]$$

of the processor capacity that the scheduler attributes to $\tau_{k,n}$.

Several assumptions have to be made about Π. Each function $\Pi_{k,n}$ is assumed to be *right-continuous* in t, see Appendix A.1 for an explanation. Notice that the right-continuity implies that the $\Pi_{k,n}$ are

Lebesgue-measurable. The following assumptions have interpretations in terms of the system to be modeled. The first is:

$$\forall\, t < A_{k,n} \qquad \Pi_{k,n}(t) = 0, \tag{2.10}$$

which means that an invocation can not use the processor before its activation date. Furthermore

$$\int_{t_1}^{t_2} \Pi_{k,n}(t)dt \leqslant C_{k,n} \qquad \forall\, t_1, t_2 \in \mathbb{R}, \tag{2.11}$$

which means that an invocation can not use the processor for longer than its nominal execution time, given unit processor speed. And finally

$$\forall\, t \in \mathbb{R}_+ \quad \textstyle\sum_{k=1}^{m} \sum_{n\in\mathbb{N}} \Pi_{k,n}(t) \leqslant 1, \tag{2.12}$$

meaning that the tasks can not use more capacity than available at time t. Let us denote $\mathcal{F}$ the set of functions Π satisfying (2.12). To represent the effect of a scheduling policy, we introduce the set $\mathbb{T}^{\Pi} \subset \mathbb{T} \times \mathcal{F}$ of *scheduled tasks sequences*, satisfying (2.10) and (2.11).

Definition 2.2 *A mapping* (A, C, D, Π) *from* Ω *into* $\mathbb{T}^{\Pi}$ *is called* scheduled task process.

A scheduled task process can represent *random* scheduling policies which take their decisions "by tossing a dice", but we will not consider such policies here. We are only interested in policies that apply a fixed rule. These policies are *deterministic* in the sense that the same task sequence (a, c, d) can be scheduled in only one way, i.e. is transformed into exactly one scheduled task sequence (a, c, d, π) and hence the abstract concept "deterministic policy" can be represented as a function from $\mathbb{T}$ into $\mathbb{T}^{\Pi}$.

Definition 2.3 *A mapping* ν *from a* $\mathbb{T}$ *into* $\mathbb{T}^{\Pi}$ *is called a* deterministic scheduling policy.

With this definition, a task process is said to be scheduled according to policy ν if

$$(A, C, D, \Pi) = \nu((A, C, D)). \tag{2.13}$$

Figure 1 shows a graphical representation of ν and all other concepts defined within our model.

The scheduling function Π of a task process does depend on ω because the task process is a function of ω, but not because of the scheduling policy. For random policies it would be necessary to introduce a "stochastic driver", that is a random variable which represents the random part of the scheduling policy.

2.2.2 Response times

The basic structure being settled, we can define the quantities we are interested in, such as *response times*, *execution beginnings* and *execution ends.* ends. A very useful tool for their analysis are workload functions, defined by: *workload function*

$$W_{k,n}(t) \overset{\mathsf{def}}{=} S_{k,n}(0, t) - \int_0^t \Pi_{k,n}(u)du. \tag{2.14}$$

For each instance $\tau_{k,n}$ it represents the amount of work which has arrived before t and which is still waiting for execution. Notice that $W_{k,n}(A_{k,n}) = 0$ because $S_{k,n}(0, A_{k,n})$ does not account for the arrival at $A_{k,n}$. This property corresponds to the left-continuity of $W_{k,n}(t)$, which is induced by the continuity of the integral of $\Pi_{k,n}$ and the left-continuity of $S_{k,n}(0, t)$. Taking the right-hand side limit in t gives:

$$W_{k,n}(t^+) = S_{k,n}(0, t^+) - \int_0^t \Pi_{k,n}(u)du. \tag{2.15}$$

It represents the pending work at t whether it arrives before or at t.

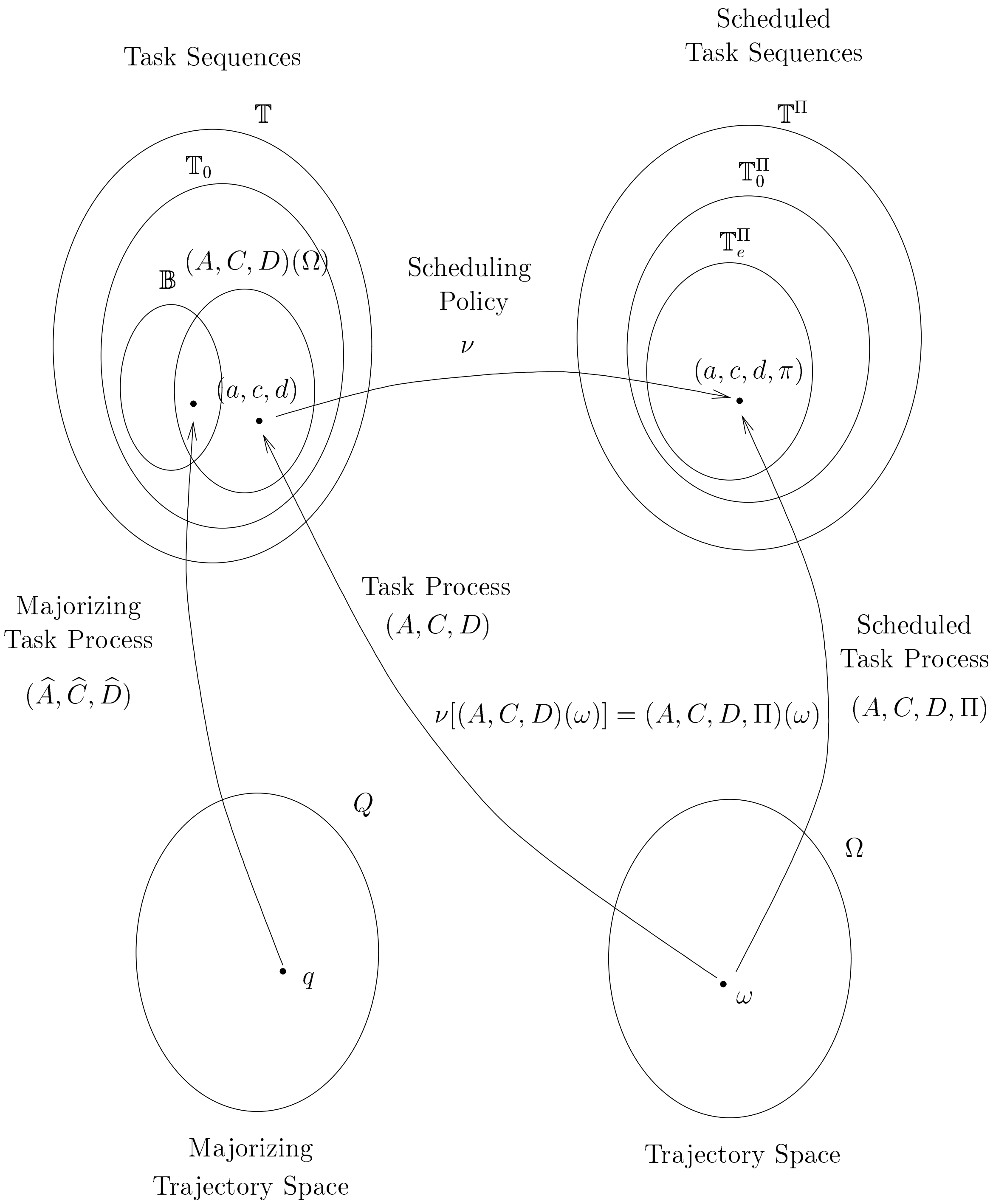


Figure 1: Overview of the model

Definition 2.4 *An instance $\tau_{k,n}$ is said to be* pending *at a time t if $W_{k,n}(t^+) > 0$.*

Notice that a task can only be executed if it is pending:

$$\Pi_{k,n}(t) > 0 \Rightarrow W_{k,n}(t) > 0. \tag{2.16}$$

It is implied by assumptions (2.10) and (2.11). Notice also the following property:

Lemma 2.5 *The number of pending instances is finite on any interval of finite length:*

$$\forall\, t_1 < t_2 \in \mathbb{R}_+ \qquad \sum_{\tau_{k,n}} \mathbb{1}_{[W_{k,n}(t^+)>0]} \cdot \mathbb{1}_{[t_1 \leqslant t < t_2]} < \infty. \tag{2.17}$$

Proof: Because of (2.4), $\Pi(t) \in [0,1]$ and the definition (2.14), $W_{k,n}(t) > 0$ implies $A_{k,n} \leqslant t$. But in the interval $[0, t_2[$ there can only be a finite number of activations because of (2.1). ■

It is sometimes necessary to consider only instances that have deadlines earlier than some deadline d. For this purpose we define deadline based version of the involved functions:

$$W_{k,n}(t,d) \overset{\mathsf{def}}{=} W_{k,n}(t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]} \quad S_{k,n}(0,t,d) \overset{\mathsf{def}}{=} S_{k,n}(0,t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]} \quad \Pi_{k,n}(t,d) \overset{\mathsf{def}}{=} \Pi_{k,n}(t) \cdot \mathbb{1}_{[D_{k,n} \leqslant d]}.$$

The sum over all instances of a task gives $W_k(t,d)$, $S_k(0,t,d)$ and $\Pi_k(t,d)$. It can easily be seen that

$$W_{k,n}(t,d) = S_{k,n}(0,t,d) - \int_0^t \Pi_{k,n}(u,d)du.$$

A pending instance completes its execution as soon as its workload vanishes. Thus, the *execution end* or *completion time* of an instance can be defined by

$$E_{k,n} \overset{\mathsf{def}}{=} \min\{t > A_{k,n} \mid W_{k,n}(t) = 0\}, \tag{2.18}$$

if it exists, that is if the set on which the definition is based on is not empty. Notice that between $A_{k,n}$ and $E_{k,n}$, the execution of $\tau_{k,n}$ could be interrupted if the scheduling policy is preemptive and thus, the *response time*, defined by

$$R_{k,n} \overset{\mathsf{def}}{=} E_{k,n} - A_{k,n} \tag{2.19}$$

can be longer than the execution time $C_{k,n}$. Notice that because we assume $\Pi_{k,n}$ to be right-continuous, we have $\Pi_{k,n}(t) = 0$ for $t \geqslant E_{k,n}$, see Corollary A.3.

The *execution begin* of an instance can be defined as the first time after its activation, where its scheduling function becomes positive (see appendix A.1.2, for why it can indeed be defined like this):

$$B_{k,n} \overset{\mathsf{def}}{=} \inf\{t \geqslant A_{k,n} \mid \Pi_{k,n}(t) > 0\}. \tag{2.20}$$

2.2.3 Feasibility

Under real-time constraints, tasks must complete their execution within their *relative deadline* $\overline{D}_{k,n}$, i.e. before their *absolute deadline* $D_{k,n} = A_{k,n} + \overline{D}_{k,n}$. If all instances meet their deadlines then the task set is *feasible*. In our model feasibility is defined a follows:

Definition 2.6 *A set of tasks is* feasible *under a certain scheduling policy, if in the corresponding scheduled task process (A, C, D, Π), all task instances finish before their deadline:*

$$E_{k,n}(\omega) \leqslant D_{k,n}(\omega) \qquad \forall \omega \in \Omega, \forall k = 1..m, \forall n \in \mathbb{N}.$$

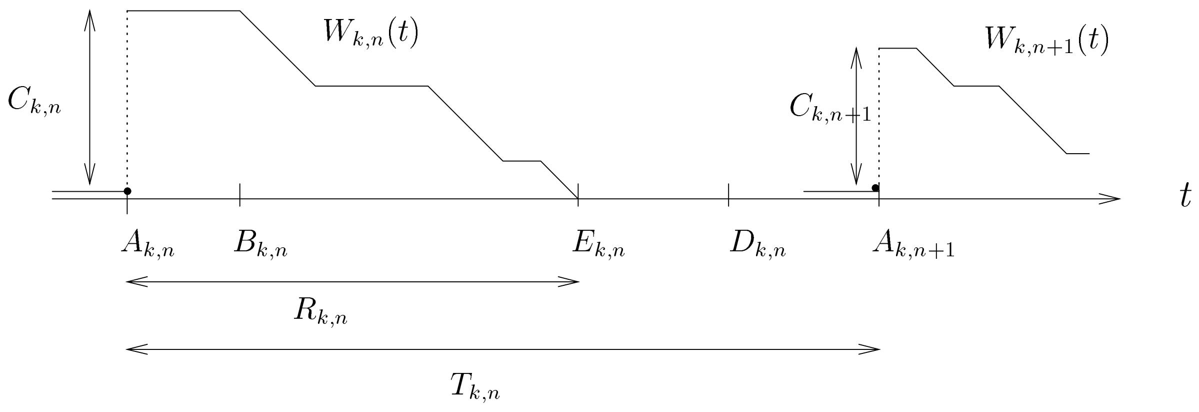


Figure 2: Times and functions associated with the instance of a task τ_k.

Notice that feasibility only concerns the property that response times are shorter than relative deadlines. On the other hand, maximal response times or bounds on response times, provide additional information. They tell how close to the deadline response times could come. Close could be below or above. If a task is not feasible, it allows to know by how much a deadline is not met and if it is feasible it gives an idea about the safety margin. These are useful informations for the design of a system. If the relative deadlines of the instances of a task are all equal, $D_{k,n} = \overline{D}_k \ \forall n \in \mathbb{N}$, then the maximal response times of a task allows to say if the task is feasible or not. A task is then feasible if and only if

$$R_k^{max} \overset{\mathsf{def}}{=} \max_{\omega \in \Omega, n \in \mathbb{N}} R_{k,n}(\omega) \leqslant \overline{D}_k.$$

2.3 Basic assumptions

As said above, execution ends or even beginnings might be undefined. This can happen if the schedule does not allow an instance of a task to execute. In the context of real-time system where feasibility is the seeked property, such behaviors are of course undesired. If the used scheduling is *non-idling*, meaning that the processing unit does not idle if work is pending, then the existence or non-existence of execution ends or beginnings is related to the long term load on the processing unit by task's demands. Since we are only interested in *non-idling* scheduling policies, we will impose a *deterministic stability* condition, that guarantees together with the non-idling assumption that all execution ends are defined. In this section we express these properties in the model of scheduled task processes and show which kind of basic properties they induce.

2.3.1 Deterministic stability

The aim of introducing this stability condition is to ensure that response times are finite. It does not imply feasibility but it is a necessary condition. If response times are not finite, then there is no use in trying to derive response time bounds. Furthermore, the commonly used iterative algorithm to compute response time bounds numerically is only valid if the stability condition is satisfied.

Stability can be interpreted as the guarantee that the processing unit is not overloaded by task demands in the long run. To determine if a set of tasks does or does not overload the processing unit on the long run the tasks demands must be estimated. For this purpose, *(σ,ρ)-bounds* [10] are appropriate. They are based on a slope $\rho_k \in \mathbb{R}_+$ and constant $\sigma_k \in \mathbb{R}_+$ that satisfy

$$S_k(t_1,t_2;\omega) = \sum_{n\in\mathbb{N}} S_{k,n}(t_1,t_2;\omega) \leqslant \sigma_k + \rho_k \cdot (t_2 - t_1) \qquad \forall\, t_1 \leqslant t_2 \in \mathbb{R}_+,\ \forall\, \omega \in \Omega. \tag{2.21}$$

It means that the task's WAF's are uniformly bounded by one affine function on all intervals of all trajectories. The sums $\sigma_{1..m} \overset{\mathsf{def}}{=} \sum_{i=1}^m \sigma_i$ and $\rho_{1..m} \overset{\mathsf{def}}{=} \sum_{i=1}^m \rho_i$ give a (σ,ρ)-bound for the whole task set. On any interval $[t_1,t_2[$ the average demand of tasks is bounded by

$$\frac{S_{1..m}(t_1,t_2)}{t_2-t_1} \leqslant \frac{\sigma_{1..m}}{t_2-t_1} + \rho_{1..m}.$$

If $\rho_{1..m} < 1$, then $t_2 > t_1 + \sigma_{1..m}/(1-\rho_{1..m})$ implies that the average demand is lower than the processor capacity in the interval. Thus there is some idle time left and hence the processor is not overloaded. If no (σ,ρ)-bounds exists with $\rho_{1..m} < 1$, then it can not be guaranteed that response times admit finite bounds. In the case where no (σ,ρ)-bounds exists with $\rho_{1..m} < 1$, then response times could nevertheless be bounded but examples show that the processor is then at the limit between stability and instability and thus feasibility and infeasibility. We do not want to consider such systems, because their behavior is difficult to predict in general and because they do not seem to be useful for real-world problems.

Definition 2.7 *A task process is said to be* stable *if there exist (σ,ρ)-bounds such that $\rho_{1..m} < 1$. The subset of stable task sequences is denoted $\mathbb{T}_0$.*

Notice that this property is independent of the used scheduling policy. Together with the non-idling assumption this implies policy independent properties, which are presented in Section 2.3.3.

The slope $\rho_{1..m}$ is not to be confused with a bound on the average load, which could be defined under stochastic assumptions. Suppose for example that inter-arrivals are distributed i.i.d. exponential, i.e. they are given by a Poisson process, and execution times are also i.i.d. exponential. The average load $\sum_{k=1}^m \mathbb{E}\ C_{k,n}/\mathbb{E}\ T_{k,n}$ exists, but no (σ,ρ)-bound can be found. The reason is that for any interval and any threshold, there is a positive probability that this threshold is exceeded. A (σ,ρ)-bound can only exist if the the execution time distribution is bounded from above by some C^{max} and the inter-activation time distribution is bounded from below by some $T^{min} > 0$, then

$$f(t) = C^{max} + \frac{C^{max}}{T^{min}} \cdot t$$

is a (σ,ρ)-bound. This situation is related to the fact that under real-time constraints deterministic bounds are required, which leads to a stronger kind of stability condition, than in queuing theory.

2.3.2 Non-idling

In order to state this property of scheduling policies, we introduce

$$S_{1..m}(t_1,t_2) = \sum_{i=1}^m S_i(t_1,t_2) \qquad W_{1..m}(t) = \sum_{i=1}^m W_i(t) \qquad \Pi_{1..m}(t) = \sum_{i=1}^m \Pi_i(t) \tag{2.22}$$

that are, respectively, the total amount of work arriving on $[t_1,t_2[$, the total pending work from the past and the total use of the processing time at time t. Notice that the considered sums potentially concern infinite numbers of terms, but since an arrival process is assumed to have no points of accumulation (2.2), $S_{1..m}$ is finite and piecewise constant in t_2, for a fixed ω, implying that its left-continuity is preserved.

Definition 2.8 *A scheduled task process is said to be* non-idling *if*

$$W_{1..m}(t) > 0 \quad \Rightarrow \quad \Pi_{1..m}(t) = 1. \tag{2.23}$$

The subset of scheduled task sequences satisfying (2.23) is denoted $\mathbb{T}_0^{\Pi}$.

It means that the processing unit is fully used if work is pending. As counter example, consider a task that has to do some pre-processing, acquisition of data (say, from a buffer) and finally the actual processing. If the task has to wait before the data is available, then an interval could occur where the processor is idling although work is pending - the work that corresponds to the actual processing. This work has to be considered as pending because the definition of the task comprises all three steps and by the definition of activation dates, the entire amount of work is considered as pending as soon as the task is activated. In such a case, the initial task would have to be subdivided into two tasks with a precedence relation.

2.3.3 Properties

Under the stability and the non-idling assumption some general properties of scheduled task processes can be established. We will show that in this case the response times of all tasks are finite. This property is related to the concept of processor busy periods.

Definition 2.9 *An interval* $[t_1, t_2[$ *is said to be a* processor busy period *if it satisfies:*

$$\Pi_{1..m}(t) = 1, \qquad \forall\, t \in [t_1, t_2], \tag{2.24}$$

$$W_{1..m}(t_1) = W_{1..m}(t_2^+) = 0. \tag{2.25}$$

Because activations don't have points of accumulation by (2.2) on page 7, the last arrival before t_1 and the first arrival after t_2 are situated at positive distances of the interval and thus (2.25) is equivalent to

$$\exists\, \varepsilon > 0 \text{ such that } \Pi_{1..m}(t) = 0 \quad \forall\, t \in [t_1 - \varepsilon, t_1[\, \cup\,]t_2, t_2 + \varepsilon[,$$

which is exactly the definition of Lehoczky's busy period at processor level in our model, see also Section 4.2.2.

Proposition 2.10 *Busy periods of a stable and non-idling scheduled task process are bounded.*

Proof:
Let $[t_1, t_2[$ be a processor busy period. From (2.22) we get

$$W_{1..m}(t_2) = W_{1..m}(t_1) + S_{1..m}(t_1, t_2) - \int_{t_1}^{t_2} \Pi_{1..m}(t)dt.$$

By definition 2.9, $\Pi_{1..m}(t) = 1$ on $[t_1, t_2[$ and $W_{1..m}(t_2^+) = W_{1..m}(t_1) = 0$. Since $W_{1..m}(t_2^+) = W_{1..m}(t_2) + S_{1..m}(t_2, t_2^+)$ we have $W_{1..m}(t_2) = 0$. Thus,

$$S_{1..m}(t_1, t_2) = t_2 - t_1.$$

On the other hand $S_{1..m}$ is bounded: $S_{1..m}(t_1, t_2) \leqslant \sigma_{1..m} + \rho_{1..m} \cdot (t_2 - t_1)$, see (2.21), and hence, since $\rho_{1..m} < 1$ and $\sigma_{1..m} \geqslant 0$,

$$t_2 - t_1 \leqslant \frac{\sigma_{1..m}}{1 - \rho_{1..m}}.$$

■

From this proposition follows a property which is independent of scheduling policies and task types:

Corollary 2.11 *An interval* $[A_{k,n}, E_{k,n}[$, *during which an instance* $\tau_{k,n}$ *is pending is part of some processor busy period. The response time* $R_{k,n} = E_{k,n} - A_{k,n}$ *is bounded if the scheduled task process is stable and non-idling.*

Proof:
The definitions of the workload (2.14) and the execution end (2.18) on page 12 imply that $W_{k,n}(t) > 0$ for $t \in]A_{k,n}, E_{k,n}[$ and thus $W_{1..m}(t) > 0$. Because of the non-idling assumption (2.23) on the page before we also have $\Pi_{1..m}(t) > 0$ and thus $]A_{k,n}, E_{k,n}[\subset [t_1, t_2]$, where $[t_1, t_2]$ is a processor busy period. Thus

$$R_{k,n} = E_{k,n} - A_{k,n} \leqslant t_2 - t_1 \leqslant \frac{\sigma_{1..m}}{1 - \rho_{1..m}}.$$

■

Response time bounds as in the proof of this corollary have been derived by Chaouiya et al. in [9] for periodic tasks.

2.4 Bounds and critical instances

In order to derive response time bounds and to test feasibility, it is necessary to be able to bound the demands of each task. An example are the (σ,ρ)-bounds introduced in Section 2.3.1. In a more general way, we define in this section *majorizing work arrival function*, a concept that allows to express these bounds appropriately in our model. We show the connections with the well known concept of *critical instant* in Section 2.4.2. Section 2.4.3 is about tight (σ,ρ)-bounds for tasks which are needed to detect processor overloads in the long term.

2.4.1 Majorizing work arrival functions (MWAF)

To bound the response time of an instance or to check feasibility, it is necessary to bound the concurrent demands from other tasks. To formalize this, we define the concept of *majorizing work arrival function.*

Definition 2.12 A majorizing work-arrival function *(MWAF) for the demands of a task* τ_k *after some epoch* u *on some trajectory* ω, *is a function* $\widehat{S}_k(x)$, *that bounds the WAF of* τ_k:

$$S_k(u, u + x; \omega) \leqslant \widehat{S}_k(x) \;\; \forall\, x > 0. \tag{2.26}$$

The function $\widehat{S}_k(x)$ could be a conservative bound or an exact bound. For deadline based WAF we define in an analogous way *majorizing deadline based work arrival function.*

Definition 2.13 A majorizing deadline based work-arrival function *for the demands of a task* τ_k *after a time* u *on a trajectory* ω, *is a function* $\widehat{S}_k(x, d)$, *that bounds the deadline based WAF of* τ_k:

$$S_k(u, u + x, u + d; \omega) \leqslant \widehat{S}_k(x, d) \;\; \forall\, x > 0, d > 0. \tag{2.27}$$

For the policies considered in this document these kinds of MWAF allow to derive response time bounds. We denote a MWAF, for which (2.26), resp. (2.27) do hold for every trajectory and every moment u, by $\widehat{S}_k^*$ and say it is a *unique* MWAF.

The following proposition can be helpful for deriving MWAF in practice. It allows to derive a MWAF for an interval of limited length and to extend over all $\mathbb{R}_+$.

Proposition 2.14 *Suppose a unique MWAF* $\widehat{S}_k^*$ *is available only for intervals up to a maximal length* y, *i.e* $S_k(u, u + x) \leqslant \widehat{S}_k^*(x)$ *for all* u, *but* $x \leqslant y$. *Then a MWAF for all* $x \in \mathbb{N}$ *can be obtained by extension:*

$$S_k(u, u + x) \leqslant \widehat{S}_k^*(y) \cdot \lfloor x/y \rfloor + \widehat{S}_k^*(x \bmod y) \quad \forall\, u, x. \tag{2.28}$$

If $\widehat{S}^*_k$ is furthermore a deadline based MWAF, i.e. $S_k(u,u+x,u+d) \leqslant \widehat{S}^*_k(x,d)$ for all u, all d, but $x \leqslant y$, then a deadline based MWAF for all x is given by

$$S_k(u,u+x,u+d) \leqslant \sum_{i=0}^{\lfloor x/y \rfloor} \widehat{S}^*_k(y,d-i\cdot y) + \widehat{S}^*_k(x \bmod y, d-\lfloor x/y \rfloor \cdot y) \quad \forall\, u,x,d. \tag{2.29}$$

Proof: Subdivide the interval into contiguous subintervals of equal length y as far as possible

$$[u,u+x[= \bigcup_{i=0}^{\lfloor x/y \rfloor} [u_i,u_{i+1}[\bigcup [u_{\lfloor x/y \rfloor}, u+x[\qquad \text{with } u_i = u + i\cdot y$$

and use (2.26) for each resulting WAF term, see (2.8). Since $S_k(u_i,u_{i+1}) \leqslant \widehat{S}_k(y)$, a sum of $\lfloor x/y \rfloor$ equal terms $\widehat{S}^*_k(y)$ appears and thus (2.28) is obtained. In the deadline based case we obtain $S_k(u_i,u_{i+1},u+d-u_i) \leqslant \widehat{S}^*_k(y,d-u_i)$ and since $u_i = i\cdot y$ (2.29) is proven. ■

A task may have several distinct behaviors, depending on the context in which it is released. For each behavior an individual tight MWAF's may exist but their supremum could be a rather conservative bound, resulting in response time bounds which are much longer than the actual maximum. To improve the analysis in such a case the different behaviors must be analyzed separately. This has been done for example by Tindell for tasks with offset constraints, see [33]. Hereafter, we define the appropriate concepts for our model.

The analysis of tasks with mode changes and also multi-frame tasks shows that it is useful to have several MWAF for the same task. To formalize this, let us introduce a *family of majorizing work arrival functions.*

Definition 2.15 *A family of majorizing work arrival functions for a task τ_k, is a set of functions $\mathcal{S}_k = \{\widehat{S}^{(q_k)}_k(x) \mid q_k \in Q_k\}$ such that for any time u and trajectory ω, there exists a function of the family that majorizes the WAF of the task after u on ω:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \exists q_k \in Q_k \qquad S_k(u,u+x;\,\omega) \leqslant \widehat{S}^{(q_k)}_k(x)\ \ \forall\, x > 0.$$

For the case where the scheduling policy relies explicitly on deadlines, the corresponding deadline based definition is needed:

Definition 2.16 *A family of majorizing deadline based work arrival functions for a task τ_k is a set of functions $\mathcal{S}_k = \{\widehat{S}^{(q_k)}_k(x,d) \,|\, q_k \in Q_k\}$ such that for any time u and trajectory ω, there exists a function of the family that majorizes the WAF of the task after u on ω:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \exists q_k \in Q_k \qquad S_k(u,u+x,u+d;\,\omega) \leqslant \widehat{S}^{(q_k)}_k(x,d)\ \ \forall\, x > 0.$$

A family of majorizing WAF's for tasks allows a more accurate description of their individual behavior than a single WAF's does, but as such, it does not allow to account for dependencies between tasks (like offset relations) and to represent the resulting global patterns exactly. To allow this we introduce a family of MWAF for the whole task set:

Definition 2.17 *A family of majorizing work arrival functions for a set of tasks is a set of functions $\mathcal{S} = \{\widehat{S}_k(x,q) \,|\, k = 1..m,\ q \in Q\}$ such that for any time u and trajectory ω, there exists an index $q \in Q$ such that the WAF of each task is bounded by the corresponding MWAF:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \exists q \in Q \qquad S_k(u,u+x;\,\omega) \leqslant \widehat{S}_k(x,q)\ \ \forall\, x > 0,\ k = 1,\dots,m.$$

Definition 2.18 *A family of majorizing deadline based work arrival functions for a set of tasks is a set of functions* $\mathcal{S} = \{\widehat{S}_k(x, d, q) \mid k = 1..m,\ q \in Q\}$ *such that for any time* u *and trajectory* ω*, there exists an index* $q \in Q$ *such that the WAF of each task is bounded by the corresponding MWAF:*

$$\forall u \in \mathbb{R}_+,\ \forall \omega \in \Omega\ \ \exists q \in Q \qquad S_k(u, u + x, u + d;\, \omega) \leqslant \widehat{S}_k(x, d, q)\ \ \forall\, x > 0, k = 1, \ldots, m.$$

Dependencies between activation patterns of tasks may be represented as follows. Suppose we have for each task as family of MWAF's $\mathcal{S}_k = \{\widehat{S}_k^{(q_k)}(x) \mid q_k \in Q_k\}$. If we choose $Q \subset Q_1 \times Q_2 \times \ldots \times Q_m$ with $q = (q_1, q_2, \ldots , q_m) \in Q$ and define $\widehat{S}_k(x, q) = \widehat{S}_k^{(q_k)}(x)$ then the dependencies are represented by the particular combinations $q = (q_1, q_2, \ldots , q_m)$ of MWAF that are part of Q. An example are tasks with offset constraints, where given the activation pattern of one task, other tasks can only be activated with certain offsets and thus only certain activation patterns of other tasks can occur (see Section 3.2.3 for more details).

Remark: A concept similar to MWAF has been introduced by Baruah et al. in [6] to express a feasibility conditions under EDF for generalized multi-frame tasks. It is the feasibility condition which does not rely on response time bounds, see Section 4.3.3. The so called *demand bound function* $\mathsf{dbf}(\tau_k$,t) is defined for each task and gives the maximum demand of instance of a task τ_k in an interval of length t and with absolute deadline within the considered interval.

Thus it corresponds to the case of a unique deadline based MWAF and deadlines equal to the end of the interval:

$$\mathsf{dbf}(\tau_k, t) = \widehat{S}_k^*(0, t, t).$$

2.4.2 Critical instant and majorizing task process

Above, we have made no particular assumption about what kind of functions MWAF should be, only that they must be bounds. They could for example be linear functions, like (σ,ρ)-bounds. If a MWAF is an increasing, left-continuous step-function just as an actual WAF is, then a sequence of *virtual* activation epochs $\widehat{A}_{k,n}$ with execution times $\widehat{C}_{k,n}$, may be found such that

$$\widehat{S}_k(x) = \sum_{n \in \mathbb{N}} \widehat{C}_{k,n} \cdot \mathbb{1}_{[\hat{A}_{k,n} < x]}.$$

As deadline we choose $\widehat{D}_{k,n} = \widehat{A}_{k,n} + \overline{D}_k$. We call them virtual because they do not correspond to actual instances of the task. The corresponding cycle times are $\widehat{T}_{k,n} = \widehat{A}_{k,n+1} - \widehat{A}_{k,n}$. The releases of tasks after some epoch u follow a certain pattern which is captured by the WAF's $S_k(u, u + x)$. We will therefore equivalently use the terms *majorizing activation pattern* for MWAF's.

If deadline based MWAF's are left resp. right-continuous step-functions in variables x and d, then sequences $\hat{A}_{k,n}$, $\widehat{C}_{k,n}$ and $\widehat{D}_{k,n}$ might exist such that

$$\widehat{S}_k(x, d) = \sum_{n \in \mathbb{N}} \widehat{C}_{k,n} \cdot \mathbb{1}_{[\widehat{A}_{k,n} < x]} \cdot \mathbb{1}_{[\widehat{D}_{k,n} \leqslant d]}.$$

but not necessarily.

If for each function of a family of MWAF a sequence of activations exists, then they can be seen as the WAF of a task process with trajectory space Q

$$\widehat{S}_k(0, x; q) = \widehat{S}_k(x, q).$$

Definition 2.19 *A majorizing task process for* (A, C, D)*, is a process* $(\widehat{A}, \widehat{C}, \widehat{D})$*, with trajectory space* Q*, such that for all* $\omega \in \Omega$*,* $\forall u \in \mathbb{R}_+$ *a trajectory* $q \in Q$ *exists satisfying for all* $k = 1, \dots, m$

$$S_k(u, u+x;\, \omega) \leqslant \widehat{S}_k(0, x;\, q) \qquad \forall x. \tag{2.30}$$

The set $\mathbb{B} = (\widehat{A}, \widehat{C}, \widehat{D})(Q)$ *is called the set of majorizing task sequences.*

If for each function of a family of deadline based MWAF a sequence of activation exists, then they can be seen as MWAF of a task process with

$$\widehat{S}_k(0, x, d; q) = \widehat{S}_k(x, d, q).$$

Definition 2.20 *A deadline based majorizing task process, is a task process* $(\widehat{A}, \widehat{C}, \widehat{D})$*, with trajectory space* Q*, such that for all* $\omega \in \Omega$*, and for all* $u \in \mathbb{R}_+$ *a trajectory* $q \in Q$ *exists satisfying for all* $k = 1, \dots, m$

$$S_k(u, u+x, u+d;\, \omega) \leqslant \widehat{S}_k(0, x, d;\, q) \qquad \forall x, d \in \mathbb{R}_+. \tag{2.31}$$

Notice that when $d \to +\infty$, (2.31) reduces to (2.30), implying that a deadline based majorizing WAF is also an ordinary majorizing WAF.

Definition 2.21 *A family of MWAF consists of* exact bounds *if for each* q *there exist a trajectory* ω *and a time* u*, after which the WAF are equal to the MWAF's:*

$$\forall\, q \in Q, \quad \exists\, \omega \in \Omega, u \in \mathbb{R}_+ \quad \vdash \quad S_k(u, u+x, u+d; \omega) = \widehat{S}_k(x, d, q) \quad \forall\, x, d \in \mathbb{R}_+, k = 1..m. \tag{2.32}$$

In that case a majorizing task process necessarily exists and its trajectories are a collection of worst case activation pattern that can be realized by the initial task process. A worst case activation pattern of tasks is commonly called **critical instant**. The canonical example is the critical instance for periodic tasks [22], where all task are activated at the same time. For *accumulatively monotonic multi-frame tasks*, see [24] and Section 3.1.3, there exists also a unique critical instant, but if the task execution times are not accumulatively monotonic then several critical patterns exist, because none of them represent larger demands than all the others on all intervals of time. The same is true for tasks with offset constraints, see [33] and Section 3.2.3, generalized multi-frame tasks, see [6] and Section 3.2.4, or recurrent branching tasks [4]. Because of the number of different worst-case patterns that must be analyzed, the feasibility tests or the computation of response times could be too complex to be tractable, so that approximations are necessary. We will return to this point on several occasions in subsequent sections.

2.4.3 Tight (σ,ρ)-bounds.

Let us return to the problem of deriving (σ,ρ)-bounds, which are needed to determine if a task set is stable. Usually, one tries to find the bound with the smallest possible slope to avoid over-estimation of the bound function. The question that naturally arises is if there exists a general method of determining the tightest possible slope. The answer is given by the following proposition, but requires some explanations. Let

$$\rho_k(\omega) \overset{\mathsf{def}}{=} \overline{\lim}_{t \to +\infty} \frac{S_k(0, t\,; \omega)}{t} \qquad \text{and} \qquad \rho_k^* \overset{\mathsf{def}}{=} \sup_{\omega \in \Omega} \rho_k(\omega). \tag{2.33}$$

Proposition 2.22 *If a* (σ,ρ)*-bound with slope* ρ_k *exists, then* $\rho_k \geqslant \rho_k^*$.

Proof: Suppose a (σ,ρ)-bound exists, i.e. $\forall\omega \in \Omega$

$$\frac{S_k(0,t;\omega)}{t} \leqslant \frac{\sigma_k}{t} + \rho_k.$$

As t tends towards $+\infty$, the inequality tends to $\rho_k(\omega) \leqslant \rho_k$. Since it is valid for all ω, we have $\rho_k^* \leqslant \rho_k$. ■

The slope ρ_k^* is only an infimum and not necessarily a minimum of all possible slopes for a (σ,ρ)-bound. To illustrate this consider the following example. Suppose the WAF of a task has a shape of the form $f(t) = \rho_k^* \cdot t + \ln(t+1)$. We have $\lim_{t\to\infty} f(t)/t = \rho_k^*$. For every given slope $\rho_k > \rho_k^*$ the tightest (σ,ρ)-bound is given by the tangent of $f(t)$, corresponding to that slope:

$$\hat{f}(t) = \rho_k \cdot t + \rho_k - \rho_k^* - 1 - \ln(\rho_k - \rho_k^*).$$

It is easily seen that $\sigma_k = \hat{f}(0) = \rho_k - \rho_k^* - 1 - \ln(\rho_k - \rho_k^*)$. When ρ_k decreases to ρ_k^* then σ_k tends to $+\infty$. Thus, there is no (σ,ρ)-bound with slope ρ_k^* and hence ρ_k^* is not a minimum but an infimum. A (σ,ρ)-bound can nevertheless be obtained by taking a slightly larger slope.

Another point to note is that the existence of ρ_k^* does not imply the existence of the (σ,ρ)-bound. To see this consider the case of a task with $T_{k,n} = 2^n$ and $C_{k,n} = 2^{n-2}$. Notice that $S_k(0,t) = S_k(0, A_{k,n}^+)$ for $t \in]A_{k,n}, A_{k,n+1}]$. The releases occur at $A_{k,n} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$ and $S_k(0, A_{k,n}^+) = \sum_{i=0}^{n} 2^i/4 = (2^{n+1}-1)/4$. Thus

$$\overline{\lim_{t\to+\infty}} \frac{S_k(0,t)}{t} = \lim_{n\to+\infty} \frac{(2^{n+1}-1)}{4\cdot(2^n-1)} = \frac{1}{2}.$$

But because $S_k(A_{k,n}, A_{k,n}+t) \geqslant (2^{n+1}-1)/4$, we have $\sup_u S_k(u,u+t) = +\infty$ and thus no (σ,ρ)-bound exists.

Tasks, as they appear in real-world system, do usually not behave as in the particular case just described; usually the limit ρ_k^* can easily be computed and a (σ,ρ)-bound with this slope exists. The following proposition can often be used.

Proposition 2.23 *If a unique MWAF is available for interval of maximal length y, i.e. $S_k(u, u+x;\omega) \leqslant \widehat{S}_k^*(x)$ for all ω and all u, but $x \leqslant y$ then a (σ,ρ)-bound with the following parameters exists:*

$$\sigma_k = \widehat{S}_k^*(y) \qquad\qquad \rho_k = \widehat{S}_k^*(y)/y. \tag{2.34}$$

Proof: To prove (2.34), divide both sides of (2.28) by $x = \lfloor x/y \rfloor \cdot y + (x \mod y)$. The left-hand side can be rewritten as

$$\frac{\widehat{S}_k^*(y)\cdot\lfloor x/y\rfloor}{\lfloor x/y\rfloor\cdot y} - \frac{\widehat{S}_k^*(y)\cdot\lfloor x/y\rfloor\cdot(x \mod y)}{\lfloor x/y\rfloor\cdot y\cdot x} + \frac{\widehat{S}_k^*(x \mod y)}{\lfloor x/y\rfloor\cdot y + (x \mod y)}$$

by splitting the first term into two. As $x \to +\infty$ this expression tends to $\widehat{S}_k^*(y)/y$. Now, since

$$\widehat{S}_k^*(y)\cdot\lfloor x/y\rfloor + \widehat{S}_k^*(x \mod y) \leqslant \widehat{S}_k^*(y)\cdot\frac{x}{y} + \widehat{S}_k^*(y),$$

the result is proven. ■

The slopes ρ_k of (σ,ρ)-bounds on MWAF's have some importance for the numerical computation of response times bounds, see Corollary A.8 and Proposition A.9. Let us thus introduce:

$$\hat{\rho}_k(q) \overset{\mathsf{def}}{=} \overline{\lim_{t\to+\infty}} \frac{\widehat{S}_k^{(q)}(x)}{x}, \qquad \hat{\rho}^* \overset{\mathsf{def}}{=} \sup_{q\in Q} \hat{\rho}_k(q), \qquad \hat{\sigma}^* \overset{\mathsf{def}}{=} \sup_{q\in Q} \hat{\sigma}_k(q).$$

Proposition 2.24 *Let $\mathcal{S}_k = \{\widehat{S}_k^{(q)}(x) \mid q \in Q\}$ be a family of exact MWAF's. If $\forall\, q \in Q$, $\hat{\rho}(q) < \infty$ and there exist $\hat{\sigma}_k(q) \in \mathbb{R}_+$ so that $\widehat{S}_k^{(q)}(x) \leqslant \hat{\sigma}_k(q) + \hat{\rho}_k(q) \cdot x \quad \forall\, x \in \mathbb{R}_+$ and if $\hat{\rho}^* < \infty$ and $\hat{\sigma}^* < \infty$, then $\rho^* = \hat{\rho}^*$ and $\hat{\rho}^*$ is the smallest slope of a (σ,ρ)-bound for the task.*

Proof: The definition of $\mathcal{S}_k$ implies that for each ω' there exists a q' such that $S_k(0,x;\omega') \leqslant \widehat{S}_k^{(q')}(x)$, $\forall\, x$. Hence, $\rho_k(\omega') \leqslant \hat{\rho}_k(q'(\omega'))$ and thus $\rho^* \leqslant \hat{\rho}^*$. On the other hand, since it is assumed to be an exact bound, there is for each q an ω and a u such that $\widehat{S}_k^{(q)}(x) = S_k(u, u+x;\omega)$.

Notice first that in general for given $u, v \in \mathbb{R}_+$, $\lim_{x\to+\infty}(v+f(x))/(u+x) = \lim_{x\to+\infty} f(x)/x$. Now, we can derive:

$$\lim_{t\to+\infty} \frac{S_k(u,t;\omega)}{t} = \lim_{x\to+\infty} \frac{S_k(0,u+x;\omega)}{u+x} = \lim_{x\to+\infty} \frac{S_k(0,u;\omega) + S_k(u,u+x;\omega)}{u+x} = \lim_{x\to+\infty} \frac{\widehat{S}_k^{(q)}(x)}{x}.$$

Thus, for each $q \in Q$ there exists an $\omega(q)$ such that $\hat{\rho}_k(q) = \rho_k(\omega(q))$. Thus, $\hat{\rho}_k^* \leqslant \rho_k^*$ and finally $\rho^* = \hat{\rho}^*$. Since $\hat{\sigma}_k^*$ and $\hat{\rho}_k^*$ are supposed to be finite we have a (σ,ρ)-bound for all trajectories of the task process:

$$S_k(u, u+x;\omega) \leqslant \hat{\sigma}_k^* + \hat{\rho}_k^* \cdot x \qquad \forall\, \omega \in \Omega,\ \forall\, u \in \mathbb{R}_+,\ \forall\, x \in \mathbb{R}_+.$$

Proposition 2.22 implies that $\hat{\rho}_k^*$ is the smallest possible slope of a (σ,ρ)-bound for τ_k. ■

2.5 Highest Priority First Scheduling

So far we have only considered the effects of scheduling policies via the scheduling function Π. In this section we do a first step in the direction of defining concrete policies by introducing the class of policies that allows only one task to use the processing unit at the same time. Such policies can be described within the *highest priority first* paradigm (HPF). A straightforward example is the fixed (or static) priority preemptive scheduling policy. There are many other policy where the processing unit can be used by only one task at the same time: earliest deadline first (EDF), first in first out (FIFO), Round Robin (RR), etc. To allow all this policies to be described within HPF, we introduce a concept *priority assignment*, which attributes to each instance of a task a multidimensional priority. This priority assignment enables us to write general properties, which are valid for every scheduling policy of the class. In Section 4, we take advantage of this by deriving results that are common to the timing analysis of all policies based on *time independent priorities.*

2.5.1 Motivation and definition

Consider a processing unit that can be used by only one instance at the same time, i.e. such that

$$\Pi_{k,n}(t) \in \{0,1\} \qquad \text{and} \qquad \sum_{k,n} \Pi_{k,n}(t) \in \{0,1\}. \tag{2.35}$$

In this case, the scheduling policy could be based on a mechanism that designates at each time exactly one instance among those which are active. For this purpose we introduce a *priority assignment* Γ, that gives the *priority* of every instance at any time t:

$$\Gamma_{k,n}(t) \in \mathcal{P}. \tag{2.36}$$

To promote flexibility, the priority space $(\mathcal{P}, \preccurlyeq)$ may be any set of totally ordered elements. For the policies studied in this document we choose as priority space the set of $\mathbb{R}$-valued sequences, with reverse lexicographical order. We write $p \preccurlyeq p'$ to say that priority p is smaller than or equal to priority p'.

For example: $(1,1,4,9,2,\ldots) \preccurlyeq (1,1,3,5,6,\ldots)$. The relation "$p \prec p'$", that is "$p \preccurlyeq p'$ and $p \neq p'$" is defined by

$$(p_1,\ldots,p_n,\ldots) \prec (p'_1,\ldots,p'_n,\ldots) \quad \Longleftrightarrow \quad \exists\, i \vdash \ \ p_i > p'_i \text{ and } p_j = p'_j,\ j < i. \tag{2.37}$$

It can be seen that the smaller the index of a component, the higher its importance, and the smaller a component, the larger its weight. Since the actually needed number of coordinates depends on the context where they are used, we have chosen sequences instead of vectors. The less significant components, which have no impact in a given context will not be written.

A priority assignment must be *decidable* that is such that at any time there is exactly one instance with maximal priority among the currently active instances. This implies that the priorities of the active tasks are totally ordered, but they also have to be different.

Definition 2.25 *A priority assignment is said to be decidable, if*

$$t \in \mathbb{R}_+,\ \tau_{k,n} \neq \tau_{k',n'} \text{ such that } W_{k,n}(t^+) > 0 \text{ and } W_{k',n'}(t^+) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t) \neq \Gamma_{k',n'}(t). \tag{2.38}$$

Definition 2.26 *A non-idling scheduled task process is said to satisfy the* highest priority first *rule (HPF, for short) for a priority assignment* Γ *if for all* $\tau_{k',n'} \neq \tau_{k,n}$ *and* $\forall\, \omega \in \Omega$

$$\Pi_{k,n}(t) = 1 \text{ and } W_{k',n'}(t) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t) \succ \Gamma_{k',n'}(t). \tag{2.39}$$

The question that arises now, is under which assumptions about priority assignments the HPF-rule determines a scheduling policy. This question is not only motivated by theoretical considerations. The answer tells how a policy can be defined that will always work properly, meaning in this context that the processing unit neither idles, i.e. does not execute any instance although tasks are pending, nor tries to execute two instances at the same time, although it should not, see assumption (2.35). We do not know which conditions are necessary, but we know a reasonable sufficient condition:

Definition 2.27 *A priority assignment* Γ *is said to be* piecewise order preserving *if for any finite interval* $[t_0, t_I[$ *there is a finite partition* $t_0 < t_1 < \ldots < t_I$ *such that for all* $\tau_{k,n}$, $\tau_{k',n'}$, $i \in \{1,\ldots,I\}$

$$\Gamma_{k,n}(t_{i-1}) \succ \Gamma_{k',n'}(t_{i-1}) \quad \Rightarrow \quad \forall x \in [t_{i-1}, t_i[\ \ \Gamma_{k,n}(x) \succ \Gamma_{k',n'}(x). \tag{2.40}$$

Consequently, if a priority assignment does not satisfy this definition, then there exists an interval of finite length where the priority order changes infinitely often. No physical processing unit can realize such a scheduling and thus requiring the assignment to be piecewise order preserving is not restrictive for applications. Thus, although this property is probably not necessary and only sufficient it is not a loss of generality to consider only such assignments. Furthermore, the proof of Theorem 2.28 below shows that such a scheduling can be constructed step by step, as a real-world scheduler would.

Theorem 2.28
For any decidable and piecewise priority order preserving assignment Γ *there exists exactly one non-idling scheduling function* Π *that satisfies the HPF-rule, i.e* Γ *specifies a non-idling scheduling policy. The scheduling function and the priority assignment satisfy the following identity:*

$$\forall t \quad \Pi_{k,n}(t) = \mathbb{1}_{[W_{k,n}(t^+)>0]} \cdot \prod_{\tau_{i,j} \neq \tau_{k,n}} (1 - \mathbb{1}_{[W_{i,j}(t^+)>0]} \cdot \mathbb{1}_{[\Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)]}). \tag{2.41}$$

Proof: Remind that a scheduling policy must associate to each task sequence exactly one scheduled task sequence . Let us denote $\mathbb{T}^{\Pi}_{e} \subset \mathbb{T}^{\Pi}_{0}$ the subset of non-idling scheduled task sequences satisfying (2.35). The "e" stands for "exclusive". Thus we have to prove that to each $(a,c,d) \in \mathbb{T}_0$, can be associated exactly one scheduled task sequence $(a,c,d,\pi) \in \mathbb{T}^{\pi}$ which is non-idling and satisfies the HPF rule.

Existence First we show how to construct from an element of $\mathbb{T}_0$ via (2.41) an element of $\mathbb{T}_e^\Pi$, i.e. satisfying the basic assumption right-continuity, (2.10), (2.11) and (2.12) on page 10, the non-idling assumption (2.23) and the "exclusive use" assumption (2.35). Furthermore it must obey the HPF rule (2.39). The construction is by recurrence and based on a sequence of times $\{t_i\}$. Let

$$t_0 = \min\{A_{k,0} \mid k = 1..m\}.$$

The required properties and (2.41) trivially hold on $[0, t_0[$.
Now, assume that on $[0, t_i[$, for some $i \geqslant 0$ the $\Pi_{k,n}$ are defined and satisfy the required properties and also (2.41). Notice that via $W_{k,n}(t^+) = S_{k,n}(0, t^+) - \int_0^t \Pi_{k,n}(u)du$ the workload $W_{k,n}(t^+)$ is defined on $[0, t_i]$, i.e. including t_i, because the value of $\Pi_{k,n}$ at t_i has now effect on the value of the integral - a point being a set of Lebesgue measure zero. We have to distinguish two cases, positive or zero processor load:

(i) $W_{1..m}(t_i^+) > 0$: Since the priority assignment is supposed to be decidable, there exists at t_i a pending instance $\tau_{k,n}$ with a higher priority than any other pending instance, see (2.38), i.e. such that $W_{k,n}(t_i^+) > 0$ and for any $\tau_{k',n'} \neq \tau_{k,n}$

$$W_{k',n'}(t_i^+) > 0 \quad \Rightarrow \quad \Gamma_{k,n}(t_i) \succ \Gamma_{k',n'}(t_i).$$

Thus at t_i, $\tau_{k,n}$ is the task to be executed, according to the HPF rule. Assume we would extend Π for $x \in [t_i, t_i + W_{k,n}(t_i^+)[$ by $\Pi_{k,n}(x) = 1$ and $\Pi_{k',n'}(x) = 0$ if $\tau_{k',n'} \neq \tau_{k,n}$. But as soon as there would be a pending instance with higher priority than $\tau_{k,n}$, this extension would violate the HPF-rule. Thus, let

$$X_i = \{x > t_i \mid \exists\, \tau_{k',n'} \;\vdash\; W_{k',n'}(x^+) > 0,\ \Gamma_{k,n}(x) \prec \Gamma_{k',n'}(x)\ \}$$

be the set of instants where $\Pi_{k,n}(x) = 1$ would not be allowed. Let $x_i = \inf(X_i)$, with $x_i = +\infty$ if the set is empty. By definition $x_i \geqslant t_i$, but $x_i > t_i$ is needed since otherwise, $\Pi_{k,n}(x)$ could not be defined such as to be right-continuous at t_i. The infimum of X_i is the infimum of the times where the priority of some pending instance becomes higher than that of $\tau_{k,n}$. There a two case to be distinguished:

- Instances which are pending at t_i, i.e. with $W_{k',n'}(t_i^+) > 0$, can contribute to X_i if their priority becomes higher than the priority of $\tau_{k,n}$. By (2.40), this can only happen strictly after t_i.
- Instances which are not pending at t_i, can only become pending by being activated (strictly) after t_i.

Now, since by Lemma 2.5, the total number of pending instances in $[t_i, t_i + W_{k,n}(t_i^+)[$ is finite, the infimum of the times $x \in X_i$ is strictly larger than t_i, i.e. $x_i > t_i$. Finally, let

$$t_{i+1} = \min\{t_i + W_{k,n}(t_i^+),\ x_i\}.$$

Since $W_{k,n}(t_i^+) > 0$, we finally have $t_{i+1} > t_i$. Now, for $t \in [t_i, t_{i+1}[$ set $\Pi_{k,n}(t) = 1$ and $\Pi_{k',n'}(t) = 0$ for all $\tau_{k',n'} \neq \tau_{k,n}$. The resulting scheduling functions are right-continuous on $[t_i, t_{i+1}[$. Furthermore (2.39) and (2.41) hold at t_i and also throughout $]t_i, t_{i+1}[$, since neither $\mathbb{1}_{[W_{k,n}(t^+)>0]}$, $\mathbb{1}_{[W_{k',n'}(t^+)>0]}$ nor $\mathbb{1}_{[\Gamma_{k,n}(t^+)\succ\Gamma_{k',n'}(t^+)]}$ change their value. By construction, $\Pi_{i,j}(t) = 1$ implies $W_{i,j}(t) > 0$ on $[t_i, t_{i+1}[$ for any task $\tau_{i,j}$. The definition of the workload (2.14) on page 10, implies (2.10) and (2.11). Furthermore, since only one task is active (2.12) is satisfied, too.

(ii) $W_{1..m}(t_i^+) = 0$: Let $t_{i+1} = \inf\{A_{k,n} > t_i \mid k = 1..m, n \in \mathbb{N}\}$. As above, using Lemma 2.5, we have $t_{i+1} > t_i$. For $t \in [t_i, t_{i+1}[$ and all $\tau_{k,n}$ set $\Pi_{k,n}(t) = 0$. Then also $W_{k,n}(t^+) = 0$ and hence equation (2.39) and also (2.41) will be satisfied for all $t \in [t_i, t_{i+1}[$. By construction $\Pi_{k,n}$ is right-continuous and satisfies (2.10), (2.11) and (2.12).

So far we have proven that the sequence of construction points is strictly increasing: $t_i < t_{i+1}$. To complete the construction by recurrence of Π, it remains to prove that the sequence diverges i.e. that Π is defined for all $t \in \mathbb{R}_+$. Each construction point t_i is either equal to an activation time or date of priority order modification (case where $t_i = y_i$), or it is equal to an execution end (case where $t_i \leqslant y_i$):

$$\forall\, i \quad \exists \tau_{k,n} \vdash (t_i = A_{k,n}) \text{ or } (t_i = E_{k,n}) \text{ or } (\exists \tau_{k',n'} \vdash [\Gamma_{k,n}(t^-) \prec \Gamma_{k',n'}(t^-)] \text{ and } [\Gamma_{k,n}(t) \succ \Gamma_{k',n'}(t)]).$$

In an interval of finite length, the number of activations is finite, by (2.2) on page 7, the number of pending instances and thus of execution ends is finite by (2.17), and the number of priority changes of active instances is finite by (2.40). Thus an interval of finite length contains only a finite number of t_i and thus the sequence must diverge.

It implies furthermore that $\Pi_{k,n}$ is piecewise constant and thus right-continuous for all t. Since (2.41) satisfies the non-idling assumption (2.23) and exclusive-use (2.35), the constructed task sequence is an element of $\mathbb{T}_e^{\Pi}$.

Uniqueness Assume for a task sequence (a, c, d) there would exist two different scheduling functions Π and Π' satisfying (2.39). Let

$$t_0 = \inf\{t \geqslant 0 \mid \exists\ \tau_{k,n} : \Pi_{k,n}(t) \neq \Pi'_{k,n}(t)\}.$$

By the right-continuity of Π and (2.35) on page 21, we have also $\Pi_{k,n}(t_0) \neq \Pi'_{k,n}(t_0)$. For all tasks $\tau_{i,j}$ $\Pi_{i,j}(t) = \Pi'_{i,j}(t)$, $t < t_0$, but

$$W_{i,j}(t) = W'_{i,j}(t) \qquad \forall t \leqslant t_0, \tag{2.42}$$

because the workload depends on Π through an integral, see (2.14). Two symmetric cases arise:

1. $\Pi_{k,n}(t_0) = 1$ and $\Pi'_{k,n}(t_0) = 0$
2. $\Pi_{k,n}(t_0) = 0$ and $\Pi'_{k,n}(t_0) = 1$.

We only need to study the first. By (2.16), $\Pi_{k,n}(t_0) = 1$ implies that $W_{k,n}(t_0) > 0$ and $W'_{k,n}(t_0) > 0$. On one hand $\Pi_{k,n}(t_0) = 1$ implies $\Gamma_{k,n}(t_0) \succ \Gamma_{i,j}(t_0)$ for any other pending instance $\tau_{i,j}$, see (2.41). On the other hand, $\Pi'_{k,n}(t_0) = 0$ implies that there is a task $\tau_{i,j}$ with $\Pi_{i,j}(t_0) = 1$ because of the non-idling assumption. The HPF-rule implies then $\Gamma_{k,n}(t) \prec \Gamma_{i,j}(t)$, which is a contradiction. ■

A simple example of a piecewise order preserving assignment is $\Gamma_{k,n}(t) = (k, n)$, which produces the fixed priority preemptive scheduling (Section 4.2). It is decidable, since never two instance have the same priority and order preserving, because the order is always the same. If we had chosen $\Gamma'_{k,n}(t) = k + 1, n + 13$, the resulting schedule would have been the same. More generally, different assignments can be equivalent in the sense that they produce the same schedule.

Definition 2.29 *Two priority assignments* Γ *and* Γ, *with values in priority spaces* $(\mathcal{P}_1, \overset{1}{\succ})$ *and* $(\mathcal{P}_2, \overset{2}{\succ})$ *respectively, are said to be* equivalent *if any pair of task* $\tau_{k,n} \neq \tau_{i,j}$ *have at any time* t *the same priority order:*

$$\Gamma_{k,n}(t) \overset{1}{\succ} \Gamma_{i,j}(t) \Leftrightarrow \Gamma'_{k,n}(t) \overset{2}{\succ} \Gamma'_{i,j}(t). \tag{2.43}$$

Proposition 2.30 *For a given task process, two equivalent priority assignments produce the same scheduled task process.*

Proof: By contradiction, using (2.41). ■

The fact that two different assignments can implement the same policy is actually a very useful fact for the construction and analysis of complex policies.

2.5.2 Some formulas

Consider the set of instances $\mathcal{H}_{k,n}(t)$ having a higher priority that $\tau_{k,n}$ at some epoch $t \in [A_{k,n}, E_{k,n}[$ where it is pending:

$$\mathcal{H}_{k,n}(t) \overset{\mathsf{def}}{=} \{\tau_{i,j} \mid \Gamma_{i,j}(t) \succ \Gamma_{k,n}(t)\}. \tag{2.44}$$

Let the corresponding set of instances with lower priority be

$$\mathcal{L}_{k,n}(t) \overset{\mathsf{def}}{=} \{\tau_{i,j} \mid \Gamma_{i,j}(t) \prec \Gamma_{k,n}(t)\}. \tag{2.45}$$

The only instance being in neither of the two is the considered instance itself, because of the decidability assumption about the priority assignment. Notice that these sets only take the priority structure into account, independently of activation times. Such a set can contain instances that have already been executed before t or that will be activated far after t. Recall the execution end formula:

$$E_{k,n} = \min\{t > A_{k,n} \mid W_{k,n}(t) = 0\}.$$

It is based on the workload, which in general is the result of task activations and their scheduling. In order to derive response time bounds, it is necessary to introduce WAF, because the worst-case characteristics of tasks are given in terms of majorizing WAF, see Section 3. Workloads and WAF's are related via the scheduling function Π, see equation (2.14).
One could try the following approach: since $W_{k,n}(t) > 0$ implies $\Pi_{\mathcal{L}_{k,n}(t)}(t) = 0$, we have $1 = \Pi_{\mathcal{H}_{k,n}(t)}(t) + \Pi_{k,n}(t)$. But the set of instances with a higher priority might depend on the considered instant t. An instance that is in $\mathcal{H}_{k,n}(t_1)$ but not in $\mathcal{H}_{k,n}(t_2)$ for $t_1 < t_2$ might completely execute during $[t_1, t_2[$, or only partially or not at all. Thus, it is not possible to characterize $\Pi_{\mathcal{H}_{k,n}(t)}(t)$ in a general way.
Another approach is needed to work around this difficulty: since $W_{\mathcal{H}_{k,n}(E_{k,n})}(E_{k,n}) = 0$, we have $W_{k,n}(t) = 0 \Leftrightarrow W_{1..m}(t) = W_{\mathcal{L}_{k,n}(t)}(t)$. Thus, the execution time formula can be rewritten as

$$E_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{1..m}(t) = W_{\mathcal{L}_{k,n}(t)}(t)\}, \tag{2.46}$$

which means that the execution end is the first epoch after the activation time, where the total amount of pending work is equal to the pending work of instances with lower priority.

Now, for $t \in]A_{k,n}, E_{k,n}[$, $W_{k,n}(t) > 0$ implies $W_{1..m}(t) > 0$. Furthermore, because $\Pi_{k,n}(t) = 0$, the non-idling assumption implies then $\Pi_{\mathcal{H}_{k,n}(t)}(t) + \Pi_{\mathcal{L}_{k,n}(t)}(t) = 1$. Using this with the general formula

$$W_{k,n}(t_2) = W_{k,n}(t_1) + S_{k,n}(t_1, t_2) - \int_{t_1}^{t_2} \Pi_{k,n}(u)du \tag{2.47}$$

derived from (2.14), leads to

$$W_{1..m}(t) = W_{1..m}(A_{k,n}) + S_{1..m}(A_{k,n}, t) - (t - A_{k,n}).$$

Thus, the seeked execution formula making the WAF appear is

$$E_{k,n} = \min\{t > A_{k,n} \mid W_{1..m}(A_{k,n}) + S_{1..m}(A_{k,n}, t) + A_{k,n} = t + W_{\mathcal{L}_{k,n}(t)}(t)\}. \tag{2.48}$$

For the response time we obtain, by applying Lemma A.5:

$$R_{k,n} = \min\{x > 0 \mid W_{1..m}(A_{k,n}) + S_{1..m}(A_{k,n}, A_{k,n} + x) = x + W_{\mathcal{L}_{k,n}(A_{k,n}+x)}(A_{k,n} + x)\}. \tag{2.49}$$

These formulas still contain workloads, which can only be transformed under further assumptions.

The execution begin formula can be transformed in a similar way. From (2.20) on page 12 and (2.41) on page 22 it follows that

$$B_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{\mathcal{H}_{k,n}(t)}(t^+) = 0\}. \tag{2.50}$$

It means that the execution begin is the first time after $A_{k,n}$ where the workload of instances with higher priority becomes zero.

Since $\Pi_{k,n}(t) = 0$ for $t \in [A_{k,n}, B_{k,n}[$ and $W_{k,n}(t) = C_{k,n}$ the same formula also reads

$$B_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{1..m}(t^+) = C_{k,n} + W_{\mathcal{L}_{k,n}(t)}(t^+)\}. \tag{2.51}$$

Thus $B_{k,n}$ can also be seen as the first epoch after $A_{k,n}$ where the total amount of pending work is equal to the execution time of the instance and the pending work of instances with lower priority. Replacing $W_{1..m}(t^+)$ using (2.47), and (2.15) finally gives

$$B_{k,n} = \min\{t \geqslant A_{k,n} \mid W_{1..m}(A_{k,n}) + S_{1..m}(A_{k,n}, t^+) + A_{k,n} = t + C_{k,n} + W_{\mathcal{L}_{k,n}(t)}(t^+)\}. \tag{2.52}$$

3 Examples of tasks

Several kinds of tasks have been considered in the literature, in general with increasing complexity to model more and more accurately real-world tasks. In this section we give their corresponding definition in the framework of task processes. Notice that here, defining a task means: give the collection of conditions that discriminate in the set of all possible task sequences $\mathbb{T}$ those sequences which are in accordance with the task's behavior. We shall do this for the types of tasks we have encountered in the literature, and new ones combining their features. For each type of task we give a family of MWAF, which will be needed for the *timing analysis*.

We distinguish independent tasks and dependent tasks. By "independence" we mean that the defining assumptions do not involve more than one task at the same time. Examples are the *periodic tasks* which have been extended to *sporadically periodic task* to account for inter-activation time patterns, and to multi-frame tasks to account for execution time patterns. As synthesis of both, we propose *sporadic multi-frame tasks*. Examples of dependent tasks are *transactions* and tasks defined by a directed acyclic graph. These can be seen as high level tasks consisting of dependent sub-tasks. Another example are tasks with *mode change*.

It can also be noticed that by the use of statistical inference, WAF's allow even to characterize tasks whose internal mechanisms are not known, see Section 3.4.

We will see that for a (sporadically) periodic task, a unique MWAF exists which is based on the worst-case activation pattern which moreover is realized by the task. But a unique release pattern which is worse than any other, does not necessarily exist. In such a case one can either divide the set of all possibilities patterns into subsets, such that for each subset a worst pattern exists, so that a family of MWAF can be found or one can use a unique MWAF which is then a conservative bound. Using a family of MWAF's, increases the accuracy of the bound, but the computational complexity of the resulting analysis increases too. A tradeoff might be necessary to keep the analysis tractable, see Section 4.6.3.

Timing analysis assumes that the temporal characteristics of task are known. To be concretely applicable one must be able to determine these characteristics for a real-world tasks. Cycle times are often chosen by the designer, see for example [14]. The tasks execution times depend on the speed of the processing unit and also of the code that is actually executed when the task is invoked. In practice, worst-case execution times (WCET) of tasks are often estimated with the help of a profiler, although analytical approaches to *worst case execution time analysis* do exist, see for example [25].

3.1 Independent tasks

In Section 3.1.4, we define *sporadically periodic multi-frame tasks* as synthesis of two well known types of tasks and derive (deadline based) MWAF's. These tasks are characterized by two aspects. On one hand, exact MWAF's of the whole task set can be derived by examining each task independently of all others, because their are no relations between them. On the other hand, all instances of the same task have the same relative deadline, implying that (exact) response time bounds can be used for sufficient (and necessary) feasibility tests. If instances of the same task can have different relative deadlines, as with *generalized multi-frame tasks*, then subtasks must be considered (not always, but in general), see Section 3.2.

3.1.1 Periodic tasks

Periodic tasks are at the basis of the seminal paper [22] by Liu & Leyland. In discrete time, a task $\tau_k \in \mathcal{T}$ is periodic with period $T_k \in \mathbb{N}$, constant execution time $C_k \in \mathbb{N}$ and constant relative deadline $\overline{D}_k$, if

$$T_{k,n} = T_k \quad \text{and} \quad C_{k,n} = C_k \quad \overline{D}_{k,n} = \overline{D}_k \qquad \forall n \in \mathbb{N}. \tag{3.1}$$

Consider a set of periodic tasks. If at least one task, τ_k say, can start with different initial offsets, which are given by the first activation time $A_{k,0}$, then several trajectories are to be considered. The n^{th} activation of $\tau_{k,n}$ satisfies $A_{k,n} = A_{k,0} + n \cdot T_k$. If for each task, the possible initial offsets are $A_{k,0} \in \{0, 1, \ldots, T_k - 1\}$ and if all offset combinations are possible then Ω consists of $|\Omega| = T_1 \cdot \ldots \cdot T_m$ different trajectories, which are characterized by the initial offsets $A_{1,0}, A_{2,0}, \ldots A_{m,0}$. In this example a task process (A, C, D) is equivalent to a collection of *concrete task sets*, as defined in [18]. A concrete task set is a set of periodic tasks with specified vector of initial offsets. To every possible vector corresponds a trajectory of the task process.

It is often assumed that a real-world task τ_k has a known smallest cycle time T_k, a longest execution time C_k and fixed relative deadline $\overline{D}_k$, i.e. $\forall n \in \mathbb{N}$

$$T_{k,n} \geqslant T_k > 0 \tag{3.2}$$

$$C_{k,n} \leqslant C_k \tag{3.3}$$

$$\overline{D}_{k,n} = \overline{D}_k. \tag{3.4}$$

As in [28], we refer to these kind of tasks as *sporadic periodic tasks*. Notice that the periodic task with constant execution time C_k and period T_k does also satisfy this assumption. As a result their MWAF is the same. Thus from the point of view of timing analysis, describing a task as sporadic task (even if it has more complex behavior) is like using a periodic task as model for the real-world task.

Consider an interval $[u, u + x[$. Let A_{k,n_0} be the first activation after or at u and A_{k,n_1} be the last before $u + x$, with deadline smaller or equal to $u + d$. Let $i = n_1 - n_0 + 1$ be the number of instances taken into account by $S_k(u, u + x, u + d)$. The amount of work of these instances $\sum_{n=n_0}^{n_1} C_{k,n}$ is thus bounded by $i \cdot C_k$. Assuming there is a least one activation, the last activation takes place at

$$A_{k,n_1} = A_{k,n_0} + \sum_{n=n_0}^{n_1-1} T_{k,n} \geqslant u + (i-1) \cdot T_k. \tag{3.5}$$

On one hand, $A_{k,n_1} < u + x$ and on the other $A_{k,n_1} + \overline{D}_k \leqslant u + d$. Thus, $i \leqslant 1 + x/T_k$, implying $i \leqslant \lceil x/T_k \rceil$, because in general $\lceil x \rceil - 1 < x$. Furthermore, $i < 1 + (d - \overline{D}_k)/T_k$, implying $i \leqslant 1 + \lfloor (d - \overline{D}_k)/T_k \rfloor$, because in general $\lfloor x \rfloor \leqslant x$. Hence the demand of the task is bounded by

$$\widehat{S}_k(x, d) = \min\left(\left\lceil \frac{x}{T_k} \right\rceil, \left\lfloor \frac{d - \overline{D}_k}{T_k} \right\rfloor + 1 \right) \cdot C_k \qquad d \geqslant \overline{D}_k,\ x \geqslant 0. \tag{3.6}$$

For $d \to +\infty$, we obtain $\widehat{S}_k(x) = \lceil x/T_k \rceil \cdot C_k$. This is exactly the demand of a task being activated first at $\hat{A}_{k,0} = 0$ and then at $\hat{A}_{k,n} = n \cdot T_k$ with execution times $\widehat{C}_{k,n} = C_k$. Thus we recognize the *critical instance* [22], where all tasks are activated at the same time and with the shortest inter-activation times.

Since for $x \in \mathbb{R}$, $\lceil x \rceil < x + 1$, we can deduce from (3.6) a (σ,ρ)-bound with parameters

$$\sigma_k = C_k \qquad\qquad \rho_k = C_k/T_k. \tag{3.7}$$

3.1.2 Sporadically periodic tasks

Sporadically periodic tasks have been introduced by Tindell in his thesis [32] to model more accurately the inter-activation patterns of bursty tasks arising in applications. A burst consists of at most N_k activations with minimal inter-activation time $T_k^{(1)}$ and maximal execution time C_k. The starting times of two bursts are separated by at least $T_k^{(1)}$, the so-called *inner cycle time*. The time $T_k^{(2)}$ is called *outer cycle time*. It is assumed that $N_k \cdot T_k^{(1)} \leqslant T_k^{(2)}$ in order to guarantee that the time between the last activation of the previous burst and the first activation of the next burst are separated by at least

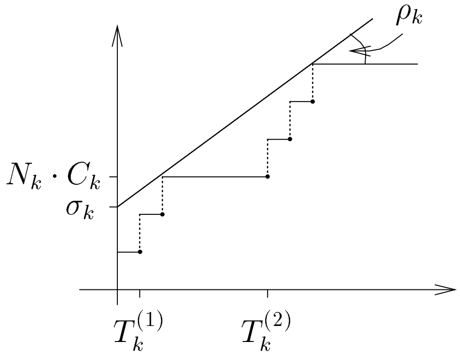


Figure 3: Afin bound of a sporadically periodic task.

$T_k^{(1)}$. We define sporadically periodic task in our model axiomatically by the following conditions. For all $n \in \mathbb{N}$:

$$C_{k,n} \leqslant C_k \tag{3.3}$$

$$\overline{D}_{k,n} = \overline{D}_k \tag{3.4}$$

$$A_{k,n+1} \geqslant A_{k,n} + T_k^{(1)} \tag{3.8}$$

$$A_{k,n+N_k} \geqslant A_{k,n} + T_k^{(2)}. \tag{3.9}$$

Conditions (3.3), (3.8), and (3.9) determine the subset of all task sequences in $\mathbb{T}$ that can be realized by sets of sporadically periodic tasks.

The (deadline based) MWAF for this kind of task are particular cases of Proposition 3.1, below. They are equivalent to bounds on "the maximal interference from higher priority tasks", used in [31] under FPP and in [26] under EDF.

The majorizing WAF, illustrated in Figure 3 together with a (σ,ρ)-bound, is given by

$$\widehat{S}_k(x) = C_k \cdot \left(\left(\left\lceil \frac{x}{T_k^{(2)}} \right\rceil - 1 \right) \cdot N_k + \left\lceil \frac{x - (\lceil x/T_k^{(2)} \rceil - 1) \cdot T_k^{(2)}}{T_k^{(1)}} \right\rceil \wedge N_k \right), \tag{3.10}$$

see Proposition 3.1 below. Proposition 2.23 with $y = T_k^{(2)}$ gives

$$\rho_k = \frac{N_k \cdot C_k}{T_k^{(2)}} \tag{3.11}$$

as slope of the (σ,ρ)-bound and $\sigma_k = N_k \cdot C_k$ as additive constant. As can be seen in Figure 3, it is possible to find a smaller value:

$$\sigma_k = N_k \cdot C_k \cdot \frac{T_k^{(2)} - (N_k - 1) \cdot T_k^{(1)}}{T_k^{(2)}}$$

which characterizes better the burstiness of the task. For the stability condition however it is only important to have the smallest possible slope.

A sporadically periodic task also satisfies conditions (3.2) and thus (3.6) is also a majorizing WAF for the task, with $T_k = T_k^{(1)}$. But the more important the difference between $N_k \cdot T_k^{(1)}$ and $T_k^{(2)}$, the less accurate (3.6) is as a bound for a sporadically periodic task and the looser response time bound that could be derived. Furthermore, even if the slope given by (3.11) is smaller than 1, i.e. the task does not overload by itself the processing unit, it is possible that $C_k/T_k^{(1)} > 1$, meaning that in that case, using a periodic task to bound the worst case behavior of a sporadically periodic task does not even allow to compute finite bounds.

3.1.3 Multi-frame tasks

Multi-frame tasks have been introduced by Mok and Chen in [24] to represent more accurately task that decode MPEG video. Under the MPEG standard, successive frames are not all encoded in the same way. If intra-frame encoding is used, the encoded frame contains all informations to entirely rebuild the picture, while with inter-frame encoding informations about the previous frame are used, which reduces the size of the encoded frame, but also requires the previous frame for being decoded. The size of encoded frames has an impact on the execution times of the corresponding decoding task. The MPEG standard foresees a certain pattern of intra- and inter-frame encoding. In [24] it is shown that the resulting execution time patterns of the real-world task are more accurately captured by so called *multi-frame tasks*. It is shown that the multiframe task model improves the utilization bound of the processor in comparison to the usual periodic task model. The authors also give feasibility conditions based on an appropriate critical instant and show that the rate-monotonic priority assignment is optimal for multiframe tasks. The latter result requires the assumption of relative deadlines being equal to minimal inter-activation times P_k. In this section, we define multiframe tasks in the framework of our model, but without restriction on relative deadlines. As a result of our timing analysis (developed in later sections) it is possible to compute response time bounds for multiframe tasks with general deadlines and under any of the scheduling policies considered in this document. An optimal priority assignment under FPP can be found by Audsley's algorithm, see Section 4.2.5.

A multi-frame task is characterized by a vector of execution times $\vec{C}_k = (C_k^0, C_k^1, \ldots, C_k^{M_k-1})$ and a minimal inter-activation time P_k. Axiomatically they are defined by

$$C_{k,n} = C_k^{n \bmod M_k} \tag{3.12}$$

$$T_{k,n} \geqslant P_k. \tag{3.13}$$

In a similar way as for periodic tasks, one obtains $S_k(u, u+x) \leqslant \sum_{n=n_0}^{\lceil x/P_k \rceil} C_{k,n}$. The first of these execution times, C_{k,n_0}, could be equal to any of the components of $\vec{C}_k$. Since the following execution times only depend on C_{k,n_0}, a family of MWAF is $\mathcal{S}_k = \{\widehat{S}_k^{(i_k)}(x,d) \mid i_k = 0, \ldots, M_k - 1\}$, where

$$\begin{aligned}
\widehat{S}_k^{(i_k)}(x,d) &= \sum_n C_k^{(n+i_k) \bmod M_k} \cdot \mathbb{1}_{[n \cdot P_k < x]} \cdot \mathbb{1}_{[n \cdot P_k + \overline{D}_k \leqslant d]} \\
&= \left\lfloor \frac{\widehat{N}_k(x,d)}{M_k} \right\rfloor \cdot \sum_{j=0}^{M_k-1} C_k^j + \sum_{j=0}^{\widehat{N}_k(x,d) \bmod M_k - 1} C_k^{(i_k+j) \bmod M_k}
\end{aligned}$$

with

$$\widehat{N}_k(x,d) = \min(\lceil x/P_k \rceil, \lfloor (d - \overline{D}_k)/P_k \rfloor + 1).$$

This family consists in one WAF for each possible case $C_{k,n_0} = C_k^{i_k}$. In order to obtain a single MWAF that is valid in every case one must take the maximum over all possibilities:

$$\widehat{S}_k^*(x) = \max_{i=0,\ldots,M_k-1} \widehat{S}_k^{(i)}(x) = \left\lfloor \frac{\lceil x/P_k \rceil}{M_k} \right\rfloor \cdot \check{C}_k^{M_k-1} + \check{C}_k^{(\lceil x/P_k \rceil \bmod M_k)-1} \tag{3.14}$$

which leads to define a vector $(\check{C}_k^0, \ldots, \check{C}_k^{M_k-1})$ of maximal partial sums of execution times

$$\check{C}_k^h \stackrel{\mathsf{def}}{=} \max_{j=0,\ldots,M_k-1} \sum_{l=j}^{j+h} C_k^{l \bmod M_k}. \tag{3.15}$$

A task with $\sum_{j=0}^{i} C_k^j = \check{C}_k^i$ is called *accumulatively monotonic* in [24]. In that case, a unique MWAF $\widehat{S}_k^*(x)$ (3.14) represents the worst case release pattern, otherwise it is a conservative bound.

3.1.4 Sporadically periodic multi-frame tasks

In this section we define in our model *sporadic multi-frame tasks* as synthesis of the different types of task described so far. We derive the corresponding MWAF's which can then be used to derive response time bounds under the different scheduling policies, which are treated in later sections.

Let $\vec{C}_k = (C_k^0, C_k^1, \ldots, C_k^{M_k-1})$ be the vector of execution times. As for sporadically periodic tasks let $T_k^{(1)}$ be the inner and $T_k^{(2)}$ the outer period, satisfying $N_k \cdot T_k^{(1)} \leqslant T_k^{(2)}$. A sporadic multi-frame task is a task that satisfies

$$\overline{D}_{k,n} = \overline{D}_k \tag{3.4}$$

$$A_{k,n+1} \geqslant A_{k,n} + T_k^{(1)} \tag{3.8}$$

$$A_{k,n+N_k} \geqslant A_{k,n} + T_k^{(2)} \tag{3.9}$$

$$C_{k,n} \leqslant C_k^{n \bmod M_k}. \tag{3.16}$$

In the particular case where $N_k \cdot T_k^{(1)} = T_k^{(2)}$, (3.9) is actually implied by (3.8) and has then no additional effect.

Proposition 3.1 *A family of exact MWAF's for a sporadically periodic multiframe task is given by* $\mathcal{S}_k = \{\widehat{S}_k^{(i_k)}(x,d) \mid i_k = 0, \ldots, M_k - 1\}$, *where*

$$\widehat{S}_k^{(i_k)}(x,d) = \left\lfloor \frac{\widehat{N}_k(x,d)}{M_k} \right\rfloor \cdot \sum_{j=0}^{M_k-1} C_k^j + \sum_{j=0}^{\widehat{N}_k(x,d) \bmod M_k - 1} C_k^{(i_k+j) \bmod M_k}$$

with

$$\widehat{N}_k(x,d) = \min\Big(I_k\big(\, x, \lceil x/T_k^{(2)} \rceil \,\big),\ I_k\big(\, d - \overline{D}_k, \lfloor (d - \overline{D}_k)/T_k^{(2)} \rfloor + 1 \,\big) \Big),$$

and

$$I_k(t,j) = j \cdot N_k + \left\lceil \frac{t - j \cdot T_k^{(2)}}{T_k^{(1)}} \right\rceil \wedge N_k.$$

A unique MWAF is given by

$$\widehat{S}_k^*(x,d) = \left\lfloor \frac{\widehat{N}_k(x,d)}{M_k} \right\rfloor \cdot \check{C}_k^{M_k-1} + \check{C}_k^{(\widehat{N}_k(x,d) \bmod M_k)-1} \qquad \textit{with} \quad \check{C}_k^h \stackrel{\mathsf{def}}{=} \max_{j=0,\ldots,M_k-1} \sum_{l=j}^{j+h} C_k^{l \bmod M_k}.$$

Proof: Assume an interval $[u, u+x[$ contains at least one release of a task τ_k. Let A_{k,n_0} be its first release and A_{k,n_1} its last in this interval, satisfying also $A_{k,n_1} + \overline{D}_k \leqslant u + d$. There are thus $i = n_1 - n_0 + 1$ activations of τ_k in the interval and we have:

$$S_k(u, u+x, u+d) = \sum_{n \in \mathbb{N}} C_{k,n} \cdot \mathbb{1}_{[u \leqslant A_{k,n} < u+x]} \cdot \mathbb{1}_{[D_{k,n} \leqslant u+d]} = \sum_{n=n_0}^{n_0+i-1} C_{k,n},$$

which is increasing in i. The first step consists in deriving a bound on i. We have

$$A_{k,n_1} = A_{k,n_0} + \sum_{n=n_0}^{n_1-1} T_{k,n}.$$

Starting from the first activation, we subdivide the releases into groups of N_k consecutive releases. The last group may contain less than N_k releases. Let $j_1 = \lceil i/N_k \rceil - 1$. Since for $x \in \mathbb{R}$, $\lceil x \rceil - 1 < x$, we have $j_1 \cdot N_k < i$. It implies that $n_0 + j_1 \cdot N_k < n_1 + 1$ and thus

$$n_0 + j_1 \cdot N_k \leqslant n_1,$$

a property that will be needed below. From (3.9) we can derive

$$A_{k,n_0+j_1\cdot N_k} \geqslant A_{k,n_0} + j_1 \cdot T_k^{(2)}.$$

Let $j_2 = i - j_1 \cdot N_k$ be the number of releases remaining after the first j_1 groups of N_k releases. There is at least one such instance because $j_2 = i - j_1 \cdot N_k = i - (\lceil i/N_k \rceil - 1) \cdot N_k > i - (i/N_k) \cdot N_k = 0$ and thus $j_2 \geqslant 1$. The first of them is released at $A_{k,n_0+j_1\cdot N_k}$. Furthermore, since for $x \in \mathbb{R}$, $\lceil x \rceil \geqslant x$, we have $j_1 = \lceil i/N_k \rceil - 1 \geqslant i/N_k - 1$, implying $i \leqslant (j_1 + 1) \cdot N_k$ and therefore $j_2 = i - j_1 \cdot N_k \leqslant N_k$. Thus $1 \leqslant j_2 \leqslant N_k$. We have

$$A_{k,n_1} = A_{k,n_0+j_1\cdot N_k} + \sum_{n=n_0+j_1\cdot N_k}^{n_1-1} T_{k,n}.$$

Condition (3.8) applied to the remaining $n_1 - 1 - (n_0 + j_1 \cdot N_k) + 1 = j_2 - 1$ releases implies

$$A_{k,n_1} = A_{k,n_0+j_1\cdot N_k+j_2} \geqslant A_{k,n_0} + j_1 \cdot T_k^{(2)} + (j_2 - 1) \cdot T_k^{(1)}.$$

Now, on one hand $A_{k,n_1} < u + x$ and on the other $A_{k,n_1} + \overline{D}_k < u + d$. Since $A_{k,n_0} \geqslant u$, it implies

$$j_1 \cdot T_k^{(2)} + (j_2 - 1) \cdot T_k^{(1)} \quad \left\{ \begin{array}{ll} < x & (i) \\ \leqslant d - \overline{D}_k & (ii). \end{array} \right.$$

The left hand side is a function of i through $j_1 = j_1(i)$:

$$F(i) \stackrel{\text{def}}{=} j_1(i) \cdot T_k^{(2)} + (i - j_1(i) \cdot N_k - 1) \cdot T_k^{(1)} \quad \left\{ \begin{array}{ll} < x & (i) \\ \leqslant d - \overline{D}_k & (ii). \end{array} \right. \tag{3.17}$$

In order to derive the maximal possible value for i we prove that F is increasing in i. First notice that $j_1(i+1) = j_1(i) + \mathbb{1}_{[N_k|i]}$, meaning that j_1 is increasing in i. We can derive then

$$F(i+1) = F(i) + T_k(1) + (T_k(2) - N_k \cdot T_k(1)) \cdot \mathbb{1}_{[N_k|i]},$$

implying $F(i+1) \geqslant F(i) + T_k(1)$, since $T_k(2) \geqslant N_k \cdot T_k(1)$. Now, since F is increasing in i we only have to find the largest value for i, for which (3.17) holds. We have to treat separately the two inequalities:

(i) Because of the term $j_1(i) \cdot T_k^{(2)}$, (3.17) can only hold if $j_1(i) < x/T_k^{(2)}$. Then, since $x < y$ for $x \in \mathbb{N}, y \in \mathbb{R}$ implies $x \leqslant \lceil y \rceil - 1$ the largest value is given by

$$j_1(i) \leqslant \left\lceil \frac{x}{T_k^{(2)}} \right\rceil - 1.$$

From (3.17) we deduce

$$i < \frac{x - j_1(i) \cdot T_k^{(2)}}{T_k^{(1)}} + 1 + j_1(i) \cdot N_k \leqslant \left\lceil \frac{x - j_1(i) \cdot T_k^{(2)}}{T_k^{(1)}} \right\rceil + j_1(i) \cdot N_k,$$

using again $x + 1 \leqslant \lceil x \rceil$, $\forall\, x \in \mathbb{R}_+$. Since $j_2 \leqslant N_k$, we have on the other hand $i \leqslant N_k + j_1^{max}(i) \cdot N_k$ and thus we finally obtain

$$i \leqslant I_k(x,\, j_1(i)) \stackrel{\text{def}}{=} j_1(i) \cdot N_k + \left\lceil \frac{x - j_1(i) \cdot T_k^{(2)}}{T_k^{(1)}} \right\rceil \wedge N_k. \tag{3.18}$$

(ii) Because of the term $j_1(i) \cdot T_k^{(2)}$, (3.17) can only hold if $j_1(i) \leqslant (d - \overline{D}_k)/T_k^{(2)}$. Then, since $x \leqslant y$ for $x \in \mathbb{N}, y \in \mathbb{R}$ implies $x \leqslant \lfloor y \rfloor$ the largest value is given by

$$j_1(i) \leqslant \left\lfloor \frac{d - \overline{D}_k}{T_k^{(2)}} \right\rfloor .$$

By analogy we derive from (3.17) in this case $i \leqslant I_k(d - \overline{D}_k,\ j_1(i))$.

Thus, we finally obtain

$$i = n_1 - n_0 + 1 \leqslant \widehat{N}_k(x,d) \overset{\mathsf{def}}{=} \min(\quad I_k(\ x,\ \lceil x/T_k^{(2)} \rceil - 1\ ), \quad I_k(\ d - \overline{D}_k,\ \lfloor (d - \overline{D}_k)/T_k^{(2)} \rfloor\ ) \quad).$$

Having derived a bound on the number of activations, we still have to bound the sum of their execution times. Notice that C_{k,n_0} can be any element $C_k^{i_k}$ of $\vec{C}_k$. Hypothesis (3.16) implies in particular that any sequence of M_k consecutive execution times is bounded by $\sum_{j=0}^{M_k-1} C_k^j$. The number of activations $\widehat{N}_k(x,d)$ constitute at most $\lfloor \widehat{N}_k(x,d)/M_k \rfloor$ complete bundles of M_k activations and a rest of $\widehat{N}_k(x,d) \bmod M_k$. :

$$\begin{aligned} S_k(u, u+x, u+d) &= \sum_{j=0}^{n_1-n_0} C_{k,n_0+j} \leqslant \sum_{j=0}^{\widehat{N}_k(x,d)-1} C_k^{(i_k+j) \operatorname{mod} M_k} \\ &= \left\lfloor \frac{\widehat{N}_k(x,d)}{M_k} \right\rfloor \cdot \sum_{j=0}^{M_k-1} C_k^j + \sum_{j=0}^{(\widehat{N}_k(x,d) \operatorname{mod} M_k)-1} C_k^{(i_k+j) \operatorname{mod} M_k} \\ &\overset{\mathsf{def}}{=} \widehat{S}_k^{(i_k)}(x,d). \end{aligned}$$

So far we have derived a family of MWAF's. It remains to show that these bounds are exact, i.e. that for each i_k there is a trajectory $\omega \in \Omega$ and a time $u \in \mathbb{R}_+$ such that $S_k(u, u+x, u+d; \omega) = \widehat{S}_k^{(i_k)}(x,d)$. The trajectory ω with

$$\begin{aligned} C_{k,n}(\omega) &= C_k^{n \operatorname{mod} M_k} \quad \forall\, n \in \mathbb{N} \\ A_{k,n}(\omega) &= \begin{cases} n \cdot T_k^{(2)} & \text{if } n < i_k \\ \widehat{A}_{k,n-i_k} & \text{if } n \geqslant i_k \end{cases} \end{aligned}$$

satisfies the assumption that define a sporadically periodic multiframe task, and thus is part of $(A,C,D)(\Omega)$, the set of task sequences of the process. Of course, $u = i_k \cdot T_k^{(2)}$. Notice that we have chosen the outer period $T_k^{(2)}$ as inter-activations at the beginning of the trajectory. It is a simple way to ensure that on the interval $[0, i_k \cdot T_k^{(2)}]$, the assumption are satisfied.

It remains to derive a unique MWAF. Since

$$\sum_{j=0}^{(\widehat{N}_k(x,d) \operatorname{mod} M_k)-1} C_k^{(i_k+j) \operatorname{mod} M_k} = \sum_{j=i_k}^{i_k+(\widehat{N}_k(x,d) \operatorname{mod} M_k)-1} C_k^{(j) \operatorname{mod} M_k} \leqslant \check{C}_k^{(N_k \operatorname{mod} M_k)-1},$$

and $\widehat{N}_k(x,d)$ is independent of i_k, $\widehat{S}_k^*(x,d)$ is derived.

■

Notice that the activation pattern, underlying to $\widehat{S}_k^{(i_k)}(x,d)$ is:

$$\widehat{A}_{k,n} = \lfloor n/N_k \rfloor \cdot T_k^{(2)} + (n \bmod N_k) \cdot T_k^{(1)} \tag{3.19}$$

$$\widehat{C}_{k,n} = C_k^{(i_k+n) \operatorname{mod} M_k} \tag{3.20}$$

$$\widehat{\overline{D}}_{k,n} = \overline{D}_k. \tag{3.21}$$

Proposition 3.2 *The smallest slope ρ_k of a (σ,ρ)-bound for a sporadically periodic task is given by*

$$\rho_k = \frac{N_k \cdot \check{C}_k^{M_k-1}}{M_k \cdot T_k^{(2)}}.$$

Proof: Let $\widehat{S}_k^{(i_k)}(x) = \lim_{d\to\infty} \widehat{S}_k^{(i_k)}(x,d)$ be the non deadline based version, with $\widehat{N}_k(x) = \lim_{d\to\infty} \widehat{N}_k(x,d)$. We intend do derive

$$\hat{\rho}_k(i_k) = \lim_{x\to\infty} \frac{\widehat{S}_k^{(i_k)}(x)}{x}.$$

We have

$$\left\lfloor \frac{\widehat{N}_k(x)}{M_k} \right\rfloor \cdot \check{C}_k^{M_k-1} \leqslant \widehat{S}_k^{(i_k)}(x) \leqslant \left\lfloor \frac{\widehat{N}_k(x)}{M_k} \right\rfloor \cdot \check{C}_k^{M_k-1} + \check{C}_k^{M_k-1}.$$

Since in general $t-1 \leqslant \lfloor t \rfloor \leqslant t$,

$$\frac{\widehat{N}_k(x)}{M_k} \cdot \check{C}_k^{M_k-1} - \check{C}_k^{M_k-1} \leqslant \widehat{S}_k^{(i_k)}(x) \leqslant \frac{\widehat{N}_k(x)}{M_k} \cdot \check{C}_k^{M_k-1} + \check{C}_k^{M_k-1}.$$

First we will bound $\widehat{N}_k(x)$. Since in general $\lceil t \rceil \geqslant t$,

$$x - (\lceil x/T_k^{(2)} \rceil - 1) \cdot T_k^{(2)} \leqslant x - ((x/T_k^{(2)}) - 1) \cdot T_k^{(2)} \leqslant T_k^{(2)}.$$

Recalling that $T_k^{(2)}/T_k^{(1)} \geqslant N_k$, and using $\lceil t \rceil - 1 \leqslant t$ we obtain

$$\widehat{N}_k(x) \leqslant \frac{x}{T_k^{(2)}} \cdot N_k + \lceil T_k^{(2)}/T_k^{(1)} \rceil \wedge N_k \leqslant \frac{x \cdot N_k}{T_k^{(2)}} + N_k.$$

Furthermore $x \cdot N_k/T_k^{(2)} \leqslant \widehat{N}_k(x)$. If we introduce this into the double inequality above, divide by x and let $x \to +\infty$ then be obtain

$$\hat{\rho}_k(i_k) = \lim_{x\to\infty} \frac{\widehat{S}_k^{(i_k)}(x)}{x} = \frac{N_k \cdot \check{C}_k^{M_k-1}}{M_k \cdot T_k^{(2)}}.$$

Furthermore, with $\hat{\sigma}_k(i_k) = \check{C}_k^{M_k-1}$ we have a (σ,ρ)-bound for $\widehat{S}_k^{(i_k)}(x)$. Since $\hat{\rho}_k(i_k)$ and $\hat{\sigma}_k(i_k)$ are finite and independent of i_k, $\hat{\rho}_k^*$ and $\hat{\sigma}_k^*$ are finite and thus Proposition 2.24 applies, which proves the statement of the proposition. ■

Notice that in the context of finding a tight slope for the (σ,ρ)-bound for a task, the parameter σ of the used (σ,ρ)-bound do not need to be "as small as possible", only their existence is required. If response time bounds are derived from (σ,ρ)-bound, better results might be obtained by choosing σ tighter, which can induce a loss of accuracy on ρ.

3.2 Dependent tasks

So far, we have considered tasks which are defined independently of each other, by assumptions that concern tasks separately. In this section we will consider assumptions in which instances of different tasks are involved.

By examining types of task such as *transactions*, introduced by Tindell in [32], and tasks that are defined by an *directed acyclic graph* (DAG), see [5] by Baruah or even *generalized multiframe tasks* (GMF) [6], we can identify two kinds of constraints on activation patterns that we call *offset constraints* and *sequencing constraints*. In this section we give their definition in our model and show how to derive a majorizing task process that takes these constraints into account. As illustration we apply the result to transactions, generalized multiframe tasks and DAG-tasks. By doing this, we implicitly derive some new results. For the considered types of tasks, mainly feasibility condition are known. We are interested in response time bounds. It will be possible to compute response time bounds under EDF, for dependent task; in [26], response time are only derived for sporadically periodic task. Furthermore, it will be possible to derive response time bounds under FPP for GMF and DAG tasks, which is so far limited to transactions [32]. The same applies to FIFO and LIFO.

Notice that the so called *precedence constraints* mean that an instance of one task must finish its execution before an instance of another task starts to execute. This is for example required if the first instance has to communicate informations to the second. Notice also that sequencing and offset constraints only concern activation times. They do not have any influence on the order in which pending instances are executed, because this is determined by the scheduling policy. Offset constraints can not guarantee precedence relations, but can be used to realize them, see [1] for FPP and [27] for EDF. See also [16], [29] for a policy based on fixed priorities, but with the additional constraints, that a certain task can not be activated before some other task has finished.

3.2.1 Offset-relations

3.2.1.1 Introduction The critical instant [22] is based on synchronous activations of different tasks. But if for some reason the activation times of two different tasks can never occur at the same time or more precisely only with a minimal offset, then the sum of their demands as function of time is always lower than after the critical instant. As a result, the preemption these tasks impose on lower priority tasks is overestimated by considering the critical instant. The aim of taking *offset constraints* into account is to obtain tighter majorizing WAF's and thus tighter response time bounds. There are several situations where the accuracy of response time bounds can be improved by taking offset relation into account.

- Suppose for example that several preemptively scheduled tasks need a same *critical resource*, i.e. a resource during the use of which task should not be preempted because otherwise it could leave the resource in an inconsistent state. It can be possible to impose offset constraints on the task activation patterns such that each instances of these task ends before another is activated, making the use of semaphores unnecessary. It is the response time bounds analysis that takes offset relations into account which allows to guarantee such properties.

- Offset constraints can be deliberately introduced to scatter the demands of tasks in order to reduce response times, by making the critical instant impossible; because of the offset constraints, only certain tasks are released at the same time.

- They can also appear naturally if for example two tasks need a common resource which after use is not immediately available again for tasks of the set, because of an external actor that needs the resource. To remain within the non-idling assumption, the second task must be released when the external actor has released the resource, which implies a certain offset with the activation time of the first task.

3.2.1.2 Definition In [32], Tindell has considered a task model called *transaction*, where recurrent tasks, belonging to a certain group are activated with specific offsets. A transaction can be seen as a higher level task consisting of sub-tasks that are not necessarily executed as soon as possible one after the other, but at certain minimal temporal distances. We generalize the idea as follows.

Definition 3.3 *Let with each pair of tasks* $(\tau_k, \tau_{k'})$ *be associated an offset* $\phi_{k,k'} \geqslant 0$. *The offset constraints specified by the matrix* $[\phi_{k,k'}]$ *are respected if every pair of instances* $(\tau_{k,n}, \tau_{k',n'})$, *with*

$n \neq n'$ if $k = k'$, satisfies

$$A_{k,n} > A_{k',n'} \qquad \textit{or} \qquad A_{k',n'} \geqslant A_{k,n} + \phi_{k,k'}. \tag{3.22}$$

Sometimes $\phi^*_{k,k'} = \phi_{k,k'} \cdot 1\!\!1_{[k \neq k']}$ is needed.

Several points can be noticed. If $\phi_{k,k'} = 0$ (3.22) is actually not constraint. The offsets $\phi_{k,k}$ concern consecutive instances of the same task; thus for a sporadic task $\phi_{k,k} = T_k$. Offset constraints give two kinds of information. On one hand, they give by ϕ_{k_1,k_2} a minimal distance between two instances τ_{k_1,n_1} and τ_{k_2,n_2}, independently of the context in which they are activated. On the other hand, if some other task τ_{k_3} is activated between them: $A_{k_1,n_1} \leqslant A_{k_3,n_3} \leqslant A_{k_2,n_2}$, then also a minimal distance depending on the context can be obtained $A_{k_2,n_2} \geqslant A_{k_1,n_1} + \phi_{k_1,k_3} + \phi_{k_3,k_2}$. If on a trajectory τ_{k_3} is activated between τ_{k_1,n_1} and τ_{k_2,n_2}, the minimal distance is

$$\max(\phi_{k_1,k_3} + \phi_{k_3,k_2}, \phi_{k_1,k_2}),$$

but only ϕ_{k_1,k_2} if τ_{k_3} is not activated in between. This is of some importance for finding exact MWAF. Notice that offset constraints must be chosen carefully to avoid contradiction with other assumption such as those which define the tasks. For example, two strictly periodic task τ_k, τ_i with equal periods $T_k = T_i = P$ can not satisfy offset constraints given by $\phi_{k,i} = \phi_{i,k} = P$.

Notice also, that with an additional constraint on the first activation times $A_{k,0}$ it is possible to represent systems of strictly periodic tasks with some fixed initial offsets, such as they have been studied in [2] and [15]. We will however not consider such systems.

3.2.1.3 Majorizing task process The process $(\hat{A}, \widehat{C}, \widehat{D})$ that we construct below, is just $\underline{\text{a}}$ majorizing task process, with WAF's that are not necessarily exact bounds for all kinds of offset relations. But, for transactions and GMF-task they are exact MWAF, as we will see.

Recall that by definition, for each trajectory $\omega \in \Omega$ and every interval $[u, u + x[$, there must exist a trajectory $q \in Q$ such that

$$S_k(u, u + x, u + d; \omega) \leqslant \widehat{S}_k(0, x, d; q).$$

Consider a trajectory ω after some time u and let us construct the corresponding majorizing trajectory q. In general, the earlier an instance is activated, the larger the resulting WAF but offset constraints can imply that a task can not be activated anywhere because of the presence of some other task. However, if $\phi_{k,k'} = \phi_{k',k} = 0$, then the activations between τ_k and $\tau_{k'}$ have no influence on each other. Let us thus partition the set of task $\mathcal{T}$ into subsets $\mathcal{G}_h$ $(h = 1, \dots, g)$ with size m_h of related tasks:

$$\begin{aligned} &h \neq h' \quad \Rightarrow \quad \mathcal{G}_h \cap \mathcal{G}_{h'} = \emptyset \\ &\tau_k \in \mathcal{G}_h,\ \tau_{k'} \in \mathcal{G}_{h'},\ h \neq h' \quad \Rightarrow \quad \phi_{k,k'} = \phi_{k',k} = 0. \end{aligned}$$

Let τ_{q_h} be a task of $\mathcal{G}_h$ which is activated earlier than any other task of $\mathcal{G}_h$ after u. For τ_{q_h}, the earliest possible activation time is u. Since τ_{q_h} is the first task, any other task in $\tau_k \in \mathcal{G}_h$ can not be activated earlier than $u + \phi_{q_h,k}$, i.e. there is no activation of τ_k in the interval $[u, u + \phi_{q_h,k}[$, implying

$$S_k(u, u + x, u + d) = S_k(u + \phi_{q_h,k}, u + x, u + d) \leqslant \widehat{S}_k^{(i_k)}(x - \phi_{q_h,k}, d - \phi_{q_h,k}),$$

assuming there is for each task a family of MWAF's $\mathcal{S}_k = \{\hat{S}_k^{(i_k)} \mid i_k = 0, \dots, M_k - 1\}$. If $q_h = k$, then the first arrival is not at $\phi_{q_h,k} = \phi_{q_h,q_h}$ but at $\phi^*_{q_h,q_h} = 0$. We can thus choose as majorizing trajectory $q = (q_1, \dots, q_h, \dots, q_g, i_1, \dots, i_m)$, with activation defined by

$$\widehat{S}_k(0, x, d; q) = \widehat{S}_k^{(i_k)}(x - \phi^*_{q_h,k}, d - \phi^*_{q_h,k}) \quad \forall k. \tag{3.23}$$

3.2.2 Sequencing constraints

Offset constraints are not sufficient to capture certain behaviors of real-world tasks. They can not express in general, that between each activation of a task τ_k and the following activation of a task τ_i there must be exactly one activation of some other task τ_l. It can be noticed that such a behavior occurs with *recurring branching tasks* as introduced by Baruah in [4] to model task that can execute differently, depending on the state of the system during the period where they execute. The task is subdivided into subtasks which represent the (alternative) parts of the task. The possible combinations of the parts and their order of execution can be expressed by sequencing constraints, as we will see. We formalize such constraints by a matrix $[\alpha_{k,k'}]$ with values in the set $\{0,1\}$. If $\alpha_{k,k'}=1$, then an activation of τ_k can be followed by an activation of $\tau_{k'}$, but if $\alpha_{k,k'}=0$, then some other task must be activated in between.
More precisely: let the set of tasks $\mathcal{T}$ be partitioned into subsets $\mathcal{G}_h$ of sizes m_h, such that

$$\mathcal{T}=\bigcup_{h=1,\dots,g}\mathcal{G}_h \qquad h\neq h':\quad \mathcal{G}_h\cap\mathcal{G}_h=\emptyset.$$

For each subset $\mathcal{G}_h$, let with each pair of tasks $\tau_k,\tau_i\in\mathcal{G}_h$ be associated a number $\alpha_{k,i}\in\{0,1\}$ and define $\mathcal{A}_k\overset{\mathsf{def}}{=}\{\tau_i\in\mathcal{G}_h \mid \alpha_{k,i}=1\}$. The set $\mathcal{A}_k$, can be seen as the set of those tasks that can be activated "immediately" after τ_k.

Definition 3.4 *The* sequencing constraints *specified by the matrix* $[\alpha_{k,i}]$ *for a group of tasks* $\mathcal{G}_h$ *are respected if each pair of tasks* $\tau_k,\tau_i\in\mathcal{G}_h$ *satisfies* $\forall\, n,j\in\mathbb{N}$

$$\begin{aligned}&A_{i,j}<A_{k,n} && \text{or}\\ &A_{k,n}\leqslant A_{i,j},\quad \alpha_{k,i}=0 && \Rightarrow\qquad \exists\,\tau_l\in\mathcal{A}_k,\ p\in\mathbb{N}\ \vdash\ A_{k,n}\leqslant A_{l,p}\leqslant A_{i,j}.\end{aligned}\tag{3.24}$$

Notice that a task process can only satisfy sequencing constraints if

$$\mathcal{A}_k\neq\emptyset \qquad \forall k.\tag{3.25}$$

Notice also that $\alpha_{k,i}=0$ does not necessarily imply $A_{k,n}\neq A_{i,j}$ for all $n,j\in\mathbb{N}$, i.e. there is not necessarily an offset $\phi_{k,i}>0$. In the case $A_{k,n}=A_{i,j}$, the tasks that appear between τ_k and τ_i in the sequencing constraint are also activated at $A_{k,n}=A_{i,j}$.

3.2.2.1 A majorizing task process Consider a set of tasks with offset and sequencing constraints. On a trajectory ω after some time u, the sequence of activation times of tasks belonging to a group $\mathcal{G}_h$ can be described by a sequence of indexes (k_i,n_i) in the order of their appearance

$$A_{k_i,n_i}\leqslant A_{k_{i+1},n_{i+1}}$$

without omission. i.e. such that for all i, $\tau_{k_i}\in\mathcal{G}_h$ and $\tau_k\in\mathcal{G}_h\Rightarrow\forall\, n,\ \exists\, i\ \vdash\ k_i=k,\ n_i=n$.

Notice that A_{k_i,n_i-1} is the previous activation of τ_{k_i} before A_{k_i,n_i}, if it exists. The offset constraints imply

$$A_{k_i,n_i}\geqslant\begin{cases}\max(A_{k_{i-1},n_{i-1}}+\phi_{k_{i-1},k_i},A_{k_i,n_i-1}+\phi_{k_i,k_i}) & \text{if } A_{k_i,n_i-1}\geqslant u\\ A_{k_{i-1},n_{i-1}}+\phi_{k_{i-1},k_i} & \text{if } A_{k_i,n_i-1}<u.\end{cases}$$

The sequences (k_i,n_i) are only those which satisfy the sequencing constraints. There are two cases depending on whether the activation time A_{k_i,n_i-1} of the previous instance of τ_{k_i} is is situated after u or before. In the latter case, the corresponding constraint induced by ϕ_{k_i,k_i} is ignored because A_{k_i,n_i-1} lies outside of the interval of observation $[u,t[$.

Notice that it is a deliberate choice to consider only the constraints due to the offsets ϕ_{k_{i-1},k_i} and ϕ_{k_i,k_i}; it leads to an exact majorizing task process for the kinds of task considered hereafter.

Let us now define the majorizing trajectory q. The construction will associate with each sequence (k_i, n_i) a trajectory q based on the smallest allowed offsets between activation times. Without further knowledge about the sequencing constraints we can not do more. The analysis will be refined for the different kinds of task in Propositions 3.8, 3.9 and 3.10.

We need for each task the index of the first instance activated after or at u: $n_{k_i}(u) = \min\{j \,|\, A_{k_i,j} \geqslant u\}$. Let $\hat{n}_i \overset{\mathsf{def}}{=} n_i - n_{k_i}(u)$. If the trajectory space Q of the majorizing task process contains a trajectory q with activation times for the tasks of $\mathcal{G}_h$ such that

$$\begin{aligned}
\widehat{C}_{k_i,\hat{n}_i} &= C_{k_i,n_{k_i}(u)+\hat{n}_i} \\
\hat{A}_{k_i,\hat{n}_i} &= \begin{cases} 0 & \text{if } i = 0 \\ \hat{A}_{k_{i-1},\hat{n}_{i-1}} + \phi_{k_{i-1},k_i} & \text{if } i \geqslant 1,\ n_i = 0 \\ \max(\hat{A}_{k_{i-1},\hat{n}_{i-1}} + \phi_{k_{i-1},k_i}, \hat{A}_{k_i,n_i-1} + \phi_{k_i,k_i}) & \text{if } i \geqslant 1,\ n_i \geqslant 1 \end{cases}
\end{aligned}$$

then it is indeed majorizing because $\hat{A}_{k_i,\hat{n}_i} \leqslant A_{k_i,n_{k_i}(u)+\hat{n}_i} - u$. The same must be true for all other groups and thus a majorizing trajectory is a function of the different sequences $\{(k_{i_h}, n_{i_h}) \,|\, i_h \in \mathbb{N}\}$ for the different groups $\mathcal{G}_h$.

3.2.3 Transactions

Particular kinds of offset constraints appear if groups of tasks are executed as recurrent *transactions* [32]. A transaction is based on a subset $\mathcal{G}$ of tasks. Each instance of the transaction consists basically in one activation of each task of the group, in a certain order and with certain offsets. Tasks are not necessarily activated in every transaction, but at least once in a certain number of transactions. In [32], response time bounds are derived that take offset constraints into account under the fixed preemptive priority scheduling policy. In this section we derive the corresponding family of majorizing WAF's, which will be used in later sections to determine response time bounds that take offset constraints into account, for FPP as in [32], but also for EDF.

A transaction consists of a subset of tasks $\mathcal{G} = \{\tau_{k_1}, \tau_{k_2}, \dots \tau_{k_m}\} \subset \mathcal{T}$. The activation times of each task $\tau_k \in \mathcal{G}$ are determined by the activation times $A_{\mathcal{G},n}$ of the transaction and an offset $\Phi_{\mathcal{G},k}$:

$$\tau_k \in \mathcal{G} : \qquad \forall\, j,\ \exists\, n \vdash A_{k,j} = A_{\mathcal{G},n} + \Phi_{\mathcal{G},k}. \tag{3.26}$$

It implies offset constraints between tasks. It is assumed that the offsets are smaller than the smallest cycle time of the transaction:

$$\Phi_{\mathcal{G},i} \leqslant T_{\mathcal{G}} \quad \forall\, i. \tag{3.27}$$

Definition 3.5 *A* transaction *is said to be* sporadic *if there exists a minimal inter-activation time $T_{\mathcal{G}}$:*

$$A_{\mathcal{G},n+1} \geqslant A_{\mathcal{G},n} + T_{\mathcal{G}}. \tag{3.28}$$

Conditions (3.28) and (3.26) imply for tasks of the transactions that they are sporadic with $T_i = T_{\mathcal{G}}$, i.e. a task can not be activated more often than the transaction it is belonging to. Being part of a transaction implies certain constraints on the inter-activation times of the task, but some freedom remains. We can define a kind of task by the additional constraint: for two integers $N_i \leqslant e_i$

$$\left.\begin{aligned} A_{i,j} &= A_{\mathcal{G},n} + \Phi_{\mathcal{G},i} \\ A_{i,j+N_i} &= A_{\mathcal{G},n'} + \Phi_{\mathcal{G},i} \end{aligned}\right\} \Rightarrow\ |n' - n| \geqslant e_i.$$

Such a task is activated at most N_i times in e_i transactions. Thus, it satisfies the assumptions of a sporadically periodic task with parameters

$$T_i^{(1)} = T_\mathcal{G} \qquad\qquad T_i^{(2)} = e_i \cdot T_\mathcal{G} \qquad\qquad N_i \leqslant e_i. \tag{3.29}$$

The transactions defined in [32] correspond to $N_i = 1$; task are then sporadic, with $T_i = e_i \cdot T_\mathcal{G}$. So far, constraints only concern activation times but not execution times. In [32] a maximal execution time is supposed. We will also consider multiframe execution times.

As example consider a periodic transaction $\mathcal{G} = \{\tau_1, \tau_2, \tau_3\}$ with $T_\mathcal{G} = 50$.

$e_1 = 1$	$e_2 = 2$	$e_3 = 1$
$\Phi_{\mathcal{G},1} = 20$	$\Phi_{\mathcal{G},2} = 0$	$\Phi_{\mathcal{G},3} = 40$

$$[\phi_{k,k'}] = \begin{pmatrix} 50 & 30 & \mathit{20} \\ \mathit{20} & 100 & 40 \\ 30 & \mathit{10} & 50 \end{pmatrix}.$$

The matrix $[\phi_{k,k'}]$ is derived using Proposition 3.7 below.

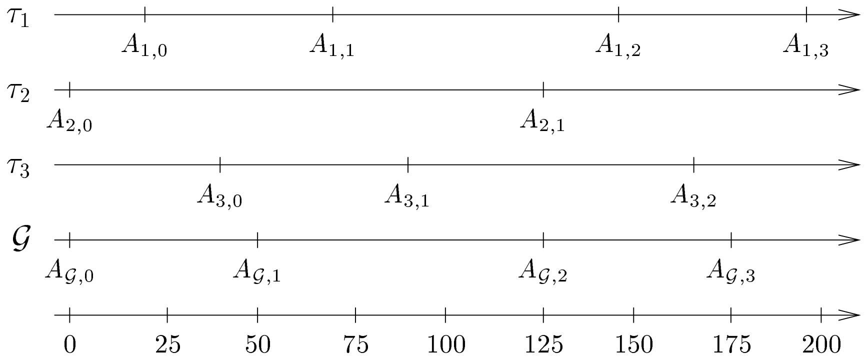


Figure 4: Offset relation due to transactions

In Figure 4 the beginning of some trajectory is shown. Notice that the activation $A_{\mathcal{G},2} = A_{\mathcal{G},1} + 75$, it is not situated at the smallest possible distance from $A_{\mathcal{G},1}$. It can also be seen that the following sequencing constraints are satisfied:

$$[\alpha_{k,i}] = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}.$$

But notice that they are actually a direct consequence of the offset constraints and thus $[\alpha_{k,i}]$ is neither needed to define transactions, nor to derive response time bounds.

Notice also that a transaction could be represented by a directed acyclic graph (see Section 3.2.5), if for each task τ_k, $\mathsf{lcm}\{e_i \mid \tau_i \in \mathcal{G}\}/e_k$ consecutive instances are modeled by different tasks (with the same execution time and relative deadline as τ_k), then an equivalent DAG task can be constructed. It has a period of $T_\mathcal{G} \cdot \mathsf{lcm}\{e_i \mid \tau_i \in \mathcal{G}\}$. However, an analysis based on the transaction model is more efficient.

One can also imagine other kinds of transactions:

Definition 3.6 *A transaction is said to be* sporadically periodic *if there exist an integer* $N_\mathcal{G} \geqslant 1$ *and* $T_\mathcal{G}^{(1)}, T_\mathcal{G}^{(2)} \in \mathbb{R}_+$ *satisfying* $N_\mathcal{G} \cdot T_\mathcal{G}^{(1)} \leqslant T_\mathcal{G}^{(2)}$ *and such that*

$$A_{\mathcal{G},n+1} \geqslant A_{\mathcal{G},n} + T_\mathcal{G}^{(1)} \qquad A_{\mathcal{G},n+N_\mathcal{G}} \geqslant A_{\mathcal{G},n} + T_\mathcal{G}^{(2)}.$$

For tasks in such a transaction, (3.26) implies

$$T_i^{(1)} = T_{\mathcal{G}}^{(1)} \qquad T_i^{(2)} = T_{\mathcal{G}}^{(2)} \qquad N_i = N_{\mathcal{G}}$$

i.e. they are sporadically periodic.
Independently of the type of transaction or task, the offset parameters are given as follows.

Proposition 3.7 *Let tasks of a set $\mathcal{T}$ be part of (sporadically) periodic transactions $\mathcal{G}_1, \dots, \mathcal{G}_g$. Tasks are supposed to be part of at most one transaction: $\mathcal{G}_h \cap \mathcal{G}_{h'} = \emptyset$. The offset parameters of tasks in sporadically periodic transactions are given by*

$$\Phi_{i,i'} = \begin{cases} 0 & \nexists h \vdash \{\tau_i, \tau_{i'}\} \subset \mathcal{G}_h & \\ e_i \cdot T_{\mathcal{G}_h} & \exists h \vdash \tau_i \subset \mathcal{G}_h & \text{and } i = i' \\ \Phi_{\mathcal{G}_h,i'} - \Phi_{\mathcal{G}_h,i} & \exists h \vdash \{\tau_i, \tau_{i'}\} \subset \mathcal{G}_h & \text{and } \Phi_{\mathcal{G}_h,i} \leqslant \Phi_{\mathcal{G}_h,i'} \\ T_{\mathcal{G}_h} + \Phi_{\mathcal{G}_h,i'} - \Phi_{\mathcal{G}_h,i} & \text{''} & \text{and } \Phi_{\mathcal{G}_h,i} > \Phi_{\mathcal{G}_h,i'}. \end{cases}$$

Proof: Let τ_i and $\tau_{i'}$ be two different tasks of a transaction $\mathcal{G}$. Let $A_{i,j}$ and $A_{i',j'}$ be some activation times. Suppose $A_{i,j} \leqslant A_{i',j'}$. By (3.26) there exist n, n' such that $A_{i,j} = A_{\mathcal{G},n} + \Phi_{\mathcal{G},i}$ and $A_{i',j'} = A_{\mathcal{G},n'} + \Phi_{\mathcal{G},i'}$. Two cases arise

- $n < n'$: $A_{i',j'} - A_{i,j} = A_{\mathcal{G},n'} - A_{\mathcal{G},n} + \Phi_{\mathcal{G},i'} - \Phi_{\mathcal{G},i} \geqslant T_{\mathcal{G}} + \Phi_{\mathcal{G},i'} - \Phi_{\mathcal{G},i}$.
- $n = n'$: $A_{i',j'} - A_{i,j} \geqslant \Phi_{\mathcal{G},i'} - \Phi_{\mathcal{G},i}$.

The case $n > n'$ can not occur. Otherwise, we would have $A_{\mathcal{G},n} \geqslant A_{\mathcal{G},n'} + T_{\mathcal{G}}^{(1)}$. On the other hand $A_{\mathcal{G},n} + \Phi_{\mathcal{G},i} \leqslant A_{\mathcal{G},n'} + \Phi_{\mathcal{G},i'}$, implying then $\Phi_{\mathcal{G},i'} - \Phi_{\mathcal{G},i} \geqslant T_{\mathcal{G}}^{(1)}$, which is not possible because of (3.27). ■

If we had not assumed (3.27), then instances of one transaction could actually be activated near instances of a following transaction and because of (3.28) no offset constraint could be guaranteed.

3.2.3.1 Majorizing task process In this section we will derive a majorizing task process $(\widehat{A}, \widehat{C}, \widehat{D})$, that gives exact bounds on WAF, i.e. consists of critical instants.

Recall the majorizing task sequences given by (3.23). For (multiframe) tasks in (sporadically periodic) transactions, the families of MWAF's $\mathcal{S}_k$ are provided by Proposition 3.1. To know if the MWAF of this majorizing task process are exact bounds, we have to check if the activation patterns represented by each $q \in Q$ can actually be realized, i.e. if there exists a trajectory $\omega \in \Omega$ which behaves after some time u exactly as q after $t = 0$.

The offsets $\phi^*_{q_h,k}$ that appear in (3.23) induce

$$\widehat{A}_{k,n} = \phi_{q_h,k} + \lfloor n/N_k \rfloor \cdot T_k^{(2)} + (n \bmod N_k) \cdot T_k^{(1)},$$

for the activations on a majorizing trajectory q; compare with (3.19). The execution times and relative deadlines satisfy (3.20) and (3.21).

Consider the trajectory ω with activation times

$$A_{\mathcal{G}_h,n} = n \cdot T_{\mathcal{G}_h}$$

for the transactions and

$$A_{k,n} = \phi_{\mathcal{G}_h,k} + \lfloor n/N_k \rfloor \cdot T_k^{(2)} + (n \bmod N_k) \cdot T_k^{(1)}$$

for $\tau_k \in \mathcal{G}_h$. Because of (3.29), they satisfy (3.26). Let the execution times and relative deadlines be given by (3.20) and (3.21) and thus this trajectory is part of (A, C, D) because all required assumptions are satisfied.

Now, since $\phi_{\mathcal{G}_h,k} = \phi_{\mathcal{G}_h,q_h} + \phi_{q_h,k} + T_{\mathcal{G}_h} \cdot \mathbb{1}_{[\phi_{\mathcal{G},k} < \phi_{\mathcal{G},q_h}]}$, see Proposition 3.7, we have

$$S_k(u, u+x, u+d; \omega) = \widehat{S}_k(t, d, q)$$

for $u = \phi_{q_h,k} + T_{\mathcal{G}_h} \cdot \mathbb{1}_{[\phi_{\mathcal{G},k} < \phi_{\mathcal{G},q_h}]}$.

Thus, q is a trajectory that can be realized by (A, C, D), i.e. using q gives exact bound. We have now proven the following:

Proposition 3.8 *The task process $(\widehat{A}, \widehat{C}, \widehat{D})$, with trajectories defined for $\tau_k \in \mathcal{G}_\langle$ by*

$$\widehat{S}_k(0, x, d; q) \overset{\mathsf{def}}{=} \widehat{S}_k^{(i_k)}(x - \phi^*_{q_h,k}, d - \phi^*_{q_h,k}) \quad \forall k,$$

where $q = (q_1, \ldots, q_h, \ldots, q_g, i_1, \ldots, i_m)$ with $q_h \in \{1, \ldots, m_h\}$, $i_k \in \{1, \ldots, M_k - 1\}$ and $S_k^{(i_k)}$ are given by Proposition 3.1, is an exact majorizing task process for a set of sporadically periodic transactions consisting of multiframe tasks.

It remains to determine the ρ_k. We have

$$\hat{\rho}_k(q) = \lim_{x \to \infty} \widehat{S}_k^{(i_k)}(x - \phi_{q_h,k}) = \lim_{t \to \infty} \widehat{S}_k^{(i_k)}(t) = \frac{N_k \cdot \check{C}_k^{M_k - 1}}{M_k \cdot T_k^{(2)}} = \hat{\rho}^*_k,$$

using Proposition 3.2 and the fact that the limit is independent of i_k and q_h. With $\hat{\sigma}_k(q) = \hat{\sigma}^*_k = \check{C}_k^{M_k-1}$, a (σ,ρ)-bound is obtained for all $q \in Q$ and thus by Proposition 2.24 implies that $\hat{\rho}^*_k$ is the tightest bound.

The size of trajectory space Q, that is the number of different majorizing trajectories, contributes to the complexity of the numerical computation of response time bounds. It is thus important to know this size. As can be seen from the definition, a trajectory q depends on the tasks τ_{q_h} activated first in each transaction $\mathcal{G}_h$ and on an appropriate MWAF $\widehat{S}_k^{(i_k)}$ for each task. Therefore, $q = (q_1, \ldots, q_h, \ldots, q_g, i_1, \ldots, i_m)$, with $q_h \in \{1, \ldots, m_h\}$ and $i_k \in \{1, \ldots, M_k - 1\}$. The different components being independent, we have

$$|Q| = m_1 \cdot \ldots \cdot m_g \cdot M_1 \cdot \ldots \cdot M_m, \tag{3.30}$$

which is a finite number. Notice that Ω has usually not a finite size. The size of Q is finite, but exponential in the number of tasks. With $M = \max_k M_k$ and given that $\sum_{h=1}^{g} m_h = m$, Proposition A.10 implies

$$|Q| \leqslant M^m \cdot \left(\frac{m}{g}\right)^g \leqslant M^m \cdot e^{\frac{m}{e}}.$$

It shows that the number of trajectories is the largest for a given number of task, when $m/g = e$, i.e. all transactions contain 2 or 3 tasks, since $e \approx 2.72$.

If exact bounds are not required, then it is possible the reduce the size of Q. If instead of families of MWAF $\mathcal{S}_k$, unique MWAF's $\widehat{S}^*_k$ are used then $|Q| = m_1 \cdot \ldots \cdot m_g$. The complexity due to different offset patterns can also be reduced, see Section 4.6.3.

3.2.4 Generalized multiframe tasks (GMF)

The definition of multiframe tasks impose the same minimal time P_k between all successive activations of its frames. Furthermore relative deadlines are supposed to be equal to P_k. The aim of defining *generalized multiframe task* (GMF) [6] is to allow different minimal offsets between frames and to weaken the assumption on relative deadlines.

We define a GMF-task as a subset $\mathcal{G}_h = \{\tau_0, \tau_1, \dots, \tau_{m_h}\} \subset \mathcal{T}$ of tasks. To simplify notations, we assume that $\mathcal{G}_h$ consists of the m_h first tasks of the set. For each task $\tau_k \in \mathcal{G}_h$ a worst-case execution time C_k and a constant relative deadline $\overline{D}_k$ are known :

$$\overline{D}_{k,n} = \overline{D}_k \tag{3.4}$$

$$C_{k,n} \leqslant C_k. \tag{3.3}$$

The tasks have to be activated in a certain cyclic order that we will express by sequencing constraints:

$$\forall\ \tau_k, \tau_{k'} \in \mathcal{G}_h \quad \exists\ i_0, \dots, i_l \quad \vdash \left| \begin{array}{l} i_0 = k,\ i_l = k' \\ \alpha_{i_j, i_{j+1}} = 1 \quad j = 0, \dots, l-1. \end{array} \right. \tag{3.31}$$

It means that between any two tasks of $\mathcal{G}_h$ there is a path which is allowed by the sequencing constraints. Furthermore, each task can only be followed by one other task:

$$|\mathcal{A}_k| = 1. \tag{3.32}$$

Offsets are supposed to satisfy

$$\tau_k, \tau_i, \tau_j \in \mathcal{G}_h \quad \alpha_{k,i} = \alpha_{i,j} = 1 \qquad \Rightarrow \qquad \phi_{k,j} = \phi_{k,i} + \phi_{i,j}. \tag{3.33}$$

The defining assumptions imply the following properties: because of (3.25), the assumption (3.32) implies a cyclic activation order containing each subtask once. Furthermore, only the m_h different offsets $\phi_{k,i}$ for which $\alpha_{k,i} = 1$ need to be given, all other being implied by (3.33). It can thus be seen that our definition is equivalent to the one given in [6].

The defining assumptions also induce for each task a certain minimal cycle time

$$T_k = \phi_{k,k} = \sum_{\tau_i, \tau_j \in \mathcal{G}_h} \phi_{i,j} \cdot \alpha_{i,j}, \tag{3.34}$$

corresponding to the sum of the offsets of the sequence of activations of other tasks two successive instances of τ_k. It is the same for all tasks. We called it minimal cycle time of the GMF-task: $T_{\mathcal{G}_h} \overset{\mathsf{def}}{=} \phi_{k,k}$ It means that subtasks (=frames) of GMF are sporadic tasks with the same period.

3.2.4.1 Majorizing task process

Proposition 3.9 *The task process $(\widehat{A}, \widehat{C}, \widehat{D})$, with trajectories defined for $\tau_k \in \mathcal{G}_h$ by*

$$\widehat{S}_k(0, x, d; q) = \min\left(\left\lceil \frac{x - \phi^*_{q_h,k}}{T_k} \right\rceil, \left\lfloor \frac{d - \phi^*_{q_h,k} - \overline{D}_k}{T_k} \right\rfloor + 1 \right) \cdot C_k \quad \forall k, \tag{3.35}$$

where $q = (q_1, \dots, q_h, \dots, q_g)$, is an exact majorizing task process for a set of GMF-tasks.

Proof: Consider a trajectory ω after some time u. Let τ_{q_h} denote the task that is activated earlier than any other task of $\mathcal{G}_h$. Because $|\mathcal{A}_k| = 1$, there is exactly one possible activation sequence (k_{i_h}, n_{i_h}) for the tasks of $\mathcal{G}_h$. Since $T_k = \phi_{k,k}$ we have for $\tau_k \in \mathcal{G}_h$

$$S_k(u, u+x, u+d; \omega) \leqslant \widehat{S}^*_k(x - \phi^*_{q_h,k}, d - \phi^*_{q_h,k}),$$

with S^*_k given by (3.6) and $q = (q_1, \dots, q_g)$. Because of (3.33), the instances of each task can actually be activated periodically and thus, (3.35) defines a majorizing task process with exact MWAF. ■

It can easily be proven that for the MWAF's given by (3.35), the smallest slope is $\hat{q}^*_k = C_k/T_k$.

Remark: The demand bound functions dbf used in [6] to express a feasibility condition for GMF-tasks under EDF, are related to (3.35) by

$$\mathsf{dbf}(\tau_k, x) = \sup_{q \in Q} \widehat{S}_k(0, x, x; q).$$

The family of MWAF's extends the concept of demand bound function so that also (exact) response time bounds can be computed.

3.2.5 Directed-acyclic-graph tasks (DAG)

The aim of introducing tasks that are defined by a directed acyclic graph [5] (DAG), or also *recurring branching* tasks, is to remove assumption (3.32), i.e. is to be able to represent alternative activation sequences of subtasks.

In Figure 5 we show the DAG-task example given in [5] with the corresponding notations of our model. Each vertex represents a subtask and each edge from a task τ_k to a task $\tau_{k'}$ corresponds to a parameter $\alpha_{k,k'} = 1$. Paths along the edges of the graph determine the different possible activation sequences. Each instance of a DAG task starts with a unique subtask, which we denote τ_0 and ends with a unique subtask, that we denote τ_{m_h}. In the corresponding graph, τ_0 appears as source vertex (no incoming edges) and τ_{m_h} as sink vertex (no outgoing edges). It is assumed that from τ_0 any other task can be reached. With each edge is associated a value, which we denote $\phi_{k,k'}$ if the edge goes from τ_k to $\tau_{k'}$. It represents the minimal time between to activations of τ_k and $\tau_{k'}$. With each vertex are associated the subtasks worst case execution time C_k and its relative deadline $\overline{D}_k$. An additional constraint is that two successive activations of τ_0, must be separated by a minimal time that we denote $T_{\mathcal{G}_h}$.

In our model we define a DAG-task as subset $\mathcal{G}_h = \{\tau_0, \tau_1, \dots, \tau_{m_h}\} \subset \mathcal{T}$ of tasks, with specific offset constraints and sequencing constraints. For each task, the worst-case execution time C_k and the relative deadline $\overline{D}_k$ are given. Two tasks play a particular role: all instances of the DAG-task must end with τ_{m_h} and (re)start with τ_0. More precisely, τ_{m_h} can only be followed by τ_0 and after some minimal time $\phi_{m_h,0}$. Thus, the sequencing constraints must satisfy

$$\mathcal{A}_{m_h} = \{\tau_0\} \tag{3.36}$$

and no other task can be followed by τ_0, i.e.

$$\alpha_{k,0} = 0 \qquad 0 < k < m_h. \tag{3.37}$$

Notice that the sequencing constraint (3.36) does not appear as an edge in the DAG.

Furthermore, any path allowed by $[\alpha_{k,k'}]$ that returns to the same task must pass through τ_{m_h}. For $0 < k < m_h$:

$$(i_0, \dots, i_l) \quad \vdash \quad \left| \begin{array}{l} i_0 = i_l = k \\ \alpha_{i_j,i_{j+1}} = 1 \quad j = 0, \dots, l-1 \end{array} \right. \qquad \Rightarrow \qquad \exists r \in \{0, \dots, l\} \quad \vdash i_r = m_h. \tag{3.38}$$

For pairs of tasks with $\alpha_{k,i} = 0$ the corresponding offset is to be chosen neutral:

$$\alpha_{k,i} = 0 \qquad \Rightarrow \qquad \phi_{k,i} = 0, \tag{3.39}$$

except for the offset between instances of the subtask τ_0

$$\alpha_{0,0} = 0 \qquad \phi_{0,0} = T_{\mathcal{G}_h}. \tag{3.40}$$

Notice that $\phi_{k,i} = 0$ does not necessarily mean that an instance of τ_k could be activated at the same time than an instance of τ_i, because of the interaction between offset and sequencing constraints. The offsets concerned by (3.39) are chosen to be zero because they are not needed to define DAG-tasks.

It is assumed that the period of the DAG-task is given by $T_{\mathcal{G}_h}$, i.e. the minimal time between two activation of τ_0 induced by the different possible path must be shorter than $T_{\mathcal{G}_h}$. The amount of work corresponding to the subtasks activated between two successive activations of τ_0 is supposed to be exactly bounded by $C_{\mathcal{G}_k}$:

$$(i_0, \dots, i_l) \quad \vdash \quad \left| \begin{array}{l} i_0 = i_l = 0 \\ \alpha_{i_j,i_{j+1}} = 1 \quad j = 0, \dots, l-1 \\ i_r \neq 0 \quad r = 1, \dots, l-1 \end{array} \right. \qquad \Rightarrow \qquad \begin{array}{l} \sum_{j=0}^{l-1} \phi_{i_j,i_{j+1}} \leqslant T_{\mathcal{G}_h} \\ \\ \sum_{j=0}^{l-1} C_{i_j} \leqslant C_{\mathcal{G}_k}. \end{array} \tag{3.41}$$

To be an exact bound there must exist a sequence $(i_0, \dots, i_l)$ such that $\sum_{j=0}^{l-1} C_{i_j} = C_{\mathcal{G}_k}$.

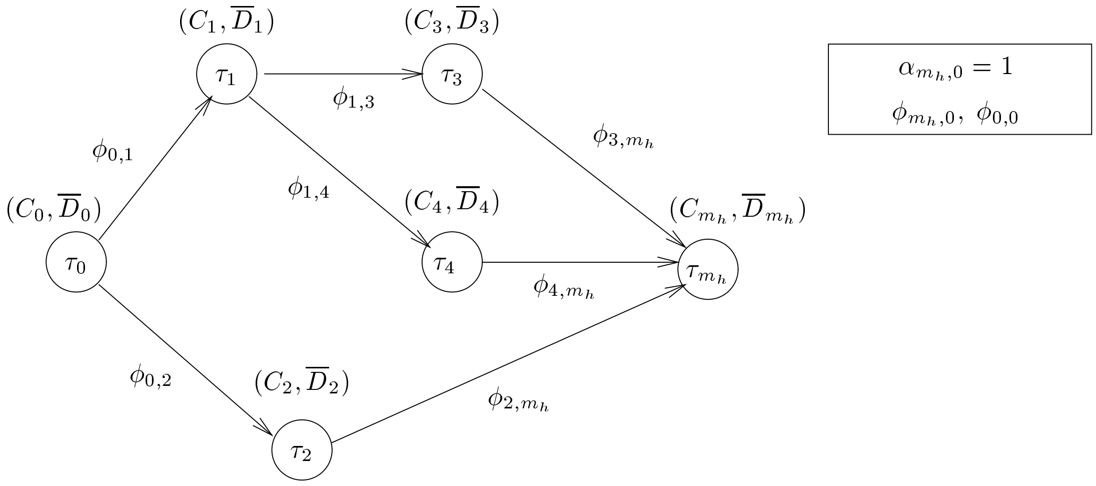


Figure 5: DAG-task.

3.2.5.1 Majorizing task process

The sequences A_{k_i,n_i} of activations that can be observed on a trajectory ω after some time u, are only those that satisfy the sequencing constraints and those only who need to be considered for the construction of $(\hat{A},\widehat{C},\widehat{D})$. But with DAG-tasks, the majorizing task sequences can not be characterized by a finite set of parameters with finite range, as it is the case for GMF-tasks. Recall that for a set of GMF-tasks majorizing task sequences are exactly determined by be tasks activated first in each group, see (3.35). Nothing like this is true for DAG-tasks and thus $(\hat{A},\widehat{C},\widehat{D})$ consists of infinitely many trajectories. However, it will appear later that for the computation of response time bounds, MWAF's only need to be defined on intervals of finite length, see Remark 4.4 on page 52. On any finite interval $[0,L[$ only finitely many trajectories are different one from another. Their number can be estimated given the number of different possible paths between τ_0 and τ_{m_h}. Let this number be b_h.

Because of assumption (3.38) and (3.41), subtask τ_0 appears periodically in the activation sequence for each q and at fixed distances $T_{\mathcal{G}_h}$. The interval $[0,L]$ contains at most $1+\lfloor L/\mathcal{G}_h\rfloor$ activations. Since furthermore there are b_h different possible paths between two successive instances of τ_0, there are thus at most

$$b_h^{(2+\lfloor L/\mathcal{G}_h\rfloor)}$$

different majorizing trajectories q on $[0,L]$. From Section 3.2.2.1 we derive:

Proposition 3.10 *For a set of DAG-tasks the exact majorizing task process $(\hat{A},\widehat{C},\widehat{D})$ consists in all possible trajectories q, that satisfy for a $\tau_k \in \mathcal{G}_h$ the following:*

$$\widehat{C}_{k,n} = C_k \quad \forall\ n,k$$

$$\widehat{D}_{k,n} = \widehat{A}_{k,n} + \overline{D}_k \quad \forall\ n,k$$

$$\hat{A}_{k_i,n_i} = \begin{cases} 0 & \textit{if } i=0 \\ \hat{A}_{k_{i-1},n_{i-1}} + \phi_{k_{i-1},k_i} & \textit{if } i \geqslant 1,\ k_i \neq 0 \textit{ or } k_i = 0,\ n_i = 0 \\ \hat{A}_{0,n_i-1} + T_{\mathcal{G}_k} & \textit{if } k_i = 0,\ n_i \geqslant 1 \end{cases}$$

$$\tau_{k_i} \in \mathcal{G}_h \ \vdash \alpha_{k_i,k_{i+1}} = 1 \ \forall\ i \in \mathbb{N}$$

$$n_i = \begin{cases} 0 & \textit{if } k_j \neq k_i \ \forall\ j < i \\ 1 + \max\{n_j \mid j < i,\ k_j = k_i\} & \textit{otherwise .} \end{cases}$$

Proof: Recall the majorizing task sequences constructed in Section 3.2.2.1. Since $\phi_{k_i,k_i} = 0$ if $k_i \neq 0$ and because of (3.41), the sequences of activation times can be rewritten as stated. ■

3.3 Mode change

3.3.1 Introduction

The functions a real-time system has to provide may change over time and consequently the task that implement them can change too, as the system moves from one mode to another. Timing analysis, adapted to mode changes, has been derived in [30] for periodic tasks scheduled under fixed preemptive priorities; in this model tasks may be removed or added to the current set and priorities can be changed. Furthermore, task characteristics may vary. In this section we are only interested in changes of task behaviors.

The analysis developed in [30] is equivalent to using a unique MWAF, i.e. a conservative bound by taking the supremum of MWAF over all possible cases that could arise in the analysis of a response time bound. We consider the case of mode changes to illustrate the concept of family of MWAF. In [30] tasks where supposed to be periodic; we remove this restriction on the type of task. Tasks can be of any type, provided a MWAF (or a family of MWAF) for each mode is known. These tasks being always more or less accurate models for real-world tasks, it improves the usefulness of the model.

An important consequence of this section is an immediate extension to all other policies considered in this document, because the timing analysis of policies we derive in following sections are only based on MWAF and not on types of task. In [30] it was assumed, in order to simplify the analysis, that $D_{k,n} \leqslant A_{k,n+1}$, i.e. every instance of a task finishes before the following instance is released. We will see that using a family of MWAF allows to remove this restriction.

3.3.2 Definition

We consider a task with two different modes. The analysis can easily be extended to more modes. In [30] it is implicitly assumed that during the execution of an instance, any preempting task can switch at most once between modes, which is like assuming that tasks remain in the same mode for a sufficiently long time, as for example the maximal length of a processor interference period. To obtain a general theoretical result, we will introduce the minimal time a mode can last.

We represent a task with two modes by a subset of task $\mathcal{G} = \{\tau_1, \tau_2\}$, mainly because in each mode the task could have a different relative deadline. We assume that for each subtask a family of MWAF's are known:

$$\mathcal{S}_1 = \{S_1^{(q_1)}(x,d) \mid q_1 \in Q_1\} \qquad \mathcal{S}_2 = \{S_2^{(q_2)}(x,d) \mid q_2 \in Q_2\}.$$

For notational convenience, we suppose the $S_i^{(q_i)}$ to be defined for $x, d \in \mathbb{R}$ with $S_i^{(q_i)}(x,d) = 0$ if $d < 0$ or $x \leqslant 0$. Let $\check{A}_{1,l}$, resp. $\check{A}_{2,l}$ denote the times where the mode change from 2 to 1, resp. 1 to 2 occurs. We assume that the task remains for at least $\check{T}_1 > 0$ in mode 1 and for at least $\check{T}_2 > 0$ in mode 2. With $e = \mathbb{1}_{[\check{A}_{1,0} > \check{A}_{2,0}]}$ this can be written as

$$\check{A}_{1,l} + \check{T}_1 \leqslant \check{A}_{2,l+e} \text{ and } \check{A}_{2,l} + \check{T}_2 \leqslant \check{A}_{1,l+1-e}.$$

Furthermore, we assume a minimal time between the last release time in a mode and the first release time in the following mode, see Figure 6. In [30], the authors have assumed that the first release of following mode can not occur earlier than would a release of the previous mode i.e. $T^{(1,2)} = T_1$ and $T^{(2,1)} = T_2$. Since we do not make any assumption about the type of task, we consider these times as additional parameters to be given.

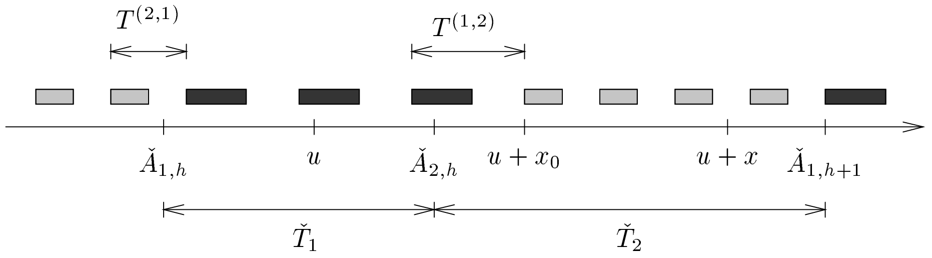


Figure 6: Minimal times related to changing modes.

3.3.3 A family of MWAF's.

On a trajectory ω after some time u we can observe a certain sequence of mode changes, but there is no way to foresee which trajectory will be worse than others. We will derive a family of MWAF's.

We limit the interval of observation to $[u, u+x[$ with $x \leqslant t_0 \overset{\mathsf{def}}{=} \min(\check{T}_1, \check{T}_2)$, so that at most one mode change can occur in the considered interval. We discuss the extension to longer intervals later.

To start with, assume the unique mode change is from mode 1 to mode 2 at $\check{A}_{2,l}$. Let $u+x_0 = A_{2,n}$ be the first release in mode 2. Notice that our analysis will be based on this release date, rather than the date of mode change itself as in [30]. To write the bound, we divide $[u, u+x[$ into two parts,

$$S_1(u,u+x,u+d) = \begin{cases} S_1(u,u+x,u+d) & \text{if } x \leqslant x_0 \\ S_1(u,u+x_0,u+d) & \text{if } x > x_0 \end{cases} \leqslant \begin{cases} \widehat{S}_1^{(q_1)}(x,d) & \text{if } x \leqslant x_0 \\ \widehat{S}_1^{(q_1)}(x_0,d) & \text{if } x > x_0 \end{cases}$$

$$= \min(\widehat{S}_1^{(q_1)}(x,d), \widehat{C}_{1,0..n_1}^{(q_1)}) \overset{\mathsf{def}}{=} \widehat{S}_1^{(q_1,n_1)}(x,d),$$

where $n_1 = \max\{n \mid \hat{A}_{1,n}^{(q_1)} < x_0\}$. For τ_2 we have

$$S_2(u,u+x,u+d) = \begin{cases} 0 & \text{if } x \leqslant x_0 \\ S_2(u+x_0,u+x,u+d) & \text{if } x > x_0 \end{cases}$$

$$= \widehat{S}_2^{(q_2)}(x - \hat{A}_{1,n_1}^{(q_1)} - T^{(1,2)}, d - \hat{A}_{1,n_1}^{(q_1)} - T^{(1,2)}) \overset{\mathsf{def}}{=} \widehat{S}_2^{(q_1,n_1,q_2)}(x,d).$$

The MWAF's for the case where the mode change is from mode 2 to mode 1 at $\check{A}_{1,l}$, are obtained in a similar way and will be denoted $\widehat{S}_2^{(q_2,n_2)}(x,d)$ and $\widehat{S}_1^{(q_2,n_2,q_1)}(x,d)$. As can be noticed, the range of n_1 and n_2 is limited because the interval of observation $[u, u+x_0[$ is limited. Thus, let

$$n^{(1)} = \min\{n \mid \hat{A}_{1,n_1}^{(q_1)} + T^{(1,2)} \geqslant x_0\} \qquad n^{(2)} = \min\{n \mid \hat{A}_{2,n_2}^{(q_2)} + T^{(2,1)} \geqslant x_0\}.$$

Thus the family of MWAF's for $\mathcal{G} = \{\tau_1, \tau_2\}$ is given by

$$\begin{aligned} &\mathcal{S} = \{\widehat{S}_k(x,d,q) \mid k \in \{1,2\},\ q = (1,q_1,n_1,q_2) \text{ or } q = (2,q_2,n_2,q_1)\} \\ &\widehat{S}_k(x,d,(z_1,z_2,z_3,z_4)) = \begin{cases} \widehat{S}_k^{(z_2,z_3)}(x,d) & \text{if } z_1 = k \\ \widehat{S}_k^{(z_2,z_3,z_4)}(x,d) & \text{if } z_1 \neq k \end{cases} \\ &q_1 \in Q_1,\ q_2 \in Q_2,\ 1 \leqslant n_1 \leqslant n^{(1)},\ 1 \leqslant n_2 \leqslant n^{(2)}. \end{aligned}$$

Let us consider the following example of a task that behaves like a sporadically periodic task in one mode and as a multiframe task in the other mode.

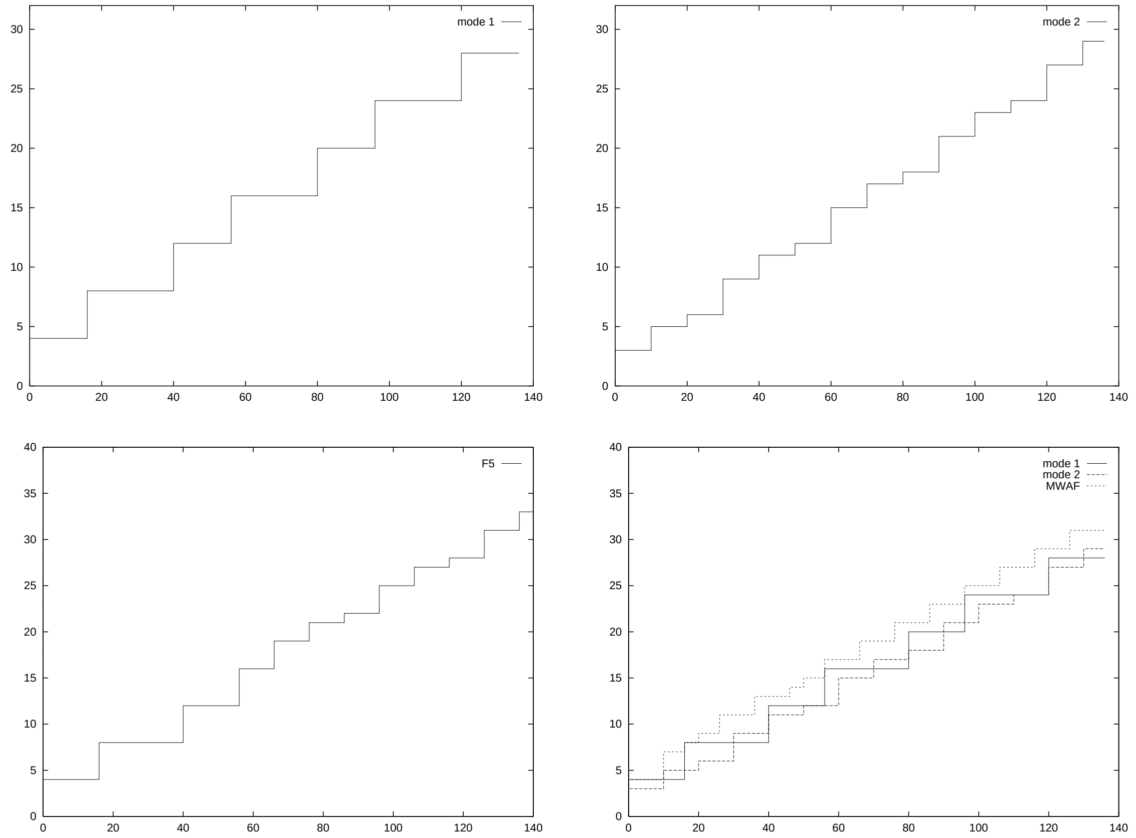


Figure 7: WAF in two modes and the corresponding MWAF

mode 1	$C_k = 4$	$\overline{D}_k = 14$	$T_k^{(1)} = 16$	$T_k^{(2)} = 40$	$N_k = 2$
mode 2	$\vec{C}_k = (3,2,1)$	$\overline{D}_k = 20$	$T_k = 10$		

The WAF's are shown in the upper part of Figure 7. We have chosen $T^{(1,2)} = T^{(2,1)} = 10$. In the lower left figure is depicted $\mathsf{F5} = \widehat{S}_k(x, d, (1, q_1, 4)) = \widehat{S}_1^{(q_1,4)}(x, d) + \widehat{S}_2^{(q_1,4,q_2)}(x, d)$, the case where the first five activations (0,1,2,3,4) are part of mode 1 and the following are part of mode 2.

We have assumed that t_0 is smaller than the longest interference period such that there is only one mode change possible. If the assumption does not hold, then one must also consider all possible scenarios of more than one mode change, but this is likely to become very complex. Another solution could be to merge the two tasks into one, and to derive a more conservative bound. With $\widehat{S}(x, d, q) = \widehat{S}_1(x, d, q) + \widehat{S}_2(x, d, q)$, $q \in Q$ we have family of MWAF for all intervals $[u, u + x[$ for $x \leqslant t_0$. With the help of Proposition 2.14 the MWAF's can be extended, for $x \in \mathbb{R}$. Notice that with this approximation it is not anymore possible to distinguish between the two modes, which implies that for testing feasibility the shorter deadline of the modes must be used. A further approximation would be to replace the family of MWAF by a unique MWAF $\widehat{S}^*$, which is shown in lower right graphic in Figure 7 as MWAF. Notice that it is not simply the maximum of the majorizing WAF's of both modes. Furthermore, it has an arbitrary shape in the sense that it is made up from activations with shorter or longer inter-activation times than the minimum of both modes. The same holds for execution times.

3.4 Tasks with arbitrary activation patterns

To illustrate further the concept of majorizing work arrival function, consider a sample of activation and execution times on an interval $[t_0, t_1[$. One could build the WAF $S_k(a, b)$ for all $t_0 \leqslant a < b < t_1$

and define $s_k(x) = \max_{t_0 \leqslant t < t_1} S_k(t, t+x)$. The function $s_k(x)$ would be a majorizing WAF at every point of the interval $[t_0, t_1[$. Since a WAF on some interval is worse if the first arrival takes place at its beginning. Thus, we actually have

$$\widehat{S}_k(x) = \max_{A_{k,n}} S_k(A_{k,n}, A_{k,n} + x).$$

One could use a more precise characterization of these tasks by deriving a family of majorizing WAF's based on offset constraints. For this purpose, the starting times t must distinguished according to the first task being activated. The first being τ_q the majorizing WAF is

$$\widehat{S}_k^{(q)}(x) = \max_{A_{q,n}} S_k(A_{q,n}, A_{q,n} + x).$$

3.5 Remarks

It can be noticed that (families) of exact MWAF's use so far are characterized by the following fact: in any time interval $[u, t[$, cumulated demands from the first to the i^{th} instance in the interval are bounded by the cumulated demands of the first i instances of the MWAF; furthermore, the time elapsed in the interval until the release of the i^{th} instance is longer than release time of the i^{th} instance of the MWAF:

$$\begin{aligned} C_{k,n_0} + \ldots + C_{k,n_0+i-1} &\leqslant \widehat{C}_{k,0} + \ldots + \widehat{C}_{k,i-1} \\ u + \hat{A}_{0,i-1} &\leqslant A_{k,n_0+i-1}, \end{aligned} \tag{3.42}$$

where τ_{k,n_0} is the first instance of τ_k activated after u. This is the reason why for example in the proof of Proposition 3.1 the number of instances and the corresponding amount of work are bounded independently of each other. These MWAF could be described as follows: *Make the first instance arrive immediately, the following with smallest possible inter arrival time and all instances with largest possible execution time.* However, Definition 2.12 does not imply the above described properties. Being more general, it allows for example a MWAF with smaller execution times than the WCET, provided a larger number of releases happen in a same period of time. As can be seen in Figure 8.a, the properties (3.42) are not satisfied. A MWAF could also consist of fewer releases than on any interval of the same length, provided the execution time are longer than the WCET, see Figure 8.b. The generality of the concept of MWAF is fully exploited by MWAF's that are conservative bounds, as for example the unique MWAF for a task with mode changes. See also Section 4.6.3.

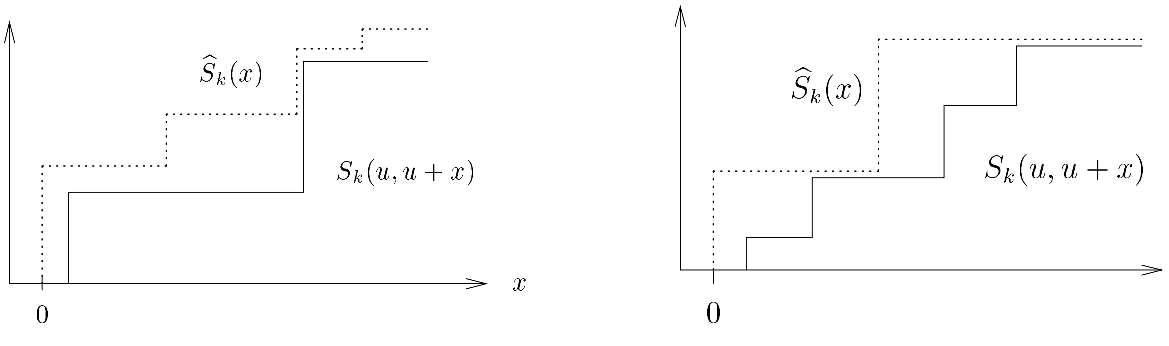


Figure 8: Examples of majorizing work arrival functions.

Notice also that whether the timing analysis for a certain kind of task is based on exact MWAF's or not is not the most important point for the practical use, since the chosen task model itself is a more of less tight (conservative) approximation of the behavior of the real-world task. Also, that a feasiblilty test is necessary and sufficient or only one of them, depends on the underlying task model. For instance, the test $\rho_{1..m} < 1$ on the load (=utilization factor) of a task set is a necessary and sufficient feasibility test under EDF if task are periodic and have deadline equal to their period. In case of shorter deadlines the test is not sufficient anymore. For the practical use of feasibility tests it is more important to be able make an efficient compromise between accuracy and complexity. We return to this problem in Section 4.6.

An important point to notice is that the timing analyses developed in subsequent sections are based on equation (2.26) only, i.e. are valid for any kind of MWAF, whether the MWAF is exactly the worst case behavior of a task or if the task can never exactly behave like the chosen MWAF or not. The latter is typically the case when the MWAF is a supremum of several WAF, of which none is everywhere a bound for all others. *For the timing analysis it means in particular that the concept of (majorizing) work arrival function allows to develop aspects which are related to behaviors of tasks separately from aspects that are related to the scheduling policies.*

4 Time independent priority functions

Under the assumption of time independence the priorities of instances are fixed. This is not to be confused with *fixed preemptive priorities* (FPP), which are a particular case, see Section 4.2. If the priority of an instance is for example given by its absolute deadline, then it has a time independent priority, because the deadline is a constant. At the task level however, the priority assignment is not time independent, since each instance has a different absolute deadline.

We will derive timing analysis for four examples of policies that are based on time independent priority functions: fixed preemptive priorities, earliest deadline first, first in first out, and last in first out. But before this, in the following section we will carry out the timing analysis as far as the time independence hypothesis allows. This will show similarities between these rather different looking policies.

4.1 Generic formulas

In this section we continue to develop the response time equation (2.49) under the additional assumption of *time independence*, i.e. $\Gamma_{k,n}(t) = \Gamma_{k,n}(0)$ for all t. In this case, the sets of higher or lower priority instances are time independent as well:

$$\mathcal{H}_{k,n} \overset{\mathsf{def}}{=} \mathcal{H}_{k,n}(0) \qquad \mathcal{L}_{k,n} \overset{\mathsf{def}}{=} \mathcal{L}_{k,n}(0). \tag{4.1}$$

Such policies are necessarily preemptive, because as soon as an instance is activated a currently executed instance with lower priority must be stopped to satisfy (2.38). As soon as no instance with higher priority is pending anymore, the execution of the stopped task continues, due to the non-idling assumption.

Consider again (2.49). For a task $\tau_{i,j} \in \mathcal{L}_{k,n}$ we have $\Pi_{i,j}(t) = 0$ during the interval $[A_{k,n}, E_{k,n}[$ and thus for all $t \leqslant E_{k,n} - A_{k,n}$:

$$W_{\mathcal{L}_{k,n}}(A_{k,n} + t) = W_{\mathcal{L}_{k,n}}(A_{k,n}) + S_{\mathcal{L}_{k,n}}(A_{k,n}, A_{k,n} + t).$$

It means that we can omit the functions related to lower priority tasks because their demands have no effect on the scheduling while τ_k is active. If we decompose $W_{1..m}$ and $S_{1..m}$, the functions indexed by $\mathcal{L}_{k,n}$ appear on both sides of the equation in (2.49) and can thus be removed.

Introducing $\mathcal{I}_{k,n} \overset{\mathsf{def}}{=} \mathcal{H}_{k,n} \cup \{\tau_{k,n}\}$ we can then rewrite the response time:

$$R_{k,n} = \min\{t > 0 \mid W_{\mathcal{I}_{k,n}}(A_{k,n}) + S_{\mathcal{I}_{k,n}}(A_{k,n}, A_{k,n} + t) = t\}. \tag{4.2}$$

Notice that $W_{\mathcal{I}_{k,n}}(A_{k,n}) = W_{\mathcal{H}_{k,n}}(A_{k,n})$, because $\tau_{k,n}$ is not yet active before $A_{k,n}$. This response time equation shows that while an instance is pending, the demands from lower priority tasks have no effect on the scheduling and consequently also no effect on the response $R_{k,n}$. Similarly we have

$$E_{k,n} = \min\{t > A_{k,n} \mid W_{\mathcal{I}_{k,n}}(A_{k,n}) + S_{\mathcal{I}_{k,n}}(A_{k,n}, t) + A_{k,n} = t\}. \tag{4.3}$$

Remind that we want to derive response time bounds. The WAF $S_{\mathcal{I}_{k,n}}$ can be bounded given the majorizing WAF's introduced in Section 2.1.2. To see this, subdivide the set of instances $\mathcal{I}_{k,n}$ according to the task to which the instances belong. Whatever the priority assignment is, the subset of instances of a task τ_i is a subset of all instances of the task and thus the WAF of the subset is bounded by S_i, which is in turn bounded by some $\widehat{S}_i^{(q_i)}$. Depending on the policy, $\mathcal{I}_{k,n}$ has a particular form for which a more specific majorizing WAF's could give a more accurate bound, but the underlying idea remains the same.

It remains $W_{\mathcal{I}_{k,n}}(A_{k,n})$, which we try to eliminate because it is the result of the work arriving before $A_{k,n}$ *and* the scheduling policy, and can thus not directly be bounded using MWAF's. As we will see, its value does not depend on the whole past before $A_{k,n}$.

Definition 4.1 *The time $U_{k,n}$ satisfying*

$$U_{k,n} \leqslant A_{k,n}, \quad W_{\mathcal{I}_{k,n}}(U_{k,n}) = 0 \qquad \text{and} \qquad U_{k,n} < t \leqslant A_{k,n} \Rightarrow W_{\mathcal{I}_{k,n}}(t) > 0 \tag{4.4}$$

is called beginning of the interference period $[U_{k,n}, E_{k,n}[$ *associated with the n^{th} instance of task τ_k.*

The term interference is motivated by the following proposition, which shows that the response time only depends on the activations after $U_{k,n}$ of instances in $\mathcal{I}_{k,n}$. These instances have an influence on the execution of $\tau_{k,n}$ via a positive workload $W_{\mathcal{I}_{k,n}}(t)$. Any instance activated before $U_{k,n}$ has necessarily finished before $U_{k,n}$ and thus can not interfere with $\tau_{k,n}$. Instances of tasks with a lower priority have no influence at all, wherever they are activated.

Proposition 4.2 *Under time independent priorities, the execution end of an instance can be written independently of workloads:*

$$E_{k,n} = \min\{t > U_{k,n} \mid S_{\mathcal{I}_{k,n}}(U_{k,n}, t) + U_{k,n} = t\ \}. \tag{4.5}$$

Proof:
Consider (4.3) under the form (A.5) on page 89 with $x_0 = A_{k,n}$ and

$$f(y) = W_{\mathcal{I}_{k,n}}(A_{k,n}) + S_{\mathcal{I}_{k,n}}(A_{k,n}, y). \tag{4.6}$$

We intend to apply Proposition A.6 with $y_0 = U_{k,n}$ and

$$g(y) = S_{\mathcal{I}_{k,n}}(U_{k,n}, y). \tag{4.7}$$

For $y \in [U_{k,n}, E_{k,n}[$ consider $W_{\mathcal{I}_{k,n}}(y)$ and write it using (2.14) as

$$W_{\mathcal{I}_{k,n}}(y) = W_{\mathcal{I}_{k,n}}(U_{k,n}) + S_{\mathcal{I}_{k,n}}(U_{k,n}, y) - \int_{U_{k,n}}^{y} \Pi_{\mathcal{I}_{k,n}}(u)du.$$

By Definition 4.1 $W_{\mathcal{I}_{k,n}}(U_{k,n}) = 0$ and $W_{\mathcal{I}_{k,n}}(u) > 0$ for all $u \in]U_{k,n}, A_{k,n}]$. The non-idling assumption implies $\Pi_{1..m}(u) > 0$. But for $\tau_{i,j} \notin \mathcal{I}_{k,n}$ we have $\Pi_{i,j}(u) = 0$ since the pending instances in $\mathcal{I}_{k,n}$ have a higher priority than $\tau_{i,j}$. Thus, $\Pi_{\mathcal{I}_{k,n}}(u) = 1$, implying

$$W_{\mathcal{I}_{k,n}}(y) = S_{\mathcal{I}_{k,n}}(U_{k,n}, y) - y + U_{k,n}.$$

Using this for with $y = A_{k,n}$ gives

$$\begin{aligned} f(y) &= S_{\mathcal{I}_{k,n}}(U_{k,n}, A_{k,n}) - A_{k,n} + U_{k,n} + S_{\mathcal{I}_{k,n}}(A_{k,n}, y) \\ &= S_{\mathcal{I}_{k,n}}(U_{k,n}, y) - (A_{k,n} - U_{k,n}). \end{aligned}$$

Thus, for $y \in [U_{k,n}, A_{k,n}[$

$$f(y) + x_0 - y_0 = S_{\mathcal{I}_{k,n}}(U_{k,n}, y) = g(y)$$

and hence (A.6) holds. Furthermore, by definition 4.1 $W_{\mathcal{I}_{k,n}}(y) > 0$, implying $g(y) > y - U_{k,n}$ and hence (A.7) holds also. ■

In (4.5) only WAF's appear and thus response times bounds can be derived based on majorizing WAF's. The structure of $\mathcal{I}_{k,n}$ depends on the scheduling policy, which induces policy specific differences for the computation of response time bound, but there is a common underlying idea, which can be expressed as follows:

Lemma 4.3 *Let a task process* (A, C, D, Π) *be scheduled according to a time independent priority assignment* Γ. *Suppose there is for each task* τ_k *a set of increasing functions* $\mathcal{P}_k = \{\widetilde{S}_k(x, a, q) \mid q \in Q\}$ *such that*

$$\forall\, \omega \in \Omega, \forall n \in \mathbb{N} \qquad \exists q \in Q \quad \vdash \quad S_{\mathcal{I}_{k,n}}(U_{k,n}, U_{k,n} + x; \omega) \leqslant \widetilde{S}_k(x, A_{k,n} - U_{k,n}, q). \tag{4.8}$$

Let $\widetilde{E}_k(a, q) \overset{\mathsf{def}}{=} \min\{x > 0 \mid \widetilde{S}_k(x, a, q) = x\}$ *and* $\mathcal{A}_k(q) \overset{\mathsf{def}}{=} \{a > 0 \mid \widetilde{S}_k(x, a, q) > x,\ \forall x \leqslant a\}$. *A bound on the response times of* τ_k *is given by*

$$R_{k,n} \leqslant \widetilde{R}_k \overset{\mathsf{def}}{=} \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a, q) - a.$$

Proof: Consider an instance $\tau_{k,n}$ on a trajectory ω and let q be given by (4.8). We have $A_{k,n} - U_{k,n} \in \mathcal{A}_k(q)$, because by the definition of $U_{k,n}$, see (4.4), and (4.8) :

$$0 < W_{\mathcal{I}_{k,n}}(x) = S_{\mathcal{I}_{k,n}}(U_{k,n}, U_{k,n} + x) - x \leqslant \widetilde{S}_k(x, A_{k,n} - U_{k,n}, q) - x,$$

for all $x \leqslant A_{k,n} - U_{k,n}$. Thus, $a = A_{k,n} - U_{k,n} \in \mathcal{A}_k(q)$. It remains to prove that that $R_{k,n}(\omega) \leqslant \widetilde{E}_k(A_{k,n} - U_{k,n}, q) - (A_{k,n} - U_{k,n})$. We use Proposition A.7 with $x_0 = U_{k,n}$, $f(x_0 + x) = S_{\mathcal{I}_{k,n}}(U_{k,n}, U_{k,n} + x)$, $\hat{x}_0 = 0$ and $\widehat{f}(\hat{x}_0 + x) = \widetilde{S}_k(x, A_{k,n} - U_{k,n}, q)$. Then (A.9) holds because of (4.8) and thus

$$x^* = E_{k,n} - U_{k,n} \leqslant \hat{x}^* = \widetilde{E}_k(A_{k,n} - U_{k,n}, q).$$

Subtracting $A_{k,n} - U_{k,n}$ on both sides gives the result. ■

The basic idea of Lemma 4.3 is to express a response time relatively to the beginning of the interference period to which it belongs:

$$R_{k,n} = E_{k,n} - A_{k,n} = (E_{k,n} - U_{k,n}) - (A_{k,n} - U_{k,n})$$

and to bound the time until the execution end from above and the time until the activation time from below. It leads to a kind of virtual response time $\widetilde{E}_k(a, q) - a$, in a virtual interference period $\mathcal{A}_k(q)$. We will see that for a majorizing task process these quantities have concrete meanings, depending on the policy.

A question which has some importance for the numerical computation of bounds is under which conditions the $\widetilde{E}_k(a, q)$ are finite. If for each function in $\mathcal{P}_k = \{\widetilde{S}_k(x, a, q) \mid q \in Q\}$ there exists a (σ,ρ)-bound in x with $\tilde{\rho}_k$ independently of a and q:

$$\widetilde{S}_k(x, a, q) \leqslant \tilde{\sigma}_k + \tilde{\rho}_k \cdot x \qquad \forall x,$$

such that $\tilde{\rho}_k < 1$ then $\widetilde{E}_k(a, q) < \infty$. It follows from Proposition A.7 and the fact that $\min\{x > 0 \mid \tilde{\sigma}_k + \tilde{\rho}_k \cdot x = x\} = \tilde{\sigma}_k/(1 - \tilde{\rho}_k)$. Furthermore $\sup \mathcal{A}_k(q) = \max \mathcal{A}_k(q) \leqslant \tilde{\sigma}_k/(1 - \tilde{\rho}_k) < \infty$. Thus:

Remark 4.4 MWAF's only need to be known for an interval of finite length.

Necessary and/or sufficient feasibility conditions are known for different policies and under different assumptions about tasks. We give now a generic necessary and sufficient condition related to Lemma 4.3. It concerns exact bounds on WAF and is thus independent of a particular type of task.

Proposition 4.5 *For a task* τ_k *with constant relative deadline* $\overline{D}_k$, *let* $\mathcal{P}_k$ *be a set of function satisfying* (4.8). *A sufficient feasibility condition is*

$$\widetilde{R}_k \leqslant \overline{D}_k. \tag{4.9}$$

It is also a necessary condition if

$$\forall q \in Q, \forall a \in \mathcal{A}_k(q) \quad \exists\, \omega \in \Omega, n \in \mathbb{N} \quad \vdash$$

$$A_{k,n} - U_{k,n} \leqslant a \tag{4.10}$$

$$\widetilde{S}_k(x, a, q) = S_{\mathcal{I}_{k,n}}(U_{k,n}, U_{k,n} + x; \omega) \quad \forall x. \tag{4.11}$$

Proof: Since (4.9) implies $R_{k,n}(\omega) \leqslant \overline{D}_k$ for all $\omega \in \Omega$ and $n \in \mathbb{N}$, it is a sufficient condition. Equation (4.11) implies $\widetilde{E}_k(a,q) = E_{k,n} - U_{k,n}$. Using this and (4.10), gives

$$\widetilde{E}_k(a,q) - a \leqslant E_{k,n} - U_{k,n} - (A_{k,n} - U_{k,n}) = R_{k,n}(\omega).$$

It means that each possible value for $\widetilde{E}_k(a,q) - a$ is weakly bounded by some realizable response time $R_{k,n}(\omega)$ and thus $\widetilde{R}_k$ can not be larger than R_k^{max}, implying

$$\widetilde{R}_k = R_k^{max}.$$

Hence, if τ_k is feasible then $\widetilde{R}_k \leqslant \overline{D}_k$. ■

In other words, a necessary and sufficient feasibility condition can be derived from exact response time bounds. For EDF another condition exists, see Section 4.3.3.

If instances of a task may have different relative deadlines, then one should try to decompose the task into subtasks to be able to use Proposition 4.5.

In the following sections, we will use Lemma 4.3 as guideline to derive response time bounds for several known scheduling policies. Notice that because Lemma 4.3 only relies on MWAF, the results are going to be generic, that is, valid for any kind of task for which MWAF are available.

4.2 Fixed preemptive priorities (FPP)

The scheduling policy which is based on fixed or static priorities for tasks is one of the policies studied in the seminal paper [22] by Liu and Leyland. Initialy tasks were assumed to be periodic and deadlines equal to periods. The assumption about deadlines has progressively been removed and the task model enriched to sporadically periodic tasks and multi-frame tasks. Offset constraints between tasks have also been taken into account. Except for multi-frame tasks not only feasibility conditions but also response times bounds have been derived.

In this section we derive response time bounds that can be computed for any kind of tasks, provided a family of MWAF's is available. On this occasion, we discuss and illustrate the (small) differences between interference periods introduced in Section 4 and busy periods introduced by Lehoczky [20].

4.2.1 Definition

To each task is assigned an integer which represents its priority. We assume the tasks are indexed such that τ_1 is the task with the highest priority and τ_m the task with the lowest. The instances of each task are executed in the "first come first served" (FIFO) order which is the case if an earlier activated instance has a higher priority than a later activated instance. Under convention (2.37) on page 22 this can be achieved by choosing the index of the task as first and the index of the instance as second component of the priority:

$$\Gamma_{k,n}(t) = (k,n). \tag{4.12}$$

This is the direct translation into a priority assignment of the commonly used definition of fixed preemptive priorities. We will call it *canonical fixed priority assignment*. We will call any assignment which is equivalent to (4.12) a *fixed priority assignment*. Because the priorities are all different and constant, this assignment is decidable and piecewise order preserving, and thus it defines a scheduling policy.

From (2.41) in Theorem 2.28, one can see that a FPP policy is characterized by

$$\Pi_{k,n}(t) = 1 \quad \Rightarrow \quad \left\{ \begin{array}{ll} \forall n, i < k & W_{i,n}(t) = 0 \\ \forall j < n & W_{k,j}(t) = 0. \end{array} \right. \tag{4.13}$$

These implications mean that an instance can not be executed as long as an instance of a task with higher priority or a previous instance of the same task is pending. But as soon as none of these instances is pending the considered instance is immediately executed if pending. This is implied by the non-idling assumption (2.23) on page 15.

Property (4.13) actually defines a policy. We want to use it to give a definition of FPP independently of priority assignments, i.e, a definition which is as general as possible. For this purpose we have to check that the priority functions effectively determine a policy. We only have to prove that for one task process there can not exist two different scheduled task processes satisfying (4.13). Assume there would exist two different scheduling functions π and π', for the same scheduling sequence (a, c). Let then t_0 be the first time where π is different from π'. The corresponding workloads are equal for $t \leqslant t_0$. There are two symmetric cases: $\pi_{k,n}(t_0) = 1$, $\pi'_{k,n}(t_0) = 0$ and $\pi_{k,n}(t_0) = 0$, $\pi'_{k,n}(t_0) = 1$. We treat the first case. When $\tau_{k,n}$ is executed, its workload is necessarily positive $W_{k,n}(t_0) > 0$ (see (2.16) on page 12). Since at t_0 the workloads are still equal we also have $W'_{k,n}(t_0) > 0$. By the non-idling assumption, there must then exist an instance $\tau_{i,j} \neq \tau_{k,n}$ which is executed at t_0 since $\pi'_{k,n}(t_0) = 0$. Because of (4.13) it can only be an instance with $i < k$ or $i = k$ and $j < n$. But (4.13) applied to $\tau_{i,j}$ implies $W_{k,n}(t_0) = 0$, which is a contradiction.

Thus, a definition which is as general as possible is:

Definition 4.6 *A scheduled task process is said to be scheduled according to fixed preemptive priorities (FPP) if and only if the tasks can be indexed such that* (4.13) *holds.*

4.2.2 Interference and busy periods

We have seen (4.2) on page 50 how workload functions appear in the computation of response times. Because these functions contain information about the past of the process, the past before the release of an invocation has to be analyzed to determine its response time. For this purpose we have introduced the concept of *interference period*. The aim is to obtain intervals that start with no remaining work from the past. Similar concepts have already been introduced long before. In [20], Lehoczky defines a *busy period* which is then reused in a slightly modified form by Tindell et al. in [31]. We will discuss them and justify the introduction of interference periods.

To illustrate the concepts considered in this section we use Figure 9. It represents the evolution of a system with three periodic tasks on a processor, starting at the critical instant without initial workload.

	τ_1	τ_2	τ_3
C	2	1	4
T	6	14	8

Remind that at a critical instant [22] all tasks are released simultaneously. In the lower part of Figure 9 the different types of periods related to task τ_3 are represented. They are discussed hereafter. Above are shown the cumulative scheduling functions with some further details. For the task of highest priority, τ_1 the horizontal segments simply represents the executions of its instances. The first instance of τ_2 is preempted by the first instance of τ_1, which is represented by a dotted subsegment for τ_2. In the same way are represented the preempted segments of τ_3, with the additional particularity that its second instance is preempted by its first one.

4.2.2.1 Lehoczky's busy period

In [20], Lehoczky gives the following definition, where "job" can be considered as a synonym for the term "instance" used in this paper, although for us two instances can not have the same priority.

> "A level-i busy period is a time interval $[a, b]$ within which jobs of priority i or higher are processed throughout $[a, b]$ but no jobs of level i or higher are processed in $(a - \varepsilon, a)$ or $(b, b + \varepsilon)$ for sufficiently small $\varepsilon > 0$."

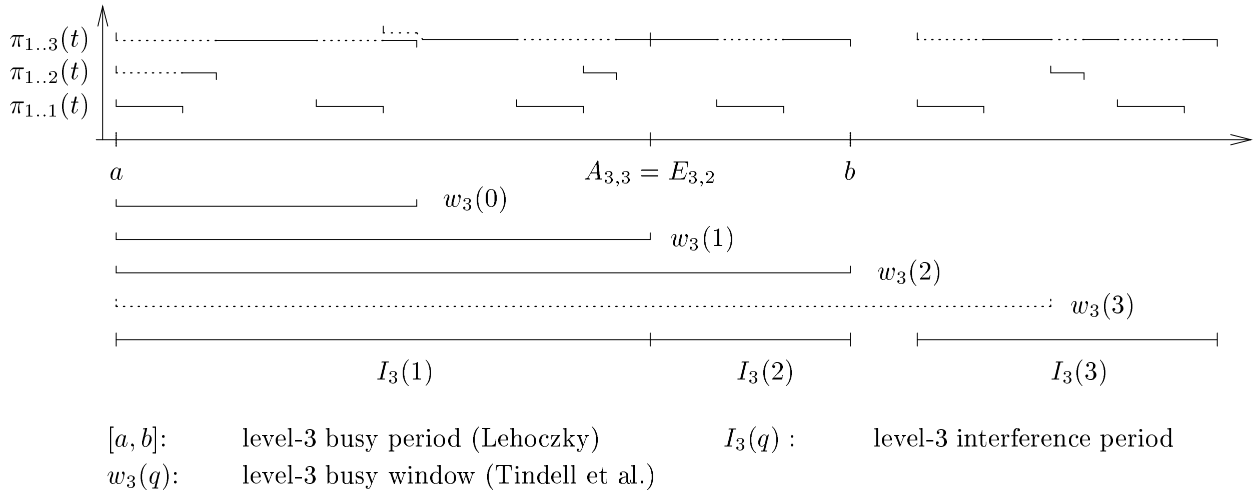


Figure 9: Critical instant for a set of 3 tasks and the different concepts of busy periods.

The main result presented in that paper is that only the busy period initialized by the critical instant needs to be analyzed in order to find the longest response time (Theorem 1).

According to this definition, the level-3 busy period in our example lasts until the execution end of the third instance of τ_3, since throughout $[A_{3,1}, E_{3,3}]$ instances of priority 3 or higher are processed but no instances of of priority 3 or higher are processed on $]A_{3,1} - \varepsilon, A_{3,1}[$ or $]E_{3,3}, E_{3,3} + \varepsilon[$.

Furthermore, the author introduces a function $W_k(n, x)$ to express a schedulability criterion.

$$W_k(n, x) = \inf_{t \leqslant x} \frac{1}{t} \cdot \left(\sum_{j=1}^{k-1} C_j \cdot \lceil \frac{t}{T_j} \rceil + nC_k \right) \tag{4.14}$$

where we used as in this paper k for the index of the task under study and n for the instances.

The function W_k determines for a time horizon x, the smallest ratio of the available processing time t and all demands that must be satisfied before the considered n^{th} instance can be completed. The instance finishes before x if and only if the ratio is smaller or equal to 1 somewhere before x. If D_n is its absolute deadline, then it is feasible if $W(n, D_n) \leqslant 1$.

According to the author, only the first $N_k = \min\{n \mid W_k(n, nT_k) \leqslant 1\}$ instances of τ_k must be checked to determine if the task is feasible. In our example, $N_k = 2$. It implies that the last instance $\tau_{3,3}$ of the busy period is not checked, although according to Theorem 1 [20] it would have to be. The reason why $W(2, 2 \cdot T_k) = 1$ is that at the release of $\tau_{3,3}$, all previous instances have finished, and there is no pending work from the past before $2 \cdot T_k$.

4.2.2.2 Tindell et al.'s busy period Tindell et al. reproduce the definition of a busy period and Lehoczky's result about worst-case response times as follows:

> A level-i busy period is defined as the maximum time for which a processor executes tasks of priority greater than or equal to the priority of tasks i. Lehoczky shows how the worst-case execution time of a task i can be found by examining a number of windows, each defined to be the length of the busy period [1] starting at the window, and each window starting at an arrival of task i (hence at some multiple of T_i before the current invocation

---

[1] starting with the critical instant

> of task i). A number of windows back in time need to be examined to find the worst-case response time (in general, the worst-case response time can be the response time corresponding to any one of the windows).

The length $w(q)$ of the above cited windows is the smallest solution of the equation

$$w_k(q) = (q+1) \cdot C_k + \sum_{j \in hp(k)} \left\lceil \frac{w_k(q)}{T_j} \right\rceil \cdot C_j, \tag{4.15}$$

where $hp(k)$ is the set of task with a higher priority than task k. In this equation, q is the number of instances of τ_k that are executed in the busy period before the instance under study.

In Figure 9 windows $w(0), \ldots, w(3)$ are represented as overlapping periods. According to Tindell's definition, a window has the length of some kind of busy period. Since (4.15) accounts for the demands of $\tau_{k,1} \ldots, \tau_{k,q}$, but not for $\tau_{k,q+1}$ nor for the following instances even if the window ends after the activation date of $\tau_{k,q+1}$, the busy periods are not level-k busy periods. The authors actually consider that from the point of view of the instance $\tau_{k,q}$ under study, the following instance has a lower priority. This is sensible for the response time computation but induces a difference with the definition of level-k busy period. In our model, these windows are level-(k,n) interference periods, see definition 4.7. Notice that the interference periods associated to instances take the particular situation at $E_{3,2}$ into account. While the first busy periods start at $U_{3,1} = U_{3,2} = 0$, the third starts at $U_{3,3} = A_{3,3}$. As will be seen below, this is the reason in our model why the response time of $\tau_{3,3}$ does not need to be analyzed. Tindell et al. use the condition

$$w(q) \leqslant (q+1)T \tag{4.16}$$

to determine the number of windows that must be examined. The condition detects the end of a window that takes place before or at the release of the next instance. In our example it implies that only two level-3 windows are analyzed though the definition of a busy period and Lehoczky's result would suggest three. The reason why the examination can effectively stop at $E_{3,2}$ is that a new level-k interference period starts at that moment and the periods the analysis is actually based on are precisely level-k interference period . The worst-case response time is realized in the level-k interference period starting with the critical instant.

4.2.2.3 Idle periods of zero length In [13] George et al. give the following definition:

> A processor busy period is an interval of time in which the processor is kept continually busy by the execution of pending instances

They add that idle periods may have zero length. In the example illustrated in Figure 9 the epoch $A_{3,3} = A_{3,2}$ is to be considered as idle period of length zero. Thus this definition is different from the one given by Lehoczky. The stopping conditions (4.16) correspond actually to Lehoczky's theorem with the definition of busy periods based on zero-length-idle-period.

4.2.2.4 Interference periods The discussion developed above shows that the concept which is really needed is based on the idea of "arrivals in the past which have an influence on the response time of the instance under study". This is another description of interference periods introduced by Definition 4.1. In the context of fixed preemptive priorities it becomes:

Definition 4.7 *The time* $U_{k,n}$ *satisfying*

$$U_{k,n} \leqslant A_{k,n}, \quad W_{1..k}(U_{k,n}) = 0 \qquad \text{and} \qquad U_{k,n} < t \leqslant A_{k,n} \Rightarrow W_{1..k}(t) > 0$$

is called beginning of the level-(k,n) interference period $[U_{k,n}, E_{k,n}[$ *associated with the* n^{th} *instance of task* τ_k.

We choose the terminology 'interference' period, because at each instant t inside such an interval the past before t interferes through a non zero workload with the present and even the future after t. The term "level-(k,n)" describes the instances that can interfere with $\tau_{k,n}$. The level has two dimensions, a first for the tasks with higher priority and a sub-level for the earlier instances of the task under study. Intuitively, the beginning of the interference period related to $A_{k,n}$ is the point where in the given priority context of the instance activated at $A_{k,n}$, there is no pending instance with higher priority at $U_{k,n}$, but at least one at every moment until $A_{k,n}$. This idea can also be used with other policies. With fixed preemptive priorities, these instances are the ones of tasks with higher priority and the previous instances of the same task. The end of the interference period is given by the completion time of the task under study. Because of the non-idling hypothesis, the interference period allows to take into account only what is essential for the the response time. It is not necessary to know the evolution of each preempting instance individually. With the interference period, it is sufficient to know the amount of presented work and the time to complete it, without bothering further about the way this is done. This was first noticed by Joseph & Pandya in [19] for periodic tasks

Notice that if $E_{k,n} > A_{k,n+1}$ then $U_{k,n} = U_{k,n+1}$, i.e. $[U_{k,n}, E_{k,n}[\subset [U_{k,n+1}, E_{k,n+1}[$. Several consecutive periods of that kind form an increasing sequence of periods. The longest one, will be called level-k interference period. In Figure 9, the level-3 interference period $I_3(1)$ contains $[A_{3,1}, E_{3,1}[$ and $[A_{3,2}, E_{3,2}[$.

Definition 4.8 *An interval* $]a, b[$ *is said to be a* level-k interference period *if it satisfies:*

$$W_{1..k}(t) > 0, \qquad \forall\, t \in]a, b[, \tag{4.17}$$

$$W_{1..k}(a) = W_{1..k}(b) = 0. \tag{4.18}$$

The level-m interference period accounts for all tasks of the task set. We call it therefore also *processor interference period.*

There are some simple relations between level-(k,n) and level-k interference periods and busy periods: It is easy to see that a level-(k,n) period is always part of some level-k interference period and that the beginning of the level-(k,n) period is necessarily the beginning of the level-k period. This plays a role in algorithm that finds response time bounds. An interference period is always included in a busy period but a busy period is only included in an interference period if they are actually equal, see Figure 9. At the end of an interference period the workload is zero, but it is possible that $\pi_{1..k}(t) = 1$ because at that very moment a task of priority k or higher is released.

Proposition 4.9

(a) *Two interference periods of the same level have an empty intersection.*

(b) *Two interference periods of different levels have an empty intersection or the period with higher priority is included in the one of lower priority.*

Proof:

(a) Trivial.

(b) Let $[a, b[$ and $[a', b'[$ be two interference periods of levels $k \leqslant k'$. Assume that their intersection is not empty. If $[a, b[$ where not included in $[a', b'[$ then a' or b' or both would fall inside $[a, b[$. But because $W_{1..k'}(a') = W_{1..k'}(b') = 0$ we would have $W_{1..k}(a') = W_{1..k}(b') = 0$, that is a zero level-k workload inside $[a, b[$, which is a contradiction with the definition.

■

For the length of an interference period there exist a similar formula as for response times of tasks:

Proposition 4.10 *The length* $L_k(U)$ *of a level-k interference period starting at* U*, is given by*

$$L_k(U) = \min\{t > 0 \mid S_{1..k}(U, U+t) = t\}, \tag{4.19}$$

with $L_k(U) = \infty$ *if the set is empty.*

Proof: Assume the period ends at $V < \infty$. From definition 4.8 and (2.14) on page 10, it follows that for $0 \leqslant t < V - U$,

$$0 < W_{1..k}(t) = W_{1..k}(U) + S_{1..k}(U, U+t) - \int_U^{U+t} \Pi_{1..k}(u)du.$$

Since $W_{1..k}(U) = 0$ and $\Pi_{1..k}(U) = 1$, see (2.23), we have

$$S_{1..k}(U, U+t) > t.$$

Using $W_{1..k}(V) = 0$, it can be proven in a similar way that $S_{1..k}(U, V) = t$ and thus, the end of the period is the first time after U, where $S_{1..k}(U, U+t) = t$. ■

From (4.19) it can be seen that the length of an interference period is like a response time the first solution of an equation as described in Proposition A.9 on page 90 and hence an iterative calculus can also be used to determine it in practice.

4.2.3 Response time bounds

First we will write the response time equation (4.2) in the the case of fixed priorities. The assignment (4.12) being time independent we can use the results of Section 4.1. The set of instances that have a higher priority has a rather simple structure, since it comprises all instances of tasks with higher priority, i.e. $k' < k$ and the previous instances of the same task, i.e. $n' < n$:

$$\mathcal{I}_{k,n} = \{\tau_{i,j} \mid i < k; \;\; i = k, \; j < n\}. \tag{4.20}$$

Thus, (4.5) in Proposition 4.2 becomes

$$E_{k,n} = \min\{t > U_{k,n} \mid S_{1..k-1}(U_{k,n}, t) + S_{k,0..n}(U_{k,n}, t) = t - U_{k,n}\}, \tag{4.21}$$

where we have used the notation

$$S_{k,0..n}(t_1, t_2) = \sum_{j \leqslant n} S_{k,j}(t_1, t_2). \tag{4.22}$$

The amount of work $S_{k,0..n}(U_{k,n}, t)$ is the simply the demand of $\tau_{k,n}$ and previous instances of τ_k in the level-k interference period starting at $U_{k,n}$.

Equation (4.21) shows as expected, that the response times of a given task τ_k is independent of task with lower priorities: $\tau_{k+1}, \ldots, \tau_m$. This has some implications for the behavior of the system. Suppose the demands of one of the real-world task are actually higher than they should, according to the assumption that define the corresponding task in the model, and such that the real-world task set isn't actually feasible. Suppose that as a result the subset of tasks $\{\tau_1, \ldots, \tau_k\}$ is stable and feasible but with task τ_{k+1} the task set overloads the processor. Then at least the tasks with a higher priority $\{\tau_1, \ldots, \tau_k\}$ are protected against any consequence this could have on the feasibility of tasks. This is a difference with the earliest deadline first policy, where a task that would demand more than it should, can cause any other task to miss its deadline. Fixed priorities allow some kind of protection levels.

For obtaining response time bounds, Lemma 4.3 means here under FPP that bounds on the functions $S_{1..k-1}(U_{k,n}, U_{k,n} + x) + S_{k,0..n}(U_{k,n}, U_{k,n} + x)$ are required. The following theorem gives them in the case where a family of MWAF is known.

Theorem 4.11
Let (A, C, D, Π) be a stable task process scheduled according to a FPP priority assignment and let $\mathcal{S} = \{\widehat{S}_k(x,q) \mid k = 1 \ldots, m;\ q \in Q\}$ be a family of MWAF for the whole task set. If

$$\widetilde{S}_k(x,a,q) \overset{\mathsf{def}}{=} \min(\widehat{S}_k(x,q), \widehat{S}_k(a^+,q)) + \widehat{S}_{1..k-1}(x,q) \tag{4.23}$$

then a bound on the response times of a task τ_k is given by

$$R_{k,n} \leqslant \max_{q\in Q} \max_{a\in\mathcal{A}_k(q)} \widetilde{E}_k(a,q) - a,$$

where $\mathcal{A}_k(q) = \{a > 0 \mid \widetilde{S}_k(x,a,q) > x,\ \forall x \leqslant a\}$ and $\widetilde{E}_k(a,q) = \min\{x > 0 \mid \widetilde{S}_k(x,a,q) = x\}$.

Proof: Consider an instance $\tau_{k,n}$ on a trajectory ω. We intend to prove (4.8) for $\widetilde{S}_k$ given by (4.23) and $S_{\mathcal{I}_{k,n}}$ under FPP, see (4.21). By definition of $\mathcal{S}$, see Definition 2.17, there exists a $q \in Q$, such that for $i = 1, \ldots, k$

$$S_i(U_{k,n}, U_{k,n} + x; \omega) \leqslant \widehat{S}_i(x,q).$$

It implies directly $S_{1..k-1}(U_{k,n}, U_{k,n} + x) \leqslant \widehat{S}_{1..k-1}(x,q)$ and furthermore

$$S_{k,0..n}(U_{k,n}, U_{k,n} + x) \leqslant \begin{cases} \widehat{S}_k(x,q) & \text{if } x \leqslant A_{k,n} - U_{k,n} \\ \widehat{S}_k((A_{k,n} - U_{k,n})^+, q) & \text{if } x > A_{k,n} - U_{k,n}. \end{cases}$$

Thus (4.8) holds and Lemma 4.3 applies. ■

Notice that $\mathcal{A}_k(q)$ could be an interval, i.e. does not necessarily contain a finite number of values. This is related to the fact that Theorem 4.11 is valid for any kind of MWAF. In the next section we consider the more concrete case of a majorizing task process, implying the finite cardinality of $\mathcal{A}_k(q)$.

4.2.4 Computing response time bounds

In this section we suppose that MWAF in $\mathcal{S}$ are are left-continuous step-functions. It implies that the family of MWAF can be represented by a task process $(\widehat{A}, \widehat{C}, \widehat{D})$. For this case we present and comment an algorithm for computing response time bounds. Notice that in general the functions $\widetilde{S}_k(x,q)$ could be defined differently than in Theorem 4.11, as in the case of conservative bounds, see Section 4.6.3. As long as the $\widetilde{S}_k(x,q)$ are left-continuous step-functions, then an algorithm very similar to the one presented here is valid.

The WAF's of the majorizing task process satisfy $\widehat{S}_k(0,x;q) = \widehat{S}_k(x,q)$. Since $\widehat{S}_k(0,a^+;q)$ only changes its values at activations $a = \hat{A}_{k,j}$, for some j, the same being thus true for $\widetilde{E}_k(a,q)$, it is sufficient to consider all $a = \hat{A}_{k,j} \in \mathcal{A}_k(q)$ to determine the maximum. If $\hat{A}_{k,j} \in \mathcal{A}_k(q)$ then $\widehat{S}_{1..k}(0,x;q) > x$ for $x \leqslant A_{k,j}$ and thus $\widehat{W}_{1..k}(x) > 0$, implying $\widehat{U}_{k,j}(q) = 0$, i.e. $\hat{A}_{k,j}$ is part of the first level-k interference period on q, see Definition 4.7. On the other hand, $\widehat{U}_{k,j}(q) = 0$ implies $\hat{A}_{k,j} \in \mathcal{A}_k(q)$. Furthermore $\widetilde{E}_k(\hat{A}_{k,j}, q) = \widehat{E}_{k,j}(q)$, i.e. it is the execution end of an instance of τ_k activated on q. Taking all this into account allows to rewrite the response time bound of a task τ_k as follows.

Corollary 4.12 *Let $(\hat{A}, \widehat{C}, \widehat{D})$ be a majorizing task process. Response time bounds under FPP are then given by*

$$R_{k,n} \leqslant \max_{q\in Q} \max_{j \,\vdash \widehat{U}_{k,j}(q)=0} \widehat{R}_{k,j}(q).$$

In other words, a response time bound of a task τ_k is the maximum of all its response times in the first level-k interference periods on all trajectories of a majorizing task process $(\widehat{A}, \widehat{C}, \widehat{D})$. Since [19] it is known that response times for periodic tasks can be computed iteratively. In Appendix A.2 we prove this result under the assumptions of our model for any kind of task. Here, for the numerical computation under FPP we have to deal with

$$\widehat{E}_{k,j}(q) = \min\{t > 0 \mid \widehat{S}_{1..k-1}(0,t;\ q) + \widehat{S}_{k,0..j}(0, \widehat{A}^{+}_{k,j}(q); q) = t\}. \tag{4.24}$$

Proposition 4.12 states that the level-k interference period containing a response time $\widehat{R}_{k,j}(q)$ must start at $\widehat{U}_{k,j}(q) = 0$. If a unique worst-case release pattern q_0 exists, e.g. the critical instant for periodic tasks, then $Q = \{q_0\}$ and necessarily $\widehat{U}_{k,0}(q_0) = 0$, or equivalently $\mathcal{A}_k(q) \neq \emptyset$. But in general, before computing the response time in the first level-k interference period of some pattern q, we have to check if $\widehat{U}_{k,0} = 0$ or equivalently that $\mathcal{A}_k(q) \neq \emptyset$, because the formula (4.24) is valid only in that case. Using it nevertheless, would give a "response time" smaller than the actual bound, and thus the final result would not be worse, but unnecessary computations would be done.

	C	T	Φ_G
τ_1	2	60	/
τ_2	12	32	0
τ_3	5	32	15

Table 1: A small task set with a transaction.

The majorizing task process $(\widehat{A}, \widehat{C}, \widehat{D})$ may indeed contain trajectories that do not need to be analyzed. To see this, let us consider a small example with parameters given in Table 1. The task set consists in a periodic task τ_1 and a transaction with period $T_{\mathcal{G}} = 32$ consisting of two other periodic tasks. Suppose we analyze τ_3. Potentially, two worst case release patterns are possible, see Figure 10:

(q_1) τ_1 and τ_2 are released at the same time, followed by τ_3 after $\Phi_{\mathcal{G},3} = 15$ units of time,

(q_2) τ_1 and τ_3 are released at the same time, followed by τ_2 after $T_G + \Phi_{\mathcal{G},2} - \Phi_{\mathcal{G},3} = 17$ units of time.

But as can be seen $\widehat{U}_{3,0}(q_1) = 15$, i.e. a level-3 interference period can never start with a release of τ_2 and thus this offset pattern would not have to be analyzed. If we had $C_1 = 6$ then we would have $\widehat{U}_{3,0}(q_1) = 0$.

Another interesting situation is where T_1 is changed to $T_1 = 6$. Since τ_1 is periodic, there would be a minimal demand $S_1(u, u+x) \geqslant \lceil (x-4)/6 \rceil \cdot 2$, implying that every release of τ_2 preempts the following release of τ_3, because $C_3 + S_1(A_{3,n}, A_{3,n} + C_3) = 12 + 2 \cdot 2 = 16 > 15 = \Phi_{G,2}$. It implies that the case (q'_2), where the interference period begins with a release of τ_3, can never occur in (A, C, D), see lower part of Figure 10. Since $\widehat{U}_{3,0}(q'_2) = 0$, the response times $R_{3,n}(q'_2)$ are computed, although it is not necessary. Whether or not the final response time bound will be worse because of that, can not be said in general.

Notice that in the example of tasks with offset relations, it is easy to determine all offset patterns that can potentially be underlying to an interference period. But in order to know which are actually realized on all trajectories of (A, C, D), one would have to construct all trajectories up to the least common multiple of the tasks periods. It is a reason why the definition of (exact) MWAF's has been chosen such as to only concern activation patterns and not interference periods.

To test if $\widehat{U}_{k,0}(q) = 0$, it is sufficient to compute

$$x^* = \min\{x > 0 \mid \widehat{S}_{1..k-1}(0, x; q) \leqslant x\}. \tag{4.25}$$

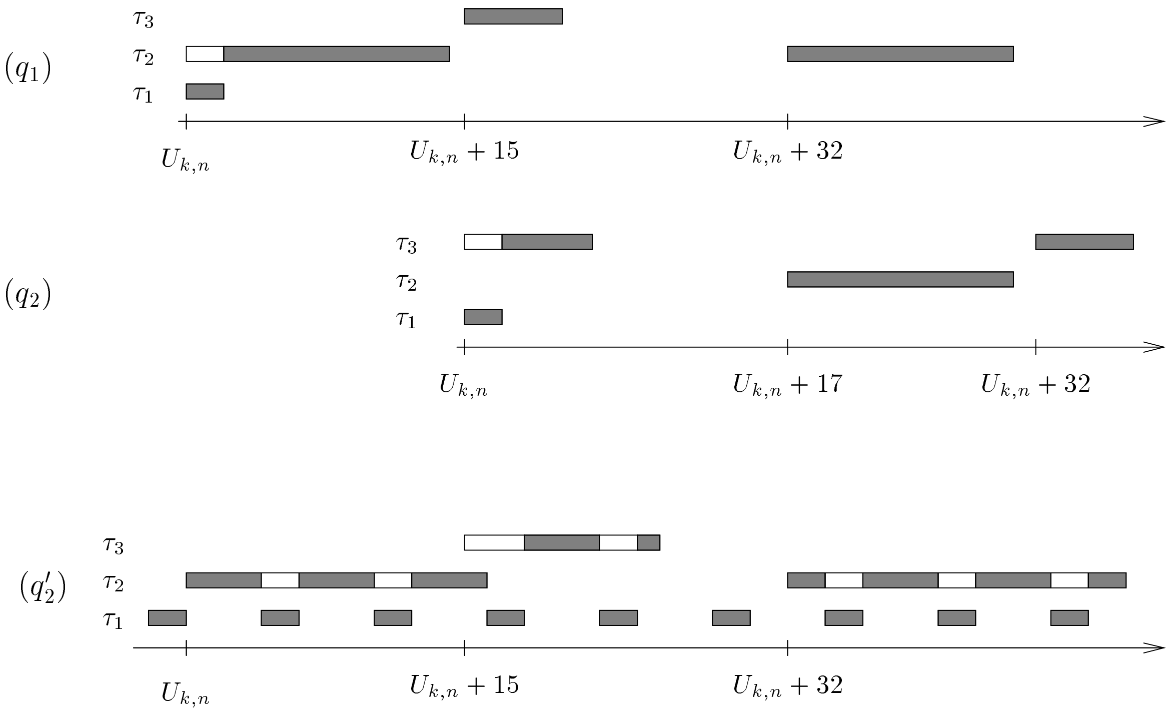


Figure 10: Potential cases for the example of Table 1.

If $x^* > \hat{A}_{k,0}$ then for all $x \in]0, \widehat{A}_{k,0}[$, $\widehat{S}_{1..k-1}(0, x; q) > x$, implying $0 < \widehat{W}_{1..k-1}(x) = \widehat{W}_{1..k}(x)$. Thus, $\widehat{U}_{k,0}(q) = 0$, see Definition 4.17 on page 68. On the other hand, $x^* \leqslant \widehat{A}_{k,0}$ implies $\widehat{U}_{k,0} > x^*$, because $\widehat{W}_{1..k-1}(x^*) = \widehat{W}_{1..k}(x^*) = 0$.

To test if $\widehat{U}_{k,j}(q) = 0$ for $j \geqslant 1$, it is only necessary to check that $\widehat{E}_{i,j-1} > \widehat{A}_{k,j}$, because then $\widehat{W}_k(\widehat{A}_{k,j}) > 0$. On the other hand, if $\widehat{E}_{i,j-1} \leqslant \widehat{A}_{i,j}$, then $\widehat{W}_k(\widehat{A}_{k,j}) = 0$, implying $\widehat{U}_{k,j} = \widehat{A}_{k,j}$. Since $\widehat{U}_{k,j+1} \geqslant \widehat{U}_{k,j}$, we have then $\widehat{U}_{k,l} > 0$ forall $l \geqslant j$, implying that no further response time $\widehat{R}_{k,l}$ needs to be computed.

The algorithm for computing the response time bound is summarized in Table 2. The function `ZeroU` returns true if the first level-k interference period starts at t=0. It is used in line 4 to test the considered case. If the case passes the test, then we know that the interference period lasts at least until $\widehat{A}_{k,0}(q)$ and thus, as initial amount of demand, we can choose $\widehat{A}_{k,0}(q) + \widehat{C}_{k,0}(q)$, see line 7. If the following instance is released after the current has finished then all following level-k interference periods do not start at $t = 0$ and no more response times need to be computed for q. This is checked in line 21.

4.2.5 The optimal priority assignment

Under the assumption that the relative deadline of each task is shorter than its period, the *deadline monotonic assignment* is optimal for periodic tasks [21]. Deadline-monotonic means the shorter the deadline, the higher the priority. This assignment is optimal, in the sense that if a feasible assignment exists, then the deadline-monotonic assignment is feasible. Such a common optimal assignment does not exist if tasks have deadlines longer than their period, if they are dependent or if the task model is more complex than that of a periodic task. Audsley [2] has developed an optimal assignment algorithm for periodic tasks, that are released with certain initial offsets such that the critical instant, where all task are activated at the same time, does not necessarily exist. The algorithm is optimal in the sense that it finds a feasible assignment if a feasible assignment exists. It has been noticed in [31], that the

```
funct $time\ RespTimeBound(task\ k)$
  $max := 0;$
  foreach $q$ in $Q$ do
    if $\neg ZeroU(k,q)$
      then continue fi;
    $n := 0;\ a := \widehat{A}_{k,0}(q);\ s := \widehat{C}_{k,0}(q);$
    $w := \widehat{A}_{k,0}(q) + \widehat{C}_{k,0}(q);$
    repeat
      repeat
        $e := w;\ w := 0;$
        for $i := 1$ to $k-1$ do
            $w := w + s(i,e,q);$
        od
        $w := w + s;$
      until $w = e;$
      $r := e - a;$
      if $r > max$ then $max := r;$ fi
      $a := a + \widehat{T}_{k,n}(q);$
      $n := n + 1;$
      $s := s + \widehat{C}_{k,n}(q)$
    until $a > e;$
  od
  return $max;$
end
```

```
funct $bool\ ZeroU(task\ k,\ case\ q)$
  $w := 1;$
  repeat
    $v := w;\ w = 0;$
    for $i := 1$ to $k-1$ do
        $w := w + s(i,v,q);$
    od
  until $w = v;$
  if $v > A_{k,0}(q)$
    then return $false$
     else return $true$ fi;
end

funct $time\ s(task\ i, time\ t, case\ q)$
  $w := 0;\ a := 0;\ j := 0;$
  while $a < t$ do
        $w := w + \widehat{C}_{i,j}(q);$
        $a := a + \widehat{T}_{i,j}(q);$
        $j := j + 1;$
  od
  return $w;$
end
```

Table 2: Computing response time bounds under FPP, for any kind of task.

algorithm is actually valid for any exact timing analysis of FPP, *where decreasing the priority of a task does not lead to a decrease in the worst-case response time of that task.* It should be added that *worst-case response times must only depend on the subset of tasks with higher priority and not on the order between those tasks.*

We will see that these properties are true for the timing analysis given in Theorem 4.11. It means that the algorithm is valid for any kind of independent or dependent tasks, whether a unique or several critical instants exists, assuming the response time bounds are derived according to Theorem 4.11.

If response time bounds are not exact, i.e longer than the worst-case response time, then the algorithm is still useful, although not optimal anymore, at least in the initial sense. It is optimal in the sense that if for a given family of MWAF's $\mathcal{S}$, there exist an assignment for which response time bounds are shorter than the relative deadlines, then the algorithm will find such an assignment.

The underlying idea of the algorithm, shown in Table 3, is to search successively for each priority level a task that is feasible at the level, starting with level m.

Let us denote $\gamma(k)$ the index of the task having at a certain step of the algorithm the priority level k. The function γ is actually a permutation of task indices. We use the usual notation based on elementary cycles. In line 2, Table 3, the permutation is initialized to the neutral element and in line 7, the next permutation is obtained by composition with the permutation $(i\ j)$, which exchanges i with j, leaving all other element where they are.

At each step k of the algorithm, the set of tasks is divided into the subset $\mathcal{A} = \{\tau_{\gamma(i)} \mid i > k\}$ of tasks, currently with priority level $i > k$ and the complementary set $\overline{\mathcal{A}} = \{\tau_{\gamma(i)} \mid i \leqslant k\}$. The task in $\mathcal{A}$ are feasible, which is an invariant property of the algorithm. The complement $\overline{\mathcal{A}}$ consists of the tasks for which the definite level still has to be determined. The task feasible at level k is determined by successively attribution of level k to each task in $\overline{\mathcal{A}}$ and to check feasibility. If a feasible task is found

```
funct perm OptimalAssignment()
  $\gamma := (1)(2)\ldots(m-1)(m)$
  $k := m$;
  while $k > 1$ do
      $j := k$;
      while $j > 0$ do
          $\gamma := (k, j) \circ \gamma$;
          if $Feasible(k,\ \gamma)$ then break fi;
          $j := j + 1$;
      od
      if $j = 0$ then return 0; fi
      $k := k - 1$;
  od
  return $\gamma$;
end
```

Table 3: Optimal priority assignment

the algorithm moves to step $k - 1$, otherwise it aborts, concluding that no feasible assignment exists. If $k = 1$ is reached all tasks are feasible under the finally obtained assignment.

Let us now return to Theorem 4.11. Recall that we usually assume that tasks are indexed in decreasing order of their priority to simplify notations. More generally, equation (4.23) should be written

$$\widetilde{S}_k(x, a, q) = \min\left(\widehat{S}_k(x, q), \widehat{S}_k(a^+, q)\right) + \sum_{i=1}^{\gamma^{-1}(k)-1} \widehat{S}_{\gamma(i)}(x, q),$$

where $\tau_{\gamma(k)}$ is the task at priority level k, i.e. with $\Gamma^1_{\gamma(k)} = k$. If γ is the result of decreasing the priority of τ_k in a set of task indexed in decreasing order of priorities, then $\gamma(i) = i,\ i < k$, because higher priority task are unaffected by the change. Thus

$$\sum_{i=1}^{\gamma^{-1}(k)-1} \widehat{S}_{\gamma(i)}(x, q) = \widehat{S}_{1..k-1}(x, q) + \sum_{i=k}^{\gamma^{-1}(k)-1} \widehat{S}_{\gamma(i)}(x, q),$$

where the second term accounts for the tasks that become higher priority tasks under γ. As a result $\widetilde{S}_k(x, a, q)$ is larger for every x and Proposition A.7 implies that $\widetilde{E}_k(x, q)$ is longer and therefore also $\widetilde{R}_k$. Hence, if the priority of a task is decreased its response time bound can not decrease.

4.2.6 A sample task set

Table 4 shows a sample task set consisting in sporadic multiframe tasks. Some of the tasks are part of sporadic transactions, which are shown in Table 6 with the corresponding offsets. The deadlines of task in transactions are chosen so that feasibility implies that these tasks are executed in the order of their offsets. Thus if the task set is feasible, then also the precedence constraints corresponding to the offsets are satisfied. For the computation of response time bounds we have used for each task a unique MWAF, which is an approximation for the multiframe tasks, but we have considered all possible offset patterns, recall Proposition 3.8 on page 41. It can be seen in Table 4 that with the deadline monotonic priority assignment some response time bounds are longer than the deadlines (the priorities increase from top to bottom). Using Audsley's algorithm, recall Section 4.2.5, gives however a feasible assignment, see Table 5.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
fpp	task19	3,2,3	1500	1400	1494	-94	0.2	83.9
	task10	30	300	1200	1073	127	10.0	83.8
	task0	95,34,53,19	600	850	834	16	8.4	73.8
	task4	7,4	100	800	655	145	5.5	65.4
	task11	92,87	1550	600	648	-48	5.8	59.9
	task15	71	1100	550	378	172	6.5	54.1
	task14	13	1550	380	379	1	0.8	47.7
	task13	15	1550	380	381	-1	1.0	46.8
	task2	4,2	100	377	355	22	3.0	45.9
	task5	156,33,170	950	350	348	2	12.6	42.9
	task7	3,3	950	250	84	166	0.3	30.3
	task18	2,4,3	1100	175	51	124	0.3	29.9
	task16	32,19	1100	170	138	32	2.3	29.7
	task8	3,9,10	950	100	87	13	0.8	27.4
	task3	3,2	100	90	85	5	2.5	26.6
	task6	8,6	950	85	82	3	0.7	24.1
	task1	9,17,32	100	80	77	3	19.3	23.3
	task9	3,2,2	950	70	45	25	0.2	4.0
	task17	40	1100	60	42	18	3.6	3.8
	task12	2	1550	50	2	48	0.1	0.1

Table 4: Response time bounds under FPP with deadline monotonic priority assignment for multiframe tasks in periodic transactions.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
fpp	task10	30	300	1200	1076	124	10.0	83.9
	task19	3,2,3	1500	**1400**	871	529	0.2	73.9
	task0	95,34,53,19	600	850	834	16	8.4	73.8
	task4	7,4	100	800	655	145	5.5	65.4
	task15	71	1100	**550**	526	24	6.5	59.9
	task11	92,87	1550	600	531	69	5.8	53.4
	task14	13	1550	380	379	1	0.8	47.7
	task2	4,2	100	377	375	2	3.0	46.8
	task13	15	1550	380	364	16	1.0	43.8
	task5	156,33,170	950	350	348	2	12.6	42.9
	task7	3,3	950	250	84	166	0.3	30.3
	task18	2,4,3	1100	175	51	124	0.3	29.9
	task16	32,19	1100	170	138	32	2.3	29.7
	task8	3,9,10	950	100	87	13	0.8	27.4
	task3	3,2	100	90	85	5	2.5	26.6
	task6	8,6	950	85	82	3	0.7	24.1
	task1	9,17,32	100	80	77	3	19.3	23.3
	task9	3,2,2	950	70	45	25	0.2	4.0
	task17	40	1100	60	42	18	3.6	3.8
	task12	2	1550	50	2	50	0.1	0.1

Table 5: Response time bounds under FPP with optimal priority assignment.

trans1	T=950		trans3	T=1100		trans2	T=1550	
Name	ϕ	D	Name	ϕ	D	Name	ϕ	D
task5	0	350	task15	0	550	task12	984	50
task6	366	85	task16	596	170	task13	1018	380
task7	456	250	task17	668	60	task14	1423	380
task8	713	100	task18	730	175			
task9	884	70						

Table 6: Periodic transactions of the sample task set.

4.3 Preemptive earliest deadline first (EDF)

EDF is the other policy studied in the seminal paper by Liu and Leyland. The underlying idea is to execute instances in the order of their urgency given by their absolute deadline. It has the appealing property of being optimal in the sense that any feasible scheduling can be transformed into an earliest deadline first scheduling, [12]. Its disadvantage is perhaps that in case of overload, any of the tasks can miss its deadline, whereas with FPP, only tasks with lower priority miss their deadline. Different strategies exist for discarding instances in overload situations under EDF, see [8] and references therein. Furthermore, adding a task to a given set has an impact on the response times of all tasks and hence a hierarchical design, where response times of tasks in some subset are independent of all other tasks, is not possible under EDF. In general, any task can influence the response time of any other task.

In the case of periodic tasks with constant execution times and relative deadlines equal to the period, a task set is feasible if and only if its load is smaller than 1, [22]. The test becomes more complex if deadlines are not necessarily equal to periods, see [13], or if the task is for example sporadically periodic, see [26]. The complexity of direct feasibility tests is in general smaller than that of the computation of response times bounds [17], but the latter do not only allow to decide upon feasibility but also to compute end-to-end response times.

In this section we develop a response time analysis in the framework of our model and show the connections with the results and concepts presented by Spuri in [26]. We extend the EDF timing analysis to any kind of task. As new result this allows to compute response time bounds under EDF for multiframe tasks and to account for offset constraints.

4.3.1 Definition

The *earliest deadline first rule* can be expressed formally as

$$\Pi_{k,n}(t) = 1 \qquad \Rightarrow \qquad D_{i,j} < D_{k,n} \Rightarrow W_{i,j}(t) = 0, \tag{4.26}$$

meaning that a task can only be executed if all instances with shorter deadlines have finished or are not yet active. It does not tell in which order instances with the same absolute deadline are to be scheduled. Thus (4.26) determines a class and not a single policy.

Definition 4.13 *A scheduled task process is said to be scheduled according to a* preemptive earliest deadline first *(EDF) policy if it satisfies* (4.26).

The EDF-rule (4.26) is implied by the HPF-rule (4.26), if to each instance is attributed its absolute deadline as priority. But to be able to perform the scheduling, the assignment must be decidable, which can only be achieved by adding a further rule. For our analysis, we choose fixed priorities to decide which task to execute when deadlines are equal:

$$\Gamma_{k,n}(t) = (D_{k,n}, k, n). \tag{4.27}$$

The response time bounds derived below will however not depend on this particular choice (4.27). Any assignment such that (A, C, D, Π) satisfies the EDF rule, will be called earliest deadline first priority assignment. Notice that instances of the same task are executed in the "first activated, first executed" order if $D_{k,n} \leqslant D_{k,n+1}$, but this is not required for our analysis.

4.3.2 Response time bounds

EDF is often termed *dynamic* in opposition to FPP, since from the task's point of view the priority order changes depending on the currently active instances. Nevertheless, the priority assignment, as defined in our model, is time independent and thus the set $\mathcal{I}_{k,n}$ of instances with a higher or equal priority than $\tau_{k,n}$, can be considered.

$$\begin{aligned} \mathcal{I}_{k,n} = \{\tau_{i,j} \mid \quad & i < k,\ D_{i,j} \leqslant D_{k,n}; \qquad i = k,\ j \leqslant n,\ D_{i,j} \leqslant D_{k,n}; \\ & i > k,\ D_{i,j} < D_{k,n}; \qquad i = k,\ j > n,\ D_{i,j} < D_{k,n} \quad \}. \end{aligned}$$

The apparent complexity is due to the fact that instances fall into four classes, according to the different cases that arise when instances have the same deadline.

We intend to derive response times bounds with the help of Lemma 4.3. Recall that we have to find a set of functions $\mathcal{P}_k = \{\widetilde{S}_k(x, a, q) \mid q \in Q\}$ such that

$$S_{\mathcal{I}_{k,n}}(U_{k,n}, x) \leqslant \widetilde{S}_k(x, A_{k,n} - U_{k,n}, q). \tag{4.8}$$

The instance in $\mathcal{I}_{k,n}$ that must satisfy a strict inequality $D_{i,j} < D_{k,n}$ on their deadline need special attention. Inequality (4.8) shall hold independently of the considered instance and trajectory, but in general no minimal distance between $D_{i,j}$ and $D_{k,n}$ exists, if $D_{i,j} \neq D_{k,n}$. Hence in order to derive a bound, we are obliged to use $D_{i,j} \leqslant D_{k,n}$ instead. Replacing strict inequalities by weak inequalities leads to considering $\mathcal{I}'_{k,n} \overset{\mathsf{def}}{=} \{\tau_{i,j} \mid D_{i,j} \leqslant D_{k,n}\}$. Since $\mathcal{I}_{k,n} \subset \mathcal{I}'_{k,n}$, we have thus

$$S_{\mathcal{I}_{k,n}}(U_{k,n}, t) \leqslant S_{\mathcal{I}'_{k,n}}(U_{k,n}, t) = S_{1..m}(U_{k,n}, t, D_{k,n}). \tag{4.28}$$

Remark: Remind that we have chosen fixed priorities as second and third priority components to render the EDF priority assignment decidable. For any other choice, $\widetilde{\mathcal{I}}_{k,n}$ would also be a majorizing set of instances and thus the response time bound derived hereafter is valid for every policy satisfying the EDF rule (4.26).

Based on that bound the following theorem gives response times in the case where a family of deadline based MWAF is known.

Theorem 4.14
Let (A, C, D, Π) be a stable task process scheduled according to an EDF priority assignment and let $\mathcal{S} = \{\widehat{S}_k(x, d, q) \mid k = 1, \dots, k;\ q \in Q\}$ be a family of deadline based MWAF for the task set. Tasks are supposed to have constant relative deadlines $\overline{D}_k$. If

$$\widetilde{S}_k(x, a, q) \overset{\mathsf{def}}{=} \widehat{S}_{1..m}(x, a + \overline{D}_k, q) \tag{4.29}$$

then a bound on the response times of a task τ_k is given by

$$R_{k,n} \leqslant \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a; q) - a,$$

where $\mathcal{A}_k(q) = \{a > 0 \mid \widetilde{S}_k(x, a, q) > x,\ \forall x \leqslant a\}$ and $\widetilde{E}_k(a, q) = \min\{x > 0 \mid \widetilde{S}_k(x, a, q) = x\}$.

Proof: Consider an instance $\tau_{k,n}$ on a trajectory ω. We intend to prove (4.8) for $\widetilde{S}_k$ given by (4.29) and $S_{\mathcal{I}_{k,n}}$ under EDF. By definition of $\mathcal{S}$, see Definition 2.18, there exists a $q \in Q$, such that for $i = 1, \dots, k$

$$S_i(U_{k,n}, U_{k,n} + x, U_{k,n} + d; \omega) \leqslant \widehat{S}_i(x, d, q). \tag{4.30}$$

Now, using (4.28), $D_{k,n} = A_{k,n} + \overline{D}_k$, (4.30) and (4.29) gives successively:

$$\begin{aligned} S_{\mathcal{I}_{k,n}}(U_{k,n}, U_{k,n} + x; \omega) &\leqslant S_{1..m}(U_{k,n}, U_{k,n} + x, D_{k,n}; \omega) \\ &= S_{1..m}(U_{k,n}, U_{k,n} + x, A_{k,n} + \overline{D}_k; \omega) \\ &\leqslant \widehat{S}_{1..m}(x, A_{k,n} - U_{k,n} + \overline{D}_k, q) \\ &= \widetilde{S}_{1..m}(x, A_{k,n} - U_{k,n}, q). \end{aligned}$$

Thus, (4.8) holds and Lemma 4.3 applies. ■

Again, the sets $\mathcal{A}_k(q)$ do not necessarily contain a finite number of points, unless some further assumption on the shape of the MWAF is imposed. As could be seen in equation (4.29), the function $\widetilde{S}_k$ is depending on τ_k only through $\overline{D}_k$. To make this clearer we can restate the result as:

Corollary 4.15 *Let $(\hat{A}, \widehat{C}, \widehat{D})$ be a majorizing task process. Response time bounds are then given by:*

$$R_{k,n} \leqslant \max_{q \in Q} \max_{d - \overline{D}_k \in \mathcal{A}_k(q)} \widetilde{V}_k(d, q) - (d - \overline{D}_k),$$

where $\widetilde{V}_k(d, q) \overset{\mathsf{def}}{=} \min\{x > 0 \mid \widehat{S}_{1..m}(x, d, q) = x\}$.

If the instances of a task may have different relative deadlines, then one can either try to decompose it into subtasks with constant relative deadlines or replace $\overline{D}_k$ by

$$\overline{D}_k^{max} = \max_{n \in \mathbb{N}, \omega \in \Omega} \overline{D}_{k,n}.$$

The latter solution gives a more conservative result, while the former necessitates longer computations.

4.3.3 Feasibility

Under EDF a direct feasibility test exists, which does not rely on response time bounds. It is based on the work that must be completed before a certain deadline. The following result has been proven for different kinds of tasks in the literature. We give its proof in the general context of our model.

Theorem 4.16
Let $(\hat{A}, \widehat{C}, \widehat{D})$ be an exact deadline based majorizing task process. A necessary and sufficient feasibility condition is

$$\forall\, q \in Q,\ d \leqslant L \qquad \widehat{S}_{1..m}(0, d, d; q) \leqslant d. \tag{4.31}$$

Proof:

N.C: for $a = d - \overline{D}_k$, (4.31) implies $\widetilde{E}_k(a, q) \leqslant a + \overline{D}_k$. Thus, by Theorem 4.14 $R_{k,n} \leqslant \overline{D}_k$, $\forall\, n \in \mathbb{N}$.

S.C: since $(\hat{A}, \widehat{C}, \widehat{D})$ is supposed to be an exact majorizing task process:

$$\forall\, q,\ \exists\, \omega,\ u\ \vdash S_k(u, u + x, u + d; \omega) = \widehat{S}_k(x, d; q) \quad \forall\, x,\ d.$$

Thus, if $\widehat{S}_{1..m}(0, d, d; q) > d$ then $S_{1..m}(u, u + d, u + d; \omega) > d$. It implies that there are pending instances: $W_{1..m}(u + d, u + d; \omega) > 0$. But if $W_{i,j}(u + d, u + d; \omega) > 0$ then $D_{i,j} < u + d$ and thus $E_{i,j} > D_{i,j}$. ■

The fact that the feasibility condition is necessary and sufficient is a particularity of EDF scheduling policy. In [17] is has been shown that no such test exists for FPP. The reason why the test is exact under EDF is essentially that priorities of instances are chosen according to their deadlines, which is not the case for FPP, where any instance of a higher priority task is executed even if the instance of a lower priority task is closer to its deadline.

4.3.4 Computing response times

In this section we suppose that the family of MWAF $\mathcal{S}$ can be represented by a majorizing task process $(\widehat{A}, \widehat{C}, \widehat{D})$, i.e. $\widehat{S}_k(0, x, d; q) = \widehat{S}_k(x, d, q)\ \forall\ x, d$. A necessary, but not sufficient condition, is that each function in $\mathcal{S}$ is an increasing, step-function, left-continuous in x and right-continuous in d.

Written in terms of the underlying process, $d - \overline{D}_k \in \mathcal{A}_k(q)$ implies $\widehat{S}_{1..m}(x, d; q) - x > 0\ \forall\ x \leqslant d - \overline{D}_k$. It means a positive workload in a certain interval:

$$x \in]0, d - \overline{D}_k], \quad \widehat{W}_{1..m}(x, d; q) > 0,$$

which shows some similarities with interference periods. Let us thus formalize the idea for a generic process (A, C, D, Π) and analyze the concept.

Definition 4.17 *An interval* $[U, V[$ *such that*

$$W_{1..m}(U, d) = W_{1..m}(V, d) = 0 \qquad \text{and} \qquad U < t < V \Rightarrow W_{1..m}(t, d) > 0$$

is called deadline-d interference period.

Notice that it is different from Definition 4.1, because it only uses deadlines. The end $V_i(d)$ of the i^{th} deadline-d interference period can be computed, assumed the starting time $U_i(d)$ is known:

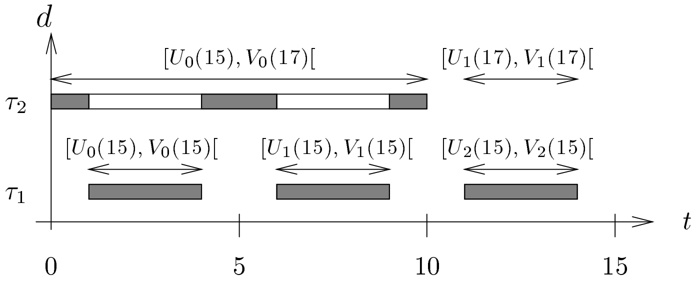
$$V_i(d) = \min\{t > U_i(d) \mid W_{1..m}(t, d) = 0\}.$$

This formula immediately follows from Definition 4.17. Indeed, inside the interference period $[U_i(d), V_i(d)[$, the scheduling function $\Pi_{1..m}(t)$ is equal to one and $W_{1..m}(U_i(d), d) = 0$ by definition. Thus,

$$W_{1..m}(t, d) = S_{1..m}(U_i(d), t, d) - (t - U_i(d)).$$

Using this relation and changing the dummy variable (see Lemma A.5) leads to

$$V_i(d) = U_i(d) + \min\{x > 0 \mid S_{1..m}(U_i(d), U_i(d) + x, d) = x\}. \tag{4.32}$$



	$A_{k,0}$	T_k	C_k	$\overline{D}_k$
τ_1	1	5	3	4
τ_2	0	20	4	17

Figure 11: Deadline-d interference periods, for increasing d.

To see that indeed, for a given deadline d, several distinct deadline-d interference periods may exist, consider the example of two tasks shown in Figure 11. For $d = 15$ we obtain $\mathcal{I}(15) = \{\tau_{1,0}, \tau_{1,1}, \tau_{1,2}\}$, to which correspond three distinct periods. At $d = 17$, the first instance of τ_2 becomes relevant, $\mathcal{I}(17) = \mathcal{I}(15) \cup \{\tau_{2,0}\}$, merging the first two periods into one. In general, with increasing deadline, the first of these periods increases irregularly, a bound being any bound on processor busy periods.

In [13] so called deadline-d busy period are defined by

(i) A deadline-d busy period is a processor busy period in which only task instances with absolute deadline smaller than or equal to d execute.

(ii) An idle-period can have zero length.

The first part taken as such could be confusing in some situation. Suppose an instance $\tau_{i,j}$ ends at the time where another instance $\tau_{k,n}$ is activated: $E_{i,j} = A_{k,n}$. Suppose both have deadlines smaller than some d and that at $E_{i,j}$, $\tau_{i,j}$ was the last pending instance to finish with deadline smaller than d: $W_{1..m}(E_{i,j}, d) = 0$. Because at the same moment $\tau_{k,n}$ is activated, $W_{1..m}(E^+_{i,j}, d) > 0$. At $E_{i,j} = A_{k,n}$ the processor can be considered as being working or busy without interruption since there is no period of time where the processor is idling, but because at $E_{i,j} = A_{k,n}$ the past has no influence on the future and thus on following response times, one would like to have the period end at that moment. To achieve this the authors "complete" their definition by considering this epoch as an idle period of zero length. A more straightforward way is to define the needed concept using workloads as in Definition 4.17. We use the term "interference" instead of "busy" by analogy with level-k interference periods.

Now we see that $d - \overline{D}_k \in \mathcal{A}_k(q)$ means for $(\widehat{A}, \widehat{C}, \widehat{D}, \widehat{\Pi})$

$$\widehat{U}_0(d; q) = 0, \qquad a = d - \overline{D}_k \leqslant \widehat{V}_0(d; q), \tag{4.33}$$

i.e. the first deadline-d interference period associated with $d = a + \overline{D}_k$ starts at $t = 0$ and contains $a = d - \overline{D}_k$. Notice also that $\widetilde{V}_k(d, q) = \widehat{V}_0(d; q)$.

The interference period end $\widehat{V}_0(d; q)$ must be computed for a certain range of d. Since it changes its value if d is equal to some deadline $\widehat{D}_{i,j}$, it sufficient to consider only these values, and $d = \overline{D}_k$ to determine the maximum. Taking also (4.33) into account leads us finally to the following alternative formulation of the response time bound.

Corollary 4.18 *Given a majorizing task process, the response time bounds under EDF are:*

$$R_{k,n} \leqslant \max_{q \in Q} \max_{d \in \mathcal{D}_k} \widehat{V}_0(d; q) - (d - \overline{D}_k),$$
$$\mathcal{D}_k = \{\overline{D}_k\} \cup \{\widehat{D}_{i,j}(q) > \overline{D}_k \mid \widehat{U}_0(\widehat{D}_{i,j}(q); q) = 0\}.$$

In other words, in order to get the response times bounds of all tasks it is only necessary to compute all different possible deadline interference period ends - which are independent of the analyzed task - and to derive then for each task the virtual response times, the maximum of which is the tasks response time bound.

To test if $\widehat{U}_0(d; q) = 0$, it is sufficient to check if there is an activation $\widehat{A}_{i,j} = 0$ with $\widehat{D}_{i,j} \leqslant d$, because then $\widehat{W}_{1..m}(0^+, d) > 0$, see Definition 4.17, or equivalently $\widehat{S}_{1..m}(0, 0^+, d) > 0$. Thus, the test about the beginning of a first level-d interference period can be based on

$$\widehat{U}_0(d; q) = 0 \qquad \Leftrightarrow \qquad \widehat{W}_{1..m}(0^+, d) = 0. \tag{4.34}$$

In Table 7 we show an algorithm which computes the response times bounds for a given task.

1. In lines 7- 12, the current $\widehat{V}_0(d; q)$ is computed using the deadline based WAF's $S_k(0, t, d)$, the values of which are returned by the function *sd*.

2. In line 4, a and d are initialized with the smallest possible values. In line 20, the function *NextD* returns the next larger possible value for d. Deadlines of different instances could be equal, so *NextD* should avoid returning the same value several times. Since $\widehat{V}_0(d)$ is increasing in d we initialize w only once in line 5 for each case q, to speed up convergence.

3. Because $\widehat{V}_0(d)$ is smaller than the longest possible processor interference period $pip(q)$, we can exit the loop in two cases. On one hand, if $w = pip(q)$, because with larger d, also a is larger, but $\widehat{V}_0(d)$ could not increase and thus $\widehat{V}_0(d) - (d - \overline{D}_k)$ only decreases, see line 19. On the other hand, since we must have $a \leqslant \widehat{V}_0(d)$ we can exit the loop as soon as $a > pip(q)$, see line 22.

4. The cases $\widehat{U}_0(d; q) > 0$, which are not to be considered, give $w = 0$. In that case, the loop exits because $e = 1$. Furthermore, w must be reinitialized, see line 17, because in the loop it is supposed to be positive.

```
1  funct time RespTimeBound(task k)
2    max := 0;
3    foreach q in Q do
4      a := 0; d := $\overline{D}_k$;
5      w := 1;
6      repeat
7        repeat
8          e := w; w := 0;
9          for i := 1 to m do
10             w := w + sd(k, e, d, q);
11         od
12       while w > e;
13       if w > 0 then
14         r := e − a;
15         if r > max then max := r; fi
16       else
17           w := 1;
18       fi
19       if w = pip(q) then break fi
20       d := NextD(d);
21       a := d − $\overline{D}_k$;
22     until a > pip(q);
23   od
24   return max;
25 end
```

```
1  funct time pip(case q)
2    w := 1;
3    repeat
4      v := w; w = 0;
5      for i := 1 to m do
6          w := w + s(k, v, q);
7      od
8    until w = v;
9    return v;
10 end
11
13 funct time sd(task i, time t, time d, case q)
14   w := 0; a := 0; j := 0;
15   while (a < t) ∧ ($\widehat{D}_{i,j}(q) \leqslant d$) do
16         w := w + $\widehat{C}_{i,j}(q)$;
17         a := a + $\widehat{T}_{i,j}(q)$;
18         j := j + 1;
19   od
20   return w;
21 end
```

Table 7: Computing the response time bounds under EDF, for any kind of task.

4.3.5 Comparison with existing results

Response time bounds for sporadically periodic tasks under EDF, have been derived by Spuri in [26]. The bounds given by Corollary 4.15 or more generally By Theorem 4.14, are also valid for DAG-tasks and the other kinds of tasks discussed in Section 3.2. Notice in particular that Corollary 4.15 allows to compute response time bounds for tasks with offset constraints, which was so far only known under FPP.

There is a difference between (4.29) and Spuri's approach. We consider the formulation given in [13] or [28]. The bounds are based on the functions

$$f_k(x, a) = \sum_{i \neq k} \min\left(\left\lceil \frac{x}{T_i} \right\rceil, \left\lfloor \frac{a + \overline{D}_k - \overline{D}_i}{T_i} \right\rfloor + 1\right) \cdot C_i + \left(\left\lfloor \frac{a}{T_k} \right\rfloor + 1\right) \cdot C_k \tag{4.35}$$

and $L_k(a) = \min\{x > 0 \mid f_k(x, a) = x\}$. The bound for a task τ_k is given by

$$\max_{a \in \mathcal{A}_k} L_k(a) - a$$

where $\mathcal{A}_k = \{n \cdot T_i + \overline{D}_i - \overline{D}_k \mid\ n \in \mathbb{N},\ i = 1..m\} \cap [0, L_k[$. The release times which are relevant for the considered task are strictly smaller than the end of some deadline busy period L_k, which can be expressed as $L_k = \max\{a \in \mathcal{A}_k \,|\, a \leqslant L,\ L_k(a) - C_k > a\}$. In this expression, L is the longest processor interference period. In [13] or [28] an algorithm is given to determine L_k.

For sporadic tasks, the function we use in Corollary 4.15 to compute the lengths of deadline interference periods $V_0(d)$ is

$$\widehat{S}_{1..m}(0, x, d) = \sum_{i=1..m} \min\left(\left\lceil \frac{x}{T_i} \right\rceil, \left\lfloor \frac{d - \overline{D}_i}{T_i} \right\rfloor + 1\right) \cdot C_i \tag{4.36}$$

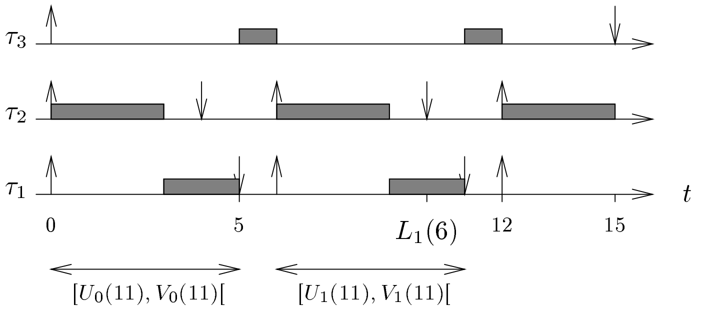
where $d \in \mathcal{D}_k = \{\overline{D}_k\} \cup \{d = n \cdot T_i + \overline{D}_i \mid d > \overline{D}_k,\ \widehat{U}_0(d) = 0\ n \in \mathbb{N},\ i = 1..m\}$. The response time bound for τ_k is

$$\max_{d \in \mathcal{D}_k} V_0(d) - (d - \overline{D}_k).$$

Notice the relation

$$a = d - \overline{D}_k.$$

To compare the two approaches let us consider for a simple task set the release pattern which is underlying to (4.36) and (4.35), see Figure 12. It can be seen that the longest processor interference period is $L = 12$, since the workload, that is the "pending work from the past", is zero at $t = 12$ ($\widehat{W}_{1..m}(12) = 0$) for the first time after the critical instant $t = 0$. Let τ_1 be the task under study. We have for example $L_1(6) = 10$, since for $a = 6$ i.e. $d = 11$, the recursive computation gives



	T_k	C_k	$\overline{D}_k$
τ_1	6	2	5
τ_2	6	3	4
τ_3	20	2	15

Figure 12: Response time bound for $a = 6$.

$$\begin{aligned} f_1(0, 6) &= 2 \cdot 2 = 4 \\ f_1(4, 6) &= 3 + 2 \cdot 2 = 7 \\ f_1(7, 6) &= 2 \cdot 3 + 2 \cdot 2 = 10 \\ f_1(10, 6) &= 2 \cdot 3 + 2 \cdot 2 = 10. \end{aligned}$$

Since $L_1(6) - C_1 = 8 > 6$, i.e. $a = 6 \in \mathcal{A}_1$, the term $L_1(6) - 6 = 4$ is indeed a value to be computed, with Spuri's approach. With our approach, the "corresponding" value is the end of the first deadline-d interference period for $d = 11$. As shown in Figure 12, we have $V_0(11) = 5$ and thus $V_0(11) - 6 = -1$. The reason for the difference is that seen as a WAF, (4.36) is bounded above by (4.35), for a given $a = d - \overline{D}_k$ (because of the term that accounts for the demands of the task under study) and thus with (4.36) the recursive computation converges earlier.

Actually, the values $L_1(6) - 6 = 4$ and $V_0(11) - 6 = -1$ do not need to be computed to determine the exact response time bound, since the release pattern after $t = 6$ is bounded by the release pattern after $t = 0$

$$\widehat{S}_{1..m}(6, 6 + x, 6 + d) \leqslant \widehat{S}_{1..m}(0, x, d).$$

If the approach based on (4.35) could give less tight bounds than our approach can not be said. The important advantage however is that (4.36) does not depend on the task under study τ_k and thus the values $V_0(d)$ do not need to be computed individually for each task but only once for the whole task set; the final values $V_0(d) - (d - \overline{D}_k)$ are of course particular for each task. As a result the complexity of the response time computation is reduced, see Section 4.6.1 for further details.

Notice that if the task set is feasible, Theorem 4.16 implies that the response time bound for a task τ_k can not be longer than its deadline $\overline{D}_k$:

Proposition 4.19 *A feasibility test based on the response time bounds derived from an exact majorizing task process according to Theorem 4.14 is a necessary and sufficient feasibility test.*

4.3.6 A sample task set

Table 8 shows the response time bounds under EDF for the task set already considered in Section 4.2.6. The optimality of EDF implies that the set is also feasible.

Suppose that the assumption about the worst execution time of a task is actually not satisfied by a task. As example we have changed the execution time of task `task13` to $C_k = 35$ so that the task set comes infeasible. The result is different for the two policies. Under FPP only tasks with lower priority than `task13` can become infeasible. Under EDF on the other hand, any of the tasks can become infeasible, compare Tables 10 and 9.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
edf	task10	30	300	1200	861	339	10.0	83.9
	task19	3,2,3	1500	1400	1061	339	0.2	73.9
	task0	95,34,53,19	600	850	725	125	8.4	73.8
	task4	7,4	100	800	675	125	5.5	65.4
	task15	71	1100	550	425	125	6.5	59.9
	task11	92,87	1550	600	475	125	5.8	53.4
	task14	13	1550	380	361	19	0.8	47.7
	task2	4,2	100	377	360	17	3.0	46.8
	task13	15	1550	380	363	17	1.0	43.8
	task5	156,33,170	950	350	333	17	12.6	42.9
	task7	3,3	950	250	125	125	0.3	30.3
	task18	2,4,3	1100	175	158	17	0.3	29.9
	task16	32,19	1100	170	153	17	2.3	29.7
	task8	3,9,10	950	100	87	13	0.8	27.4
	task3	3,2	100	90	82	8	2.5	26.6
	task6	8,6	950	85	82	3	0.7	24.1
	task1	9,17,32	100	80	77	3	19.3	23.3
	task9	3,2,2	950	70	45	25	0.2	4.0
	task17	40	1100	60	57	3	3.6	3.8
	task12	2	1550	50	47	3	0.1	0.1

Table 8: Response time bounds under EDF for multiframe tasks in periodic transactions.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
edf	task10	30	300	1200	902	298	10.0	85.2
	task19	3,2,3	1500	1400	1102	298	0.2	75.2
	task0	95,34,53,19	600	850	745	105	8.4	75.1
	task4	7,4	100	800	695	105	5.5	66.7
	task15	71	1100	550	445	105	6.5	61.2
	task11	92,87	1550	600	495	105	5.8	54.7
	task14	13	1550	380	361	19	0.8	49.0
	task2	4,2	100	377	380	-3	3.0	48.1
	task13	**35**	1550	380	383	-3	2.3	45.1
	task5	156,33,170	950	350	353	-3	12.6	42.9
	task7	3,3	950	250	145	105	0.3	30.3
	task18	2,4,3	1100	175	178	-3	0.3	29.9
	task16	32,19	1100	170	173	-3	2.3	29.7
	task8	3,9,10	950	100	87	13	0.8	27.4
	task3	3,2	100	90	93	-3	2.5	26.6
	task6	8,6	950	85	82	3	0.7	24.1
	task1	9,17,32	100	80	83	-3	19.3	23.3
	task9	3,2,2	950	70	45	25	0.2	4.0
	task17	40	1100	60	63	-3	3.6	3.8
	task12	2	1550	50	47	3	0.1	0.1

Table 9: Response time bounds under EDF in the case where a task violates the assumption about its execution time.

Layer	Task	C^{max}	T^{min}	D^{min}	bound	laxity	ρ_k	$\rho_{1..m}$
fpp	task10	30	300	1200	1366	-166	10.0	85.2
	task19	3,2,3	1500	1400	891	509	0.2	75.2
	task0	95,34,53,19	600	850	854	-4	8.4	75.1
	task4	7,4	100	800	675	125	5.5	66.7
	task15	71	1100	550	546	4	6.5	61.2
	task11	92,87	1550	600	551	49	5.8	54.7
	task14	13	1550	380	379	1	0.8	49.0
	task2	4,2	100	377	395	-18	3.0	48.1
	task13	**35**	1550	380	389	-9	2.3	45.1
	task5	156,33,170	950	350	348	2	12.6	42.9
	task7	3,3	950	250	84	166	0.3	30.3
	task18	2,4,3	1100	175	51	124	0.3	29.9
	task16	32,19	1100	170	138	32	2.3	29.7
	task8	3,9,10	950	100	87	13	0.8	27.4
	task3	3,2	100	90	85	5	2.5	26.6
	task6	8,6	950	85	82	3	0.7	24.1
	task1	9,17,32	100	80	77	3	19.3	23.3
	task9	3,2,2	950	70	45	25	0.2	4.0
	task17	40	1100	60	42	18	3.6	3.8
	task12	2	1550	50	2	48	0.1	0.1

Table 10: Response time bounds under FPP in the case where a task violates the assumption about its execution time.

4.4 First in first out (FIFO)

4.4.1 Definition

The idea behind this policy is simply to serve clients in the order of their arrival without interruption such that an instance which arrives first is leaving first. This rule can be expressed as

$$\Pi_{k,n}(t) = 1 \quad \Rightarrow \quad A_{i,j} < A_{k,n} \Rightarrow W_{i,j}(t) = 0. \tag{4.37}$$

For the case where two instances arrive at the same time an additional decision rule is needed to make the associated priority assignment decidable. We use their index as under EDF to obtain decidability:

$$\Gamma_{k,n}(t) = (A_{k,n}, k, n). \tag{4.38}$$

Definition 4.20 *A scheduled task process is said to be scheduled according to a* first in first out *rule if it satisfies* (4.37).

4.4.2 Response time bounds

The priority assignment being time independent, the results of Section 4 apply. An instance $\tau_{i,j}$ is in the set $\mathcal{I}_{k,n}$ of instances with higher priorities of a given task $\tau_{k,n}$ in the following cases:

1. $i < k$ and $A_{i,j} \leqslant A_{k,n}$
2. $i = k$ and $i < n$
3. $i > k$ and $A_{i,j} < A_{k,n}$.

Consider the response time formula (4.2). Since $\mathcal{I}_{k,n}$ does not contain any instance activated strictly after $A_{k,n}$, we have $S_{\mathcal{I}_{k,n}}(A^+_{k,n}, A_{k,n} + t) = 0$. Furthermore, since all instances activated strictly before $A_{k,n}$ are in $\mathcal{I}_{k,n}$, we have $W_{\mathcal{I}_{k,n}}(A_{k,n}) = W_{1..m}(A_{k,n})$. Thus,

$$R_{k,n} = W_{\mathcal{I}_{k,n}}(A^+_{k,n}) = W_{1..m}(A_{k,n}) + S_{\mathcal{I}_{k,n}}(A_{k,n}, A^+_{k,n}).$$

The term $S_{\mathcal{I}_{k,n}}(A_{k,n}, A^+_{k,n})$ accounts for the different cases of equal activation times. Now, a similar difficulty as with equal deadlines under EDF arises. If $A_{k,n} \neq A_{i,j}$, then there is in general no minimal distance between $A_{k,n}$ and $A_{i,j}$. Recall that we need a bound that holds for any instances $\tau_{k,n}$ and any trajectory such that a response time bound can be computed from $(\widehat{A}, \widehat{C}, \widehat{D})$ only. To achieve this, we must replace the strong inequalities by weak inequalities: because the set $\mathcal{I}'_{k,n} = \{\tau_{i,j} \mid A_{i,j} \leqslant A_{k,n}\}$, includes $\mathcal{I}_{k,n}$, we have

$$R_{k,n} \leqslant W_{\mathcal{I}'_{k,n}}(A^+_{k,n}) = W_{1..m}(A^+_{k,n}) = S_{1..m}(U_{k,n}, A^+_{k,n}) - (A_{k,n} - U_{k,n}).$$

The response time bound theorem immediately follows, given Lemma 4.3:

Theorem 4.21
Let (A, C, D, Π) be a stable task process scheduled according to an EDF priority assignment and let $\mathcal{S} = \{\widehat{S}_k(x, q) \mid k = 1, \ldots, k;\ q \in Q\}$ be a family of MWAF for the task set. If

$$\widetilde{S}_k(x, a, q) \overset{\mathsf{def}}{=} \widehat{S}_{1..m}(a^+; q) \tag{4.39}$$

then a bound on the response times of a task τ_k is given by

$$R_{k,n} \leqslant \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a; q) - a,$$

where $\mathcal{A}_k(q) = \{a > 0 \mid \widetilde{S}_k(x, a, q) > x,\ \forall x \leqslant a\}$ and $\widetilde{E}_k(a, q) = \min\{x > 0 \mid \widetilde{S}_k(x, a, q) = x\}$.

Corollary 4.22 *Given a majorizing task process, the response time bounds under FIFO are:*

$$R_{k,n} \leqslant \max_{q \in Q} \max_{\tau_{i,j} \vdash \hat{A}_{i,j} < \widehat{V}_0^m(q)} \widehat{W}_{1..m}(\hat{A}^+_{i,j}). \tag{4.40}$$

The following can be noticed:

1. Computing the workload does not require a recursive computation because
$$\widehat{W}_{1..m}(\hat{A}^+_{i,j}) = \widehat{S}_{1..m}(0, \hat{A}^+_{i,j}) - \hat{A}_{i,j}.$$
2. As can be seen from (4.40), the response time bound is the same for all tasks of the set. The technical reason is the choice of $\mathcal{I}'_{k,n}$, see above. It implies that when different instances arrive at the same time then the all other instances are considered to arrive just before the instance under study - which implies a conservative bound. If time is supposed to be discrete, then a tighter bound can be obtained with
$$\mathcal{I}''_{k,n} = \{\tau_{i,j} \mid i \leqslant k,\ A_{i,j} \leqslant A_{k,n}; \quad i > k,\ A_{i,j} \leqslant A_{k,n} - 1\}$$
but the final response time bounds are at most one time unit shorter.
3. Although the response times bounds are (almost) the same for each task under FIFO, their average response time bounds could be different, depending on the stochastic assumption, one would make about the tasks.
4. Notice that Theorem 4.21 allows to compute response time bounds for any kind of task, for which a (family) of MWAF is available. i.e. for the task considered in Section 3. The tighter the MWAF's do characterize the behavior of the tasks, the tighter the resulting response time bounds.

4.5 Last in first out (LIFO)

4.5.1 Definition

Under the *last in first out* (LIFO) rule, a task receives at its activation time the highest priority among pending instances, such as to execute without interruption, if no other instance is activated before it finishes.

$$\Pi_{k,n}(t) = 1 \quad \Rightarrow \quad A_{i,j} > A_{k,n} \Rightarrow W_{i,j}(t) = 0. \tag{4.41}$$

This property only defines a class of policies, because it does not specify how instances with the same activation time are scheduled. We choose again an FPP-like rule to obtain a decidable assignment:

$$\Gamma_{k,n}(t) = (-A_{k,n}, k, -n). \tag{4.42}$$

Definition 4.23 *A scheduled task process is said to be scheduled according to the* last in first out *rule if it satisfies* (4.41).

4.5.2 Response time bounds

The priority assignment being time independent, the results of Section 4 apply. An instance $\tau_{i,j}$ is in the set $\mathcal{I}_{k,n}$ of instances with higher priorities of given task $\tau_{k,n}$ in the following cases:

1. $i < k$ and $A_{i,j} \geqslant A_{k,n}$
2. $i = k$ and $j > n$
3. $i > k$ and $A_{i,j} > A_{k,n}$.

Since it does not contain any instance activated strictly before $A_{k,n}$, $W_{i,j}(A_{k,n}) = 0$, implying

$$U_{k,n} = A_{k,n},$$

i.e. the interference period starts at $A_{k,n}$. As with FIFO, the set $\mathcal{I}_{k,n}$ contains instances that are determined by a strict inequality. For the same reasons as before, we change them into weak inequalities, using $\mathcal{I}'_{k,n} = \{\tau_{i,j} \mid A_{i,j} \geqslant A_{k,n}\}$, which satisfies

$$S_{\mathcal{I}_{k,n}}(A_{k,n}, A_{k,n}+x) \leqslant S_{\mathcal{I}'_{k,n}}(A_{k,n}, A_{k,n}+x) = S_{1..m}(A_{k,n}, A_{k,n}+x) \text{ for } x > A_{k,n},$$

because $\mathcal{I}_{k,n} \subset \mathcal{I}'_{k,n}$. We the help of Lemma 4.3 we deduce:

Theorem 4.24
Let (A, C, D, Π) be a stable task process scheduled according to an EDF priority assignment and let $\mathcal{S} = \{\widehat{S}_k(x, q) \mid k = 1, \dots, k;\ q \in Q\}$ be a family of MWAF for the task set. If

$$\widetilde{S}_k(x, a, q) \overset{\mathsf{def}}{=} \widehat{S}_{1..m}(x; q) \tag{4.43}$$

then a bound on the response times of a task τ_k is given by

$$R_{k,n} \leqslant \max_{q \in Q} \widetilde{E}_k(0; q),$$

where $\widetilde{E}_k(0, q) = \min\{x > 0 \mid \widetilde{S}_k(x, 0, q) = x\}$.

Proof: Lemma 4.3 implies

$$R_{k,n} \leqslant \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a; q) - a.$$

But $\widetilde{E}_k(a; q)$ is independent of a, because $\widetilde{S}_k(x, a, q)$ is independent of a, see (4.43). Thus

$$\max_{a \in \mathcal{A}_k(q)} \widetilde{E}_k(a; q) - a = \widetilde{E}_k(0; q).$$

■

Corollary 4.25 *Given a majorizing task process, the response time bounds under LIFO are:*

$$R_{k,n} \leqslant \max_{q \in Q} \max_{a \in \mathcal{A}_k(q)} \widehat{V}_0^m(q).$$

The following can be noticed:

1. Processor interference period length can be computed iteratively:
$$\widehat{V}_0^m(q) \min\{x > 0 \mid \widehat{S}_{1..m}(0, x; q) = x\}.$$
2. The response time bounds are the same for all tasks of the set for similar reasons as before. The response time bound is based on the fact that the instance under study is always considered to arrive just before the instances of all other tasks.

4.6 Complexity

It is known that in case of a critical instant, i.e. a unique majorizing task sequence q_0 in our model, the complexity of computing response time bounds is pseudo-polynomial in the number of tasks. In Section 4.6.1 we will see that for EDF the complexity can be improved compared to the response time bounds developed in [26].

If tasks have offset constraints, multi-frame execution time patterns, or sequencing constraints, then an exact majorizing task process consists of more than one trajectory because there is not one critical instant, but several among which none is uniformly worst than all others. For the mentioned examples, an exponential bound on $|Q|$ exists in the number of tasks and thus the complexity of computing response time in this case is exponential. Any method for reducing the complexity is therefore welcome. In Section 4.6.2 we investigate a Branch and Bound algorithm. Furthermore, since it depends on the actual task set how intractable the computation really is, it is of interest to be able to predict the effects of approximations on complexity and accuracy. We consider this question in Section 4.6.3.

4.6.1 Lower complexity for EDF

In [17], Hermant et al. have derived the numerical complexity of computing response time bounds. More precisely, upper and lower bounds on the complexity of determining feasibility by checking response time bounds are derived for

- a set of periodic tasks,
- with one critical instant (i.e. a unique majorizing task sequence)
- in terms of operations "simple" operations (addition, multiplication, ...)

for EDF and FPP. Recall that a direct feasibility test exists for EDF. The complexity is pseudo-polynomial in the number of tasks m:

	upper	lower
FPP, (resp. time bound)	$O(m^2)$	$O(m^3)$
EDF, (direct)	$O(m)$	$O(m^2)$
EDF (resp. time bound) [26]	$O(m^3)$	$O(m^3)$

To compare the complexity of the numerical response time computation under EDF of the algorithm given in [26] and the algorithm that corresponds to Corollary 4.15, we have generated sample sets and measured the complexity in the number of computed MWAF's values, i.e. the evaluation of $\widehat{S}_k(0,x,d;q)$. The result is shown in Figure 13. The functions m^2 and m^3 appear as straight lines with different slopes because of the log – log-scale.

For FPP, the complexity behaves more like m^2 than m^3, as for the sample task set given in [17]. For EDF, computed according to the timing analysis derived in [26], the complexity behaves like m^3, see the curve `EDF task by task`. It suggests that the complexity measured in terms of computed MWAF's values, behaves approximately as the complexity measured in terms of simple operations used in [17].

The curve `EDF global` corresponds to Corollary 4.15. The complexity behaves like m^2, because the lengths of the different first deadline interference periods on a majorizing trajectory only need to be computed once for all tasks, and not individually for each analyzed task, as for `EDF task by task` [26].

For Figure 13, 10 different sample sets, consisting in periodic tasks, have been generated for each set size $m = 5, 10, 15, \ldots, 400$, and the average of the measured complexities has been taken. Each sample set was generated as follows. A target load of $\rho_{1..m} = 0.7$ for the task set is partitioned into m target loads ρ_k for the tasks, using an uniform distribution on $]0, \rho_{1..m}]$. Since our implementation uses integer variables, the tasks cycle time T_k must be long enough so that the execution time $C_k = \rho_k \cdot T_k$

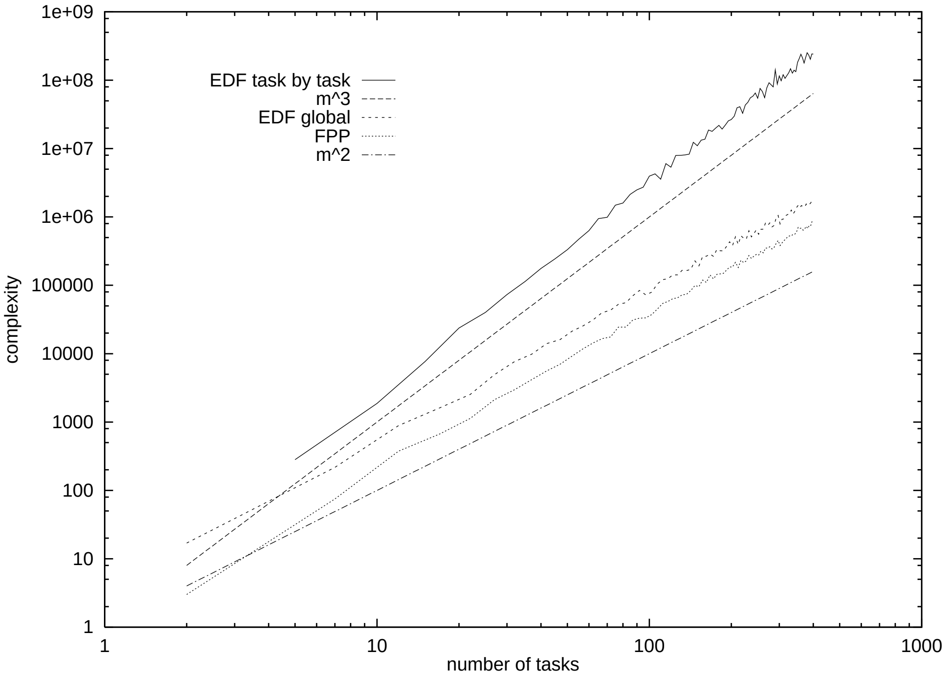


Figure 13: Complexity in terms of evaluations of MWAF's.

is larger than 1 and can be approximated by an integer in an acceptable way. We therefore generate the cycle time of a task using the uniform distribution on $[1/\rho_k, 100 \cdot m[$; the factor 100 is an arbitrary choice which gave satisfactory results. As a result we obtain a task set with a load close to the target $\rho_{1..m} = 0.7$. For a target load near to 1, overloaded sets might be obtained, which are then rejected and and for which a new set is generated.

For target loads different from $\rho_{1..m} = 0.7$, the graphs analogous to Figure 13 are very similar except for different additional constants, because the complexity depends on the load through a factor, see [17]. As foreseen by the formulas derived in [17], a lower load gives a lower slope.

4.6.2 Reducing complexity for FPP with a Branch & Bound algorithm

The final response time bound for a given task is in general the maximum over several majorizing response times which are each bounds for some subset of response times of the initial task process (A, C, D). There are several bounds for two reasons: first there could be several activations in the first interference period of a majorizing trajectory and second there might be several majorizing trajectories. If there is only one majorizing trajectory then the complexity is pseudo-polynomial (see the previous section) and the first reason is usually negligible for real-world task sets. But as we have seen, if dependencies between tasks are taken into account, then the number of majorizing trajectories can grow exponentially with the number of tasks. Examples show that for real-world task sets the computation takes a substantial amount of time.

The idea of the Branch & Bound algorithm is to use the current maximum of the already computed response time to check if the response times that must be computed on the trajectory q can be longer than the current maximum. The test is based on the property: $\widetilde{E}_k(a,q)-a > r \Rightarrow \widetilde{S}_k(a+r,a,q) > a+r$ which immediately follows from the definition of $\widetilde{E}_k(a,q)$. It means that $\widetilde{S}_k(a+r,a,q) > a+r$ is a necessary condition for the virtual response time $\widetilde{E}_k(a,q) - a$ to be longer than the current maximum r. The test is

$$\widetilde{S}_k(a+r,a,q) \leqslant a+r \qquad \Rightarrow \qquad \widetilde{E}_k(a,q) - a \leqslant r,$$

meaning that if the amount of arising work in the interval $[0, a+r[$ is smaller than $a+r$ then the resulting virtual response time is shorter than the current maximum and does not need to be computed, which saves the cost of computing iteratively $\widetilde{E}_k(a,q)$. Notice that the test has also a certain cost, which is equal to that of one step of the iterative calculus. Furthermore, the test is only necessary and not sufficient. It implies that if $\widetilde{E}_k(a,q) - a > r$ we can not draw any conclusion from it and have compute $\widetilde{E}_k(a,q)$ iteratively.

A similar test can be applied to check if the first interference period on a majorizing trajectory q is actually starting at zero. For FPP, it concerns the first level-k interference period with condition $\widehat{U}_{k,0}(q) = 0$. Recall the discussion about (4.25) on page 60. A necessary condition is $\widehat{S}_k(0, \widehat{A}_{k,0}; q) > \widehat{A}_{k,0}$. The test can thus be based on

$$\widehat{S}_{1..k-1}(0, \widehat{A}_{k,0}; q) \leqslant \widehat{A}_{k,0} \qquad \Rightarrow \qquad \widehat{U}_{k,0}(q) > 0.$$

Again, the test is not sufficient and induces a certain overhead. If the test succeeds then it saves the iterative computation of the interference period beginnings.

For EDF the test concerns deadline interference period with condition $\widehat{U}_0(d,q) = 0$. Recall the discussion about (4.34) on page 69. There is a necessary and sufficient test

$$\widehat{S}_{1..m}(0, 0^+, d; q) = 0 \Leftrightarrow \widehat{U}_0(q,d) = 0,$$

for which no iterative computation is needed and thus no simpler test exists concerning the beginning of a deadline-d interference period.

In order to find out if using the test reduces actually the total cost of the timing analysis or not we have performed numerical experiences on sample task sets consisting in periodic transactions. The method for generating the tasks is the same as in the previous section, except that the tasks periods are determined by the period of the transaction to which they belong, see equation (3.29) on page 39. The offset $\phi_{\mathcal{G},k}$ of a task τ_k belonging to a group $\mathcal{G}$ is chosen uniformly in the interval $[0, T_{\mathcal{G}}[$. The groups have all the same size. Their size, the number of groups and the load of the set are the parameters that will be studied. Figure 14 shows (for FPP) the ratio of the complexity of the computation with test against the complexity without the test. It can be seen that the complexity is reduced, but proportionally less for larger loads. The reason is that with increasing load the test fails more often. Furthermore, it can be noticed that the greater the complexity, the stronger the reduction. Recall that for g groups of r tasks, there are r^g offsets to be analyzed, see Section 3.2.3.1.

5^3	10^3	5^6	8^5
125	1000	15625	32768

The reduction does however not change the asymptotic behavior of the complexity, which remains exponential in the number of tasks.

Figure 15 shows the same graphs for the test on response times. It can be noticed that the reduction is stronger on the interval $[0, 0.85[$ but above $\rho_{1..m} = 0.85$ there is a breakdown of the performance resulting even in a greater complexity.

For EDF, the test on response times does not reduce the complexity. To understand this, notice that under FPP, the number of virtual response times $\widetilde{E}_k(a;q) - a$ to be computed in an interference period is equal to the the number of response times $\widehat{R}_{k,j}(q)$ on the majorizing trajectory q. But under EDF, the number of virtual response times $\widetilde{E}_k(a;q) - a$ is equal to the number of deadline-d interference periods $\widehat{V}_0^m(d)$, which is equal to the number of activations of all tasks in the first processor interference period. Applying the test to each of them costs almost as much as computing them, since a previous virtual response time can be used to initialize the iterative computation of the next. Furthermore, the test has a certain overhead and can fail.

We have also applied both tests at the same time, meaning that first the test on the beginning of the interference period is applied and if the trajectory q is to be studied then the test on response

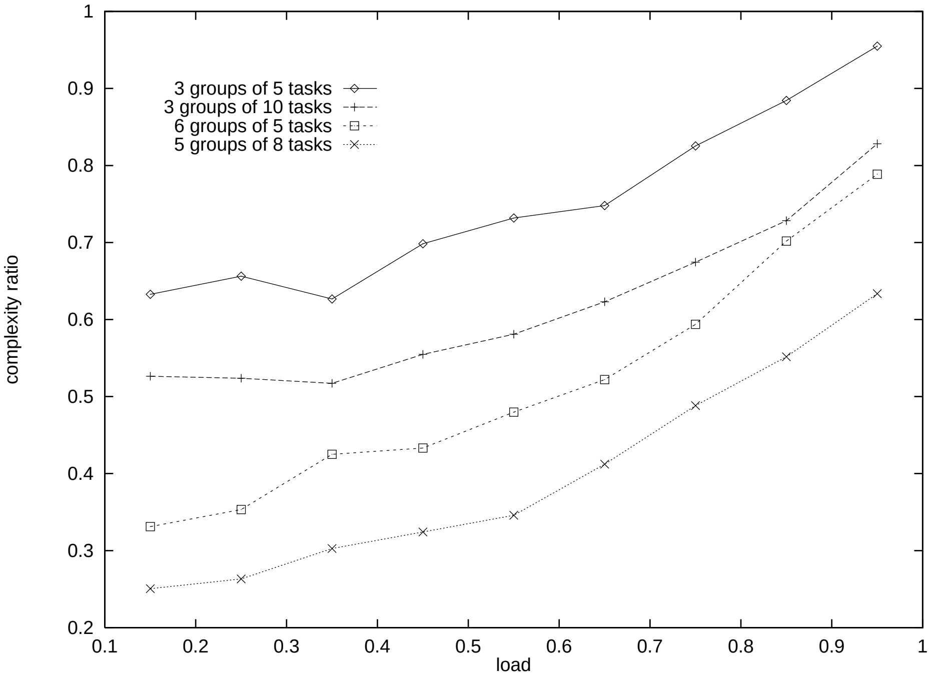


Figure 14: Branch & Bound algorithm applied to beginnings of level-k interference periods.

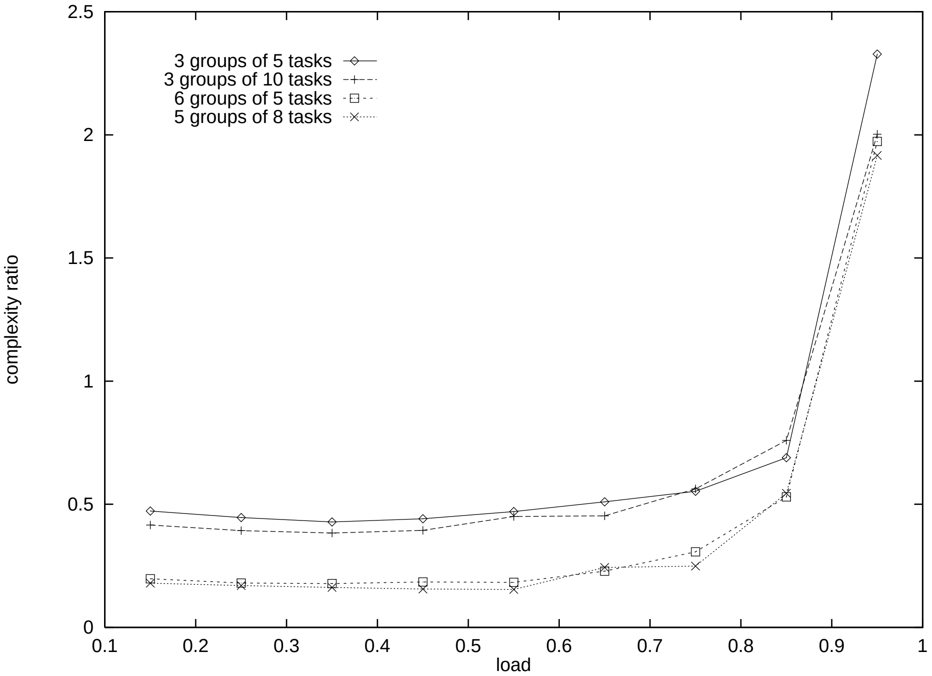


Figure 15: Branch & Bound algorithm applied to response times under FPP.

times is applied. But the resulting reduction of the complexity was always a little less than for the test on response times. One could try to understand in which cases the tests are efficient and in which there are not in order to make them collaborate and perhaps reduce further the complexity.

4.6.3 Lower complexity with conservative bounds

The simplest conservative bound consists in using unique MWAF's $\widehat{S}^*$ for tasks. For transactions, it means to base the bounds on the critical instant where all tasks are released at the same time, i.e to ignore offsets completely. In this section we will investigate more sophisticated conservative bounds which allow a tradeoff between complexity and tightness, based on the degree which offsets are taken into account.

The main idea is to choose some of the groups of depending tasks and to replace their family of MWAF's by a unique MWAF. We will apply the idea to transactions. Because the timing analysis of GMF-tasks is similar to the timing analysis of transactions, compare Proposition 3.8 and Proposition 3.9, the formulas will apply for both. The same idea can be applied to DAG-tasks, but because notations are far less handy, and we will not treat this case.

4.6.3.1 FPP Recall that under FPP all instances of higher priority tasks have an influence on the response time of the considered task τ_k, see (4.23) on page 59. These higher priority tasks can be part of any group:

$$\widehat{S}_{1..k-1}(x,q) = \sum_{h=1}^{g} \sum_{\tau_i \in \mathcal{G}_h, i<k} \widehat{S}_i(x,q).$$

Suppose $\tau_k \in \mathcal{G}_{h_0}$. For a group $\mathcal{G}_h$, $h \neq h_0$ we want to bound the MWAF's of the family by a unique function:

$$\begin{aligned}\sum_{\tau_i \in \mathcal{G}_h, i<k} \widehat{S}_i(x,q) &\leqslant \max_{q\in Q} \sum_{\tau_i \in \mathcal{G}_h, i<k} \widehat{S}_i(x,q)\\ &= \max_{(q_1,\dots,q_g,p_1,\dots,p_m)\in Q} \sum_{\tau_i \in \mathcal{G}_h, i<k} \widehat{S}_i^{(p_i)}(x - \Phi^*_{q_h,i}).\end{aligned}$$

Since the MWAF's only depends on q_h and $(p_1,\dots,p_m)$, the expression reduces to:

$$\leqslant \max_{q_h \in Q_h} \sum_{\tau_i \in \mathcal{G}_h, i<k} \max_{p_i=0..M_i-1} \widehat{S}_i^{(p_i)}(x - \Phi^*_{q_h,i}) = \max_{q_h \in Q_h} \sum_{\tau_i \in \mathcal{G}_h, i<k} \widehat{S}_i^{*}(x - \Phi^*_{q_h,i}).$$

Notice that this conservative bound is still an increasing left-continuous function, since it is the maximum of a finite number of finite sums of increasing left-continuous functions. Thus Proposition A.9 still applies for the numerical computation of the response time bounds.

The cost, in terms of evaluations of WAF's of the last expression, is more important than that of a "usual" MWAF, since several evaluations of WAF's are required to compute one of its values. On the other hand, the family of MWAF's is reduced to one element (the approximation), which reduces the size of Q. To evaluate the overall effect of the approximation on the complexity, we will measure the complexity again by the number of evaluations of WAF's.

We begin with the complexity of the exact bound. It consists in three factors. First, recall that execution ends are computed iteratively. For this factor, we suppose a bound χ_0 on the number of iteration steps performed on one trajectory q is known. Second, at each step, at most m WAF's are evaluated. Third, the same computations must be performed for each trajectory $q \in Q$. Given the number of g, we know that $|Q| \leqslant \prod_{l=1}^{g} m_l \cdot \prod_{i=1}^{m} M_i$, see (3.30). Thus we obtain as bound on the complexity in terms of evaluations of WAF's:

$$\chi_0 \cdot m \cdot \prod_{l=1}^{g} m_l \cdot \prod_{i=1}^{m} M_i.$$

Now, we turn to the conservative bound. Each evaluation of a $\widehat{S}_i^*$ requires M_i evaluations $\widehat{S}_k^{(p_i)}$ and the $\widehat{S}_i^*$ of at most all tasks in $\mathcal{G}_h$ are summed up. The result must be computed $m_h = |\mathcal{G}_h|$ times. The total cost is at most $m_h \cdot \sum_{\tau_i \in \mathcal{G}_h} M_i$ instead of $m_h = |\mathcal{G}_h|$ as for the exact bound. On the other hand, the number of trajectories $q \in Q$ is reduced, because for $\mathcal{G}_h$, the family of MWAF's is reduced to a unique MWAF. Recalling that $m = \sum_{l=1}^{g} m_l$, we obtain:

$$\chi_0 \cdot \left(m_h \cdot \sum_{\tau_i \in \mathcal{G}_h} M_i + \sum_{l \neq h} m_l \right) \cdot \prod_{l \neq h} m_l \cdot \prod_{\tau_i \in \mathcal{T}, \tau_i \notin \mathcal{G}_h} M_i.$$

If conservative bounds are used for all groups except the group $\mathcal{G}_{h_0}$ containing the considered task, then the complexity is reduced to

$$\chi_0 \cdot \left(m_{h_0} + \sum_{l \neq h_0} m_h \cdot \sum_{\tau_i \in \mathcal{G}_l} M_i \right) \cdot m_{h_0} \cdot \prod_{\tau_i \in \mathcal{G}_{h_0}} M_i.$$

4.6.3.2 EDF Under EDF all groups are treated the same way, there is no distinction of the group that contains the task under study. It is hence possible to apply the approximation to all groups.

$$\begin{aligned}
\widehat{S}_{1..m}(x, a + \overline{D}_k, q) \leqslant \widetilde{S}_k(x, a, q) &= \max_{q \in Q} \widehat{S}_{1..m}(x, a + \overline{D}_k, q) \\
&= \sum_{h=1}^{g} \max_{q \in Q} \sum_{\tau_i \in \mathcal{G}_h} \widehat{S}_i(x, a + \overline{D}_k, q) \\
&\leqslant \sum_{h=1}^{g} \max_{q_h \in Q_h} \sum_{\tau_i \in \mathcal{G}_h} \widehat{S}_i^*(x - \Phi^*_{q_h,i}, a + \overline{D}_k - \Phi^*_{q_h,i}, q).
\end{aligned}$$

As for FPP Proposition A.9 is valid for the numerical computation of the response time bounds. Notice that $\widetilde{S}_k$ can in general not be represented by the deadline based WAF's $\widehat{S}_{1..m}$ of a majorizing task process since the maximum of two deadline based WAF's is not necessarily a deadline based WAF, see Appendix A.3.

The complexity ranges from

$$\chi(g) = \chi_0 \cdot m \cdot \prod_{l=1}^{g} m_l \cdot \prod_{i=1}^{m} M_i \qquad \text{to} \qquad \chi(0) = \chi_0 \cdot \left(\sum_{l=1}^{g} m_l \cdot \sum_{\tau_i \in \mathcal{G}_l} M_i \right),$$

passing through

$$\chi(r) \stackrel{\text{def}}{=} \chi_0 \cdot \left(\sum_{l=1}^{r} m_l + \sum_{l=r+1}^{g} m_l \cdot \sum_{\tau_i \in \mathcal{G}_l} M_i \right) \cdot \prod_{l=1}^{r} m_l \cdot \prod_{\tau_i \in \mathcal{G}_l} M_i$$

if exact families of MWAF's are used for $\mathcal{G}_1, \ldots, \mathcal{G}_r$ and the approximation for $\mathcal{G}_{r+1}, \ldots, \mathcal{G}_g$.

It can be observed that with increasing number of approximations the complexity changes from a product of factors η_h, which are themselves factors, into a sum of terms κ_h, which are themselves sums. On could even introduce more refined steps by not taking the maximum over q_h and p_i in the same step, but in separate steps. Thus for a tradeoff between complexity and accuracy a wide range of steps exists. The question that remains is how the accuracy behaves as function of the complexity. We will answer this question by performing numerical experiments.

4.6.3.3 Numerical experiments As numerical example we consider a set of periodic transactions consisting of periodic tasks, i.e. $M_i = 1$. We have computed response time bounds of the lowest priority task τ_m in 100 different sample task sets, under FPP, for the different possible degrees of approximation. The sample task sets have been generated as in the previous section. Figure 16 shows the graph obtained under FPP in the case of 5 transactions consisting of 8 tasks each and Figure 17 shows the graph for 10 transaction consisting of 3 tasks each. The graphs are to be understood as follows:

complexity: ratio of the complexity of bounds with approximation over the complexity of the exact bounds, i.e. $\chi(r)/\chi(g)$.

inaccuracy: fraction by which the approximated bound is larger than the exact bound.

samples: fraction of the sample task sets for which the inaccuracy is smaller than at the considered point.

A curve that can be observed for a fixed value of complexity corresponds to a fixed number of groups with exact MWAF's. For a fixed complexity the sample-inaccuracy graph gives the distribution function of the inaccuracy; only the upper part from 75% to 100% is shown. As can be seen, for most of the samples the inaccuracy is zero. It can furthermore be noticed that the inaccuracy decreases with increasing complexity. We have also observed that choosing exact MWAF's first for transactions with greater ρ_k, makes the inaccuracy often decrease faster.

In Figure 16, a level-curve is drawn for 10% inaccuracy. The case of lowest complexity corresponds to 4 transactions with conservative MWAF and one transaction with exact MWAF's. It is the one proposed in [32] to render the computations tractable. The complexity in terms of evaluations of MWAF's is 37,190 and for less than 20% of the samples, the inaccuracy is more than 10%. The complexity of the exact bound is is 350 times larger. If the least accurate bound would take 1 second, the exact bound would take almost 6 minutes. The marked point correspond to an inaccuracy less than 10%, for less than 5% of the samples with a complexity of 909,900 evaluations of WAF's. It is the complexity in the case of 2 transactions with conservative bounds and 3 transactions with exact bounds. Thus the considered point corresponds to an only 25 times greater complexity and time to compute the bounds. It would take 25 seconds under the above assumptions. This suggest that with a reasonable of the cost the quality of the result can be quite improved. Whether the factor 25 is "too much" or "acceptable" depends of course on how much time the computation actually lasts. The important thing to notice is that the function $\chi(r)$ allows to predict the factor $\chi(r)/\chi(0)$ by which the time increases, if for r groups exact families of MWAF's are use instead of none. This possibility is useful for the design of timing analysis tools. An open question is, which properties of tasks do exactly induce the inaccuracy of the approximations.

Figures 18 and 19 show the combined effect of the the Branch and Bound algorithm described in the Section 4.6.2 and the use of conservative bounds. As before, the greater the complexity, the stronger its reduction and the breakdown above $\rho_{1..m} = 0.85$ remains.

It can be concluded, that it is not much worth computing exact bounds in the context of groups of tasks, since quite accurate bounds can be obtained at rather low cost. Furthermore, the choice of conservative bounds for all transactions as proposed in [32] can be improved while controlling the time taken by the timing analysis to produce the result.

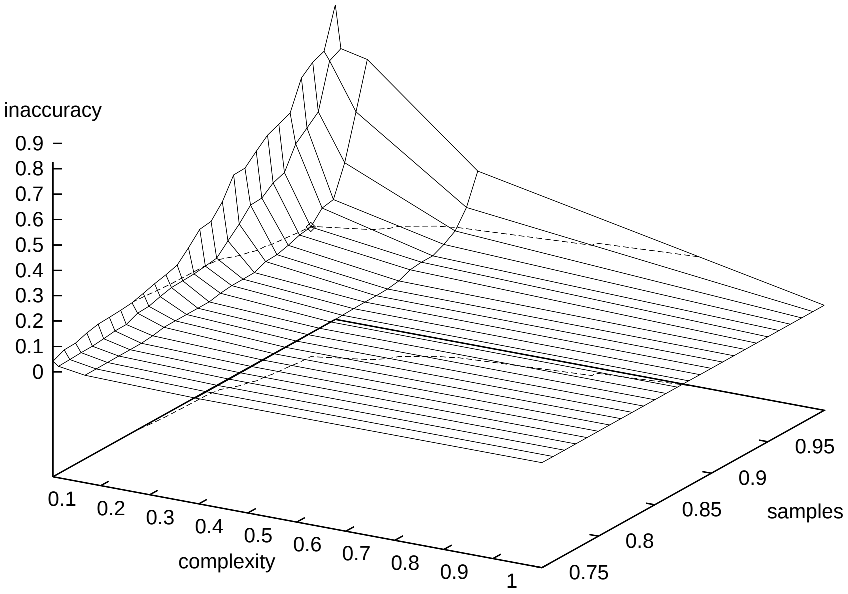


Figure 16: Conservative bounds for 5 transactions of 8 tasks.

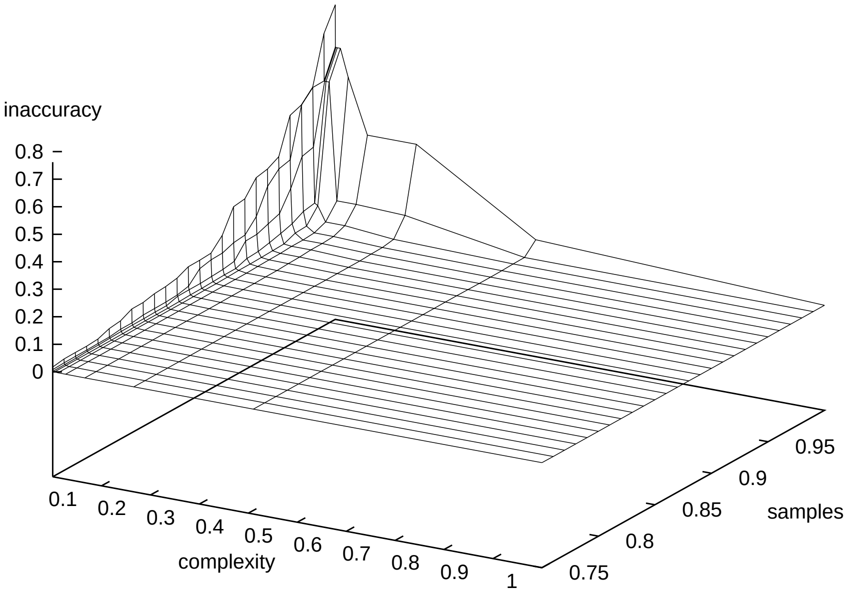


Figure 17: Conservative bounds for 10 transactions of 3 tasks.

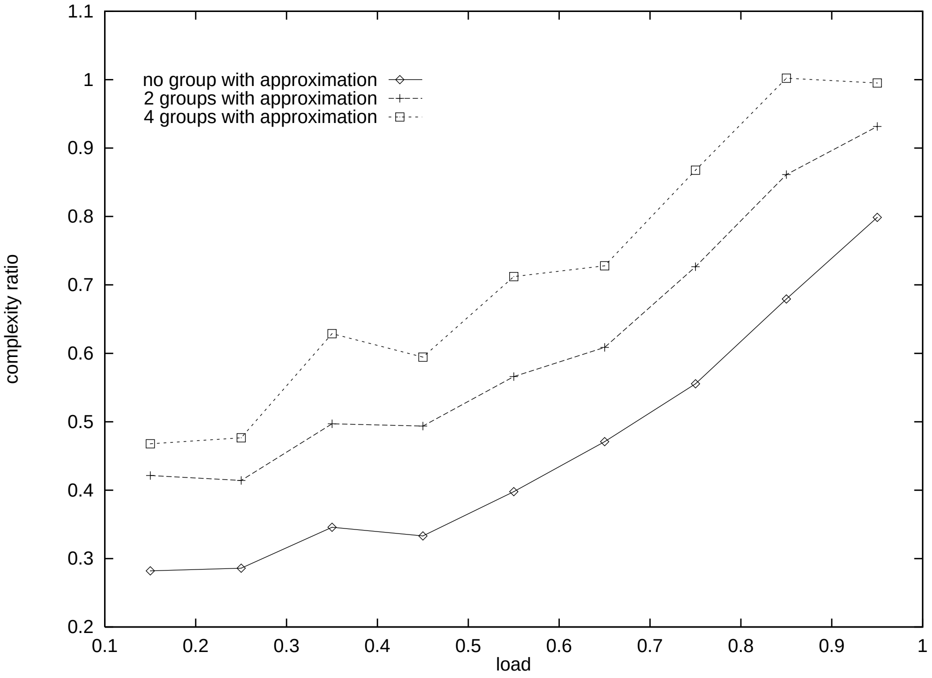


Figure 18: Branch & Bound algorithm for the beginning of interference periods and conservative bounds.

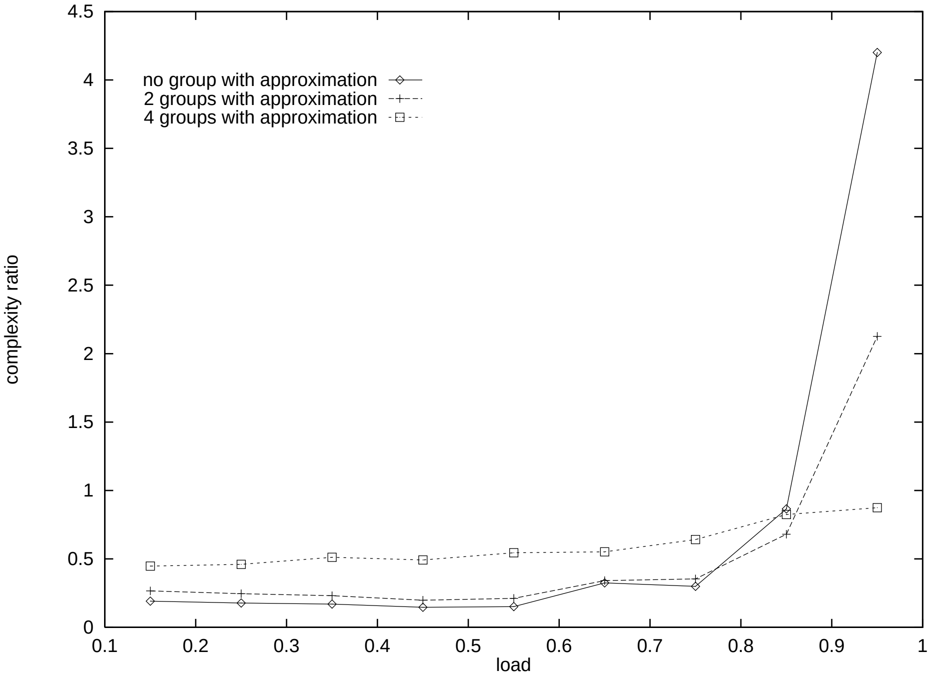


Figure 19: Branch & Bound algorithm for the response times and conservative bounds.

5 Conclusions and outlook

First let us recall some of the new results on timing analysis that have been obtained along the analysis of our scheduled task process model. For example, response time bounds can now be computed for transactions also under EDF. Furthermore, under EDF and FPP response time bounds can be computed for multiframe tasks, GMF-tasks, DAG-tasks, for which only feasibility tests had been derived in the literature. Notice in particular that with the use of a majorizing task process there is no need for the *frame separation* assumption, which was also used in the case of mode changes. It essentially ensures that any instance of a task ends before its next instance is released, which simplifies the timing analysis. The concepts of majorizing trajectory and interference periods which help to overcome this difficulty.

For the computation of response times under EDF, we have found an algorithm with lower pseudo-polynomial complexity, which behaves in practice like $O(m^2)$ instead of $O(m^3)$.

An extension of the current research could include the timing analysis of other time independent policies such as *shortest processing time first*, based on $C_{k,n}$ and not *shortest remaining processing time first* based on $W_{k,n}(t)$, or *least laxity first*, where the laxity is the difference between the relative deadline and the execution time of an instance. Further extensions, will include a similar analysis of non-preemptive scheduling policies and also the Round Robin scheduling policy [23].

We have also developed a tool with graphical interface that can be found at

`http://www.inria.fr/mistral/personnel/Jorn.Migge/rts.html`.

A Miscellaneous properties

A.1 Right continuity

A function $f : \mathbb{R} \longmapsto \mathbb{R} \ : \ x \longmapsto f(x)$ is said to be right continuous at x if

$$\forall\, \varepsilon > 0, \quad \exists\, \mu > 0 \text{ such that } x' \in \ [x, x+\mu[\Rightarrow f(x') \in]f(x) - \epsilon, f(x) + \epsilon[. \tag{A.1}$$

A.1.1 Work arrival functions

At some places, right-continuous WAF are needed. The corresponding majorizing WAF's can be derived as follows.

Proposition A.1 *The right-continuous version $\widehat{S}_k(x^+)$ of a majorizing WAF is a bound for right-continuous WAF's of the tasks:*

$$S_k(u, (u+x)^+) \leqslant \widehat{S}_k(x^+). \tag{A.2}$$

Proof: This can be proven by contradiction. If we had $\widehat{S}(x_0^+) < S_k(u_0, (u_0+x_0)^+)$ for some u_0 and x_0 then, because $\widehat{S}(x_0^+) = \lim_{\varepsilon \to 0^+} S_k(x_0 + \varepsilon)$, there would exist ε_0, such that

$$\widehat{S}_k(x_0 + \varepsilon) < S_k(u_0, (u_0 + x_0)^+) \qquad \forall \varepsilon < \varepsilon_0.$$

Since S_k is increasing in its second variable, $S_k(u_0, (u_0+x_0)^+) \leqslant S_k(u_0, u_0 + x_0 + \varepsilon)$. Thus (2.26) is violated. ■

A.1.2 The scheduling function

In this section we justify the right-continuity assumption about scheduling functions, introduced in Section 2.2.1 on page 9. First we state some related properties.

Proposition A.2 *Let f be a right continuous function and $x \in \mathbb{R}$, then:*

$$f(x) > 0 \Rightarrow \exists\, x' > x, \alpha > 0 \textit{ such that } f(u) > \alpha > 0, \ \forall\, u \in [x, x'[.$$

Proof: Choose the positive $\varepsilon = f(x)/2$. By (A.1), there exists $\mu > 0$ such that $f(u) > f(x)/2$ for $u \in [x, x+\mu[$. We can choose $x' = x + \mu$. ■

This proposition states that if a right-continuous function is strictly positive at a certain point then it is positive on some interval of non-zero length. This implies that if the integral of such a function is zero on an interval, then the function is zero at each point of the interval:

Corollary A.3 *Let f be a positive and right continuous function and let $x \in \mathbb{R}$, then:*

$$\int_a^b f(x)dx = 0 \qquad \Rightarrow \qquad f(x) = 0, \ \forall\, x \in [a, b[.$$

Proof: Notice first that the right-continuity implies that f is Lebesgue-measurable. If the required property where wrong, there would exist $x_0 \in [a, b[$ such that $f(x_0) > 0$. By Proposition A.2, f would be strictly positive on some interval $[x_0, x_0'[$. But then the integral of f would be positive, which is a contradiction. ■

Notice that $\Pi_{k,n}$ appears as a function under some integral in assumption (2.11) on page 10 and also in the workload definition (2.14). Actually a class of functions, equal except on a set of Lebesgue measure zero can satisfy this assumption and produce exactly the same workload $W_{k,n}$. The definition of the execution begin (2.20) would be "wrong", if at some isolated t_0 between $A_{k,n}$ and the first point after which $W_{k,n}$ decreases, i.e. the instance is really executed, $\Pi_{k,n}(t_0) = 1$. Hence, the definition should "better" be based on the workload:

$$B_{k,n} = \inf\{t \geqslant A_{k,n} \mid W_{k,n}(t^+) < C_{k,n}\}.$$

This definition is perhaps less intuitive than (2.20). However because we have assumed right-continuity for $\pi_{k,n}$, Corollary A.3 implies that the just described "pathological" situation can not occur and hence (2.20) is a definition consistent with the interpretation of execution begin that we want it to have.

A.2 Fixed point equations and iterative computation

A.2.1 Fixed point

Response times are often expressed as particular solutions of fixed point equations. These equations generally have more than one fixed point and such that in their neighborhood, the involved function is not necessarily a contraction. This partially explains why response times appear as "first fixed point after some point", see (A.4). In this section we give several simple properties of these fixed point equations which are repeatedly used when deriving response time equations.

Lemma A.4 *Let f be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ with $f(0) > 0$. Define*

$$\mathcal{X} \overset{\text{def}}{=} \{x > 0 \mid f(x) = x\} \qquad x^* \overset{\text{def}}{=} \inf \mathcal{X} \quad (\textit{if } \mathcal{X} \neq \emptyset).$$

We have the following properties:

1. *If there exists an $\hat{x} > 0$ such that $f(\hat{x}) \leqslant \hat{x}$ then there exists a $\tilde{x} \leqslant \hat{x}$ such that $f(\tilde{x}) = \tilde{x}$.*
2. *If $\mathcal{X} \neq \emptyset$ then $x^* = \min \mathcal{X}$,*
3. *and $f(x) > x$ for $x \in [0, x^*[$.*

Proof:

1. Let $\mathcal{L} \overset{\text{def}}{=} \{x > 0 | f(x) \leqslant x\}$ and $\tilde{x} \overset{\text{def}}{=} \inf \mathcal{L}$. Since $\hat{x} \in \mathcal{L}$, we have $\tilde{x} \leqslant \hat{x}$. Suppose we had $f(\tilde{x}) < \tilde{x}$. There would exist $\epsilon > 0$ such that $f(\tilde{x}) + \epsilon = \tilde{x}$. On the other hand $f(\tilde{x} - \epsilon/2) > \tilde{x} - \epsilon/2$, since $\tilde{x} - \epsilon/2 \notin \mathcal{L}$. Thus
$$f(\tilde{x}) = \tilde{x} - \epsilon < \tilde{x} - \epsilon/2 < f(\tilde{x} - \epsilon/2), \tag{A.3}$$
which is a contradiction with the assumption that f is increasing. Hence, $f(\tilde{x}) \geqslant \tilde{x}$.

2. Suppose $x^* \notin \mathcal{X}$. It is impossible to have $f(x^*) < x^*$ because then by 1. there would exist $\tilde{x} \leqslant x^*$ such that $f(\tilde{x}) = \tilde{x}$, implying $\tilde{x} \in \mathcal{X}$, which is contradiction with the definition of x^*. It remains the case $f(x^*) > x^*$. There exists then $\epsilon > 0$ such that $f(x^*) = x^* + \epsilon$. By definition of x^*,
$$\forall\, \mu > 0, \quad \exists\, x' \in \mathcal{X} \quad \vdash \quad x^* \leqslant x' \leqslant x^* + \mu.$$
We have $f(x') = x' \leqslant x^* + \mu = f(x^*) - \epsilon + \mu$. Choosing $\mu = \epsilon/2$ implies $f(x') < f(x^*) - \epsilon/2$ although $x^* < x'$, which is a contradiction with the assumption that f is increasing.

3. Suppose there would exist $\hat{x} \in [0, x^*[$, with $f(\hat{x}) \leqslant \hat{x}$. Then by point 1. there exists $\tilde{x} \leqslant \hat{x}$, with $f(\tilde{x}) = \tilde{x}$. It means $\tilde{x} \in \mathcal{X}$ with $\tilde{x} < x^*$, which is a contradiciton with the definition $x^* = \inf \mathcal{X}$.

■

The response time $R_{k,n} = x^*$ of a task typically satisfies an equation of the form

$$x^* = \min\{x > 0 \mid f(x_0 + x) = x\ \} \tag{A.4}$$

with $x_0 = A_{k,n}$ and f being the appropriate work arrival function. The related execution end $E_{k,n} = A_{k,n} + R_{k,n}$ is also solution of a fixed point equation. On can pass from one to the other, by changing the dummy variable $x = y - x_0$, which only affects the way of writing, but not the involved function:

$$y^* = x_0 + x^* = \min\{y > x_0 \mid f(y) + x_0 = y\ \}. \tag{A.5}$$

Lemma A.5 *Equation* (A.4) *is equivalent to* (A.5).

Proof: By point 3. of Lemma A.4 and the definition of x^*,

$$f(x_0 + x) > x, \ \ \forall\, x \in [0, x^*[\qquad \text{and} \qquad f(x_0 + x) = x \ \text{ for } x = x^*.$$

By substitution with $y = x_0 + x$ this gives

$$f(y) + x_0 > y, \ \ \forall\, y \in [x_0, x_0 + x^*[\qquad \text{and} \qquad f(y) = y \ \text{ for } y = x_0 + x^*.$$

■

Notice that the point $x_0 = A_{k,n}$ plays the role of a reference epoch after which something is considered. Furthermore the involved function f does not need to be defined before x_0. But if the execution end is expressed using the beginning of the appropriate interference period $U_{k,n}$, then the reference point changes to $y_0 = U_{k,n}$. The following proposition helps to handle this situation.

Proposition A.6 *Let $y_0 < x_0$ and g be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ which coincides with f up to an additive constant such that*

$$g(y) = f(y) + x_0 - y_0 \quad \forall\, y > x_0. \tag{A.6}$$

If

$$g(y) > y - y_0 \ \ \forall\, y \in [y_0, x_0[, \tag{A.7}$$

then for y^* defined in (A.5),

$$y^* = \min\{y > y_0 \mid g(y) + y_0 = y\ \}.$$

Proof: Simply because (A.7) implies that $g(y) + y_0 > y \ \ \forall\, y \in [y_0, x_0[\cup [x_0, x_0 + t[$. ■

If f can be bounded by another function $\hat{f}$ then the first fixed point either remains unchanged or is shifted towards the future $x^* \leqslant \hat{x}^*$. This is typically used when deriving response times bounds with $x_0 = U_{k,n}$ being the beginning of the interference period containing the instance under study and $\widehat{x}_0 = 0$ being the beginning of the first interference period of the majorizing task process.

Proposition A.7 *Let $\widehat{f}$ be an increasing function $\mathbb{R} \longmapsto \mathbb{R}$ with $\widehat{f}(\hat{x}_0) > \hat{x}_0$ for some $\hat{x}_0 \in \mathbb{R}$, and such that*

$$\widehat{x}^* = \min\{x > 0 \mid \widehat{f}(\hat{x}_0 + x) = x\ \} \tag{A.8}$$

exists. If

$$\forall\, x \in [0, \hat{x}^*[\quad f(x_0 + x) \leqslant \widehat{f}(\hat{x}_0 + x), \tag{A.9}$$

then x^* exists and $x^* \leqslant \widehat{x}^*$. If furthermore

$$\widehat{f}(x^*) = x^*, \tag{A.10}$$

then $\widehat{x}^* = x^*$.

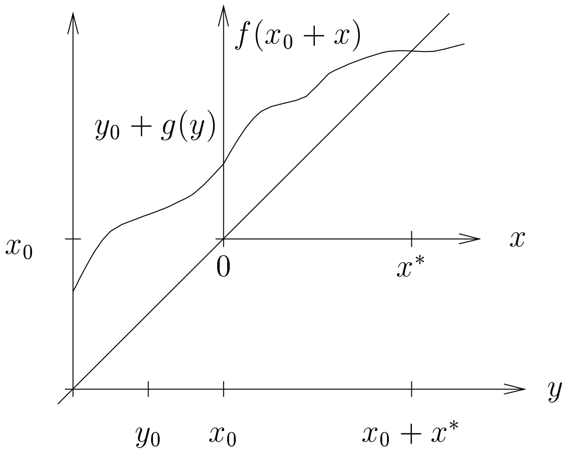


Figure 20: Illustration of Proposition A.6

Proof: Since $f(\hat{x}^*) \leqslant \hat{f}(\hat{x}^*) = \hat{x}^*$, point 1. of Lemma A.4 implies that x^* exist because $\mathcal{X} \neq \emptyset$. Furthermore $x^* \leqslant \widehat{x}^*$. ■

Corollary A.8
If $f(x_0+x) \leqslant \sigma + \rho \cdot x$, $\forall x$, *for some* $\sigma, \rho \in \mathbb{R}_+$ *with* $\rho < 1$ *then* $x^* = \min\{x > 0 \mid f(x) = x\}$ *exists.*

A.2.2 Iterations

The iterative computation of response times has been introduced by Joseph and Pandya [19]. Here we state this method in the framework of our model and give the assumption under which the computation works.

First remind the following definition. A function $f : \mathbb{R} \longmapsto \mathbb{R}$ is said to be *piecewise constant* if on any interval of finite length there is a finite partition into sub-intervals where the function is constant.

Proposition A.9 *Let* $f : \mathbb{R}_+ \longmapsto \mathbb{R}_+$ *be a left-continuous and increasing function with* $f(0) > 0$, *such that* $x^* = \min\{x > 0 \mid f(x) = x\}$ *exists. Let* $x_0 \in [0, x^*[$ *and* $x_n = f(x_{n-1})$, *for* $n \leqslant 1$.

(i) The sequence (x_n) *converges to* x^*.

(ii) If f *is piecewise constant, then the convergence takes a finite number of steps.*

Proof: First, we have

$$f(x) > x \text{ for } x \in [0, x^*[, \tag{A.11}$$

by point 3. of Lemma A.4. Second, we have

$$f(x) \leqslant x^*, \text{ for } x \in [0, x^*[. \tag{A.12}$$

Otherwise there would exist $x' \in [0, x^*[$ such that $f(x') > x^*$. Then, since f is increasing, $f(x^*) \geqslant f(x') > x^*$. But this is in contradiction with the definition of x^*.

Hence, on $[0, x^*[$, the sequence (x_n) is strictly increasing (A.11) and bounded above by x^* (A.12), so that it converges to a point in $]x_0, x^*]$.

Suppose we had $\lim_{n\to\infty} x_n = \tilde{x} < x^*$. Recall that $x_{n+1} \geqslant x_n$. Since f is left-continuous, we have

$$\lim_{n\to\infty} f(x_n) = f\Big(\lim_{n\to\infty} x_n\Big).$$

Since $x_{n+1} = f(x_n)$ it implies $\tilde{x} = f(\tilde{x})$, which is a contradiction with (A.11).

If f is furthermore piecewise constant, then it is an increasing step-function, with a finite number of steps $N = \operatorname{card}\{f(x) \mid x \in [0, x^*[\} < \infty$. Since $x_{n+1} > x_n$, this implies $\forall\, i > N$, $x_i = \lim_{n\to\infty} x_n$, i.e. x^* is reached in a finite number of steps. ■

Notice that assuming f to take integer values would have the same effect than assuming it to be piecewise constant. In view of the application of the model, discrete time and thus discrete WAF could be justified and furthermore it would not change the feasibility problem [7], but since such an assumption is not necessary we keep the model as general as possible.

A.3 The maximum of two WAF's

In this section we consider WAF as functions of one variable, by fixing the beginning of the interval to $t_1 = 0$:

$$s_k(0, x) = \sum_{n\in\mathbb{N}} c_{k,n} \cdot \mathbb{1}_{[a_{k,n} \leqslant x]}.$$

Notice that we do not use capital letters, because we are not considering task processes. As can be seen, the function is constant between two activations, where it is equal to the sum of all execution times until the last activation before x. The maximum of two majorizing WAF can be more easily computed, if we rewrite them as follows:

$$s(x) = \sum_{n\in\mathbb{N}} c_{k,0..n} \cdot \mathbb{1}_{[a_{k,n} < x \leqslant a_{k,n+1}]}.$$

Notice that in this expression the sum over n is used to enumerate different possibilities and not really as a sum.

In Table 11 we give an algorithm for computing the maximum $\hat{s}_k(x) = \max(s_k(x), s'_k(x))$, based on this alternative expression. To simplify the presentation, the algorithm is written for the WAF as defined in the model, which means in particular that both WAF have at least an activation after h. The three indexes, which are used to run through the activation have the following meaning at beginning of the loop:

n: index of the first activation of $s_k(x)$ not yet considered
n': index of the first activation of $s'_k(x)$ not yet considered
$\hat{n}$: index of the next activation of $\hat{s}_k(x)$ to be defined

Notice that because following activation times of the same WAF could be equal, e.g. $a_{k,n} = a_{k,n+1}$, we have to increase the indexes of $s_k(x)$ and $s_k(x)'$ until they point the last activation at the new time, see for example line 12. It ensures that $s_k(a^+_{k,n}) = c_{k,0..n}$.

Let us now consider deadline based WAF's as functions of two variables, by fixing the beginning of the interval to $t_1 = 0$:

$$s_k(x, d) = \sum_{n\in\mathbb{N}} c_{k,n} \cdot \mathbb{1}_{[a_{k,n} < x]} \cdot \mathbb{1}_{[d_{k,n} \leqslant d]}.$$

This function is increasing by steps in both variables. In Figure A.3 the locations of the steps are shown as projection on the domain of the function. Each activation is at the origin of a step.

We have $s_k(x, d) = s_k(a^+_{k,n}, d_{k,h})$ if $a_{k,n} < t \leqslant a_{k,n+1}$ and $d_{k,h} \leqslant d < d_{k,h+1}$. Thus, with

$$c_k(n, h) \overset{\mathsf{def}}{=} \sum_{j\in\mathbb{N}} c_{k,j} \cdot \mathbb{1}_{[a_{k,j} \leqslant a_{k,n}]} \cdot \mathbb{1}_{[d_{k,j} \leqslant d_{k,h}]} = \sum_{j=0}^{n} c_{k,j} \cdot \mathbb{1}_{[d_{k,j} \leqslant d]},$$

```
// $\hat{s}_k(x) := \max(s_k(x), s'_k(x))$ for $x \leqslant h$
proc $max(time\ h)$
  $\hat{n} := 0;\ n := 0;\ n' := 0;$
  do
     if $a_{k,n} < a'_{k,n'}$ then
        if $c_{k,0..n} > \hat{c}_{k,0..\hat{n}-1}$ then
           $\hat{a}_{k,\hat{n}} := a_{k,n};$
           $\hat{c}_{k,0..\hat{n}} := c_{k,0..n};$
           $\hat{n} := \hat{n} + 1;$
        fi
        $n := n + 1;$
        while $a_{k,n} = a_{k,n+1}$ do $n := n + 1$ od
     elsif $a_{k,n} > a'_{k,n'}$ then
          if $c'_{k,0..n'} > \hat{c}_{k,0..\hat{n}-1}$ then
             $\hat{a}_{k,\hat{n}} := a'_{k,n'};$
             $\hat{c}_{k,0..\hat{n}} := c'_{k,0..n'};$
             $\hat{n} := \hat{n} + 1;$
          fi
          $n' := n' + 1;$
          while $a'_{k,n'} = a'_{k,n'+1}$ do $n' := n' + 1$ od
     elsif $a_{k,n} = a'_{k,n'}$ then
          $\hat{a}_{k,\hat{n}} := a_{k,n};$
          if $c_{k,0..n} < c'_{k,0..n'}$ then
             $\hat{c}_{k,0..\hat{n}} := c'_{k,0..n'};$
          else
               $\hat{c}_{k,0..\hat{n}} := c_{k,0..n};$
          fi
          $\hat{n} := \hat{n} + 1;$
          $n := n + 1;$
          while $a_{k,n} = a_{k,n+1}$ do $n := n + 1$ od
          $n' := n' + 1;$
          while $a'_{k,n'} = a'_{k,n'+1}$ do $n' := n' + 1$ od
     fi
  until $\min(a_{k,n}, a'_{k,n'}) \geqslant h$
end
```

Table 11: Computing the maximum of two WAF's.

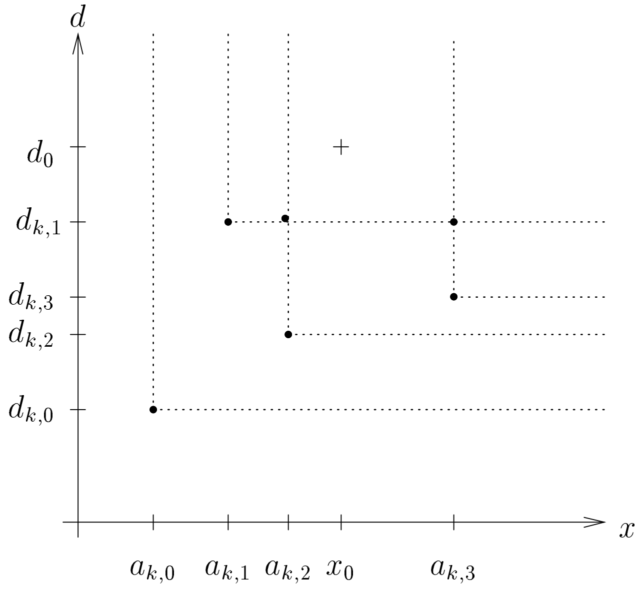


Figure 21: Domain of definition of a deadline based WAF

we can rewrite the WAF as

$$s_k(x,d) = \sum_{n \in \mathbb{N}} c_k(n,h) \cdot \mathbb{1}_{[a_{k,n} < x \leqslant a_{k,n+1}]} \cdot \mathbb{1}_{[d_{k,h} \leqslant d < d_{k,h+1}]}.$$

From the definition of $c_k(n,h)$ it can be seen that if $a_{k,h} > a_{k,n}$, then $c_{k,n}$ is not taken into account by $c_k(n,h)$. More precisely, there exists:

$$h' \ \vdash \ a_{k,h'} \leqslant a_{k,n}, \ \forall\, j \ \vdash a_{k,j} \leqslant a_{k,n}, \ \ d_{k,j} \leqslant d_{k,h'},$$

for which $c_k(n,h) = c_k(n,h')$. Similarly, if $d_{k,n} > a_{k,h}$, then $c_{k,n}$ is not taken into account by $c_k(n,h)$. More precisely, there exists:

$$n' \ \vdash \ d_{k,n'} \leqslant d_{k,h}, \ \forall\, j \ \vdash d_{k,j} \leqslant d_{k,h}, \ \ a_{k,j} \leqslant a_{k,n'},$$

for which $c_k(n,h) = c_k(n',h)$. Thus, in order to obtain the values of $s_k(x,d)$, only those $c_k(n,h)$ are needed, for which

$$d_{k,n} \leqslant d_{k,h} \qquad a_{k,h} < a_{k,n}.$$

These points a shown in Figure A.3 as thick dots. Based on this, a similar algorithm to the one given in Table 11 can be derived.

A.4 Complexity

Proposition A.10 *The function*

$$f_{m,g}(x_1, \ldots, x_{g-1}) = x_1 \ldots \cdot x_i \ldots \cdot x_{g-1} \cdot (m - x_1 \ldots - x_i \ldots - x_{g-1})$$

defined for $x_i > 0$ admits the following bound:

$$f_{m,g}(x_1, \ldots, x_{g-1}) \leqslant \left(\frac{m}{g}\right)^g \leqslant e^{\frac{m}{e}}.$$

Proof: We have

$$\begin{aligned}\frac{\partial f_{m,g}}{\partial x_i}(x_1,\dots,x_{g-1}) &= x_1 \dots \cdot x_{i-1}\cdot x_{i+1}\dots\cdot x_{g-1}\cdot(m-x_1\dots-x_i\dots-x_{g-1})\\ &\quad + x_1\cdot\dots\cdot x_i\cdot\dots\cdot x_{g-1}\cdot(-1)\\ &= \prod_{k\neq i} x_k\cdot(m-x_1\dots-2x_i\dots-x_{g-1}).\end{aligned}$$

Since $x_i > 0$,

$$\frac{\partial f_{m,g}}{\partial x_i}(x_1,\dots,x_{g-1}) = 0 \quad \forall\, i \qquad\Longleftrightarrow\qquad m-x_1\dots-2x_i\dots-x_{g-1}=0 \quad \forall\, i.$$

It is the intersection of $g-1$ hyper-plans. For $i \neq j$ we obtain

$$\begin{array}{rr} m-x_1\dots-2x_i\dots-x_{g-1} & =0\\ -\quad(m-x_1\dots-2x_j\dots-x_{g-1} & =0)\\ \hline x_j-x_i & =0.\end{array}$$

Thus, the gradient vanishes at a unique point with coordinates given by $x_i = \frac{m}{g}$. We prove now that it is a maximum. The coordinates of the Hessian Matrix are

$$H_{i,j} = \frac{\delta^2 f}{\delta x_i \delta x_j} = \begin{cases} -2\prod_{k\neq i} x_k & \text{if } i=j\\ \prod_{k\neq i,j} x_k\cdot(m-x_1\cdots-2x_i\cdots-2x_j\cdots-x_{g-1}) & \text{if } i\neq j.\end{cases}$$

Thus, at the point where the gradient is zero:

$$H_{i,j}\left(\frac{m}{g},\dots,\frac{m}{g}\right) = \begin{cases} -2(\frac{m}{g})^{g-2} & \text{if } i=j\\ -(\frac{m}{g})^{g-2} & \text{if } i\neq j.\end{cases}$$

For the matrix M given by $(\frac{m}{g})^{g-2}\cdot M = H$, we obtain

$$x^t M x = -2\sum_i x_i^2 - \sum_{i\neq j} x_i x_j = -\sum_{i,j} x_i x_j - \sum_i x_i^2 = -\left(\sum_i x_i\right)^2 - \sum_i x_i^2 < 0 \qquad \forall\, x \neq 0$$

and thus H is negative definite and the point is indeed a maximum. Let

$$f(m,g) = \left(\frac{m}{g}\right)^g = e^{g(\ln m - \ln g)}$$

with derivatives

$$\begin{aligned}\frac{\partial f}{\partial g}(m,g) &= (\ln m - \ln g - 1)\cdot e^{g(\ln m-\ln g)}\\ \frac{\partial^2 f}{\partial g^2}(m,g) &= ((\ln m-\ln g-1)^2 - 1/g)\cdot e^{g(\ln m - \ln g)}.\end{aligned}$$

The first is zero at $g = \frac{m}{e}$, where the second derivative is negative. Thus it a maximum and the second bound is derived. ■

List of Notations

Index

References

[1] N. Audsley, K.W. Tindell, and A. Burns. The end of the line for static cyclic scheduling. In *Proceedings of the 5th Euromicro Workshop on Real-Time Systems*, June 1993.

[2] N.C. Audsley. Optimal priority assignment and feasibility of static priority tasks with arbitrary deadlines. Technical Report YCS 164, University of York, November 1991.

[3] F. Baccelli and P. Brémaud. *Elements of Queuing Theory*, volume 26 of *Applications of Mathematics.* Springer-Verlag, 1994.

[4] S. Baruah. Feasibility analysis of recurring branching tasks. http://www.emba.uvm.edu/˜sanjoy/, 1998.

[5] S. Baruah. A general model for recurring real-time tasks. http://www.emba.uvm.edu/˜sanjoy/, 1998.

[6] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. http://www.emba.uvm.edu/˜sanjoy/, 1998.

[7] 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*, 2, 1990.

[8] Giogrio Buttazzo, Marco Spuri, and Fabrizio Sensini. Value vs. deadline scheduling in overload conditions. Technical Report TR ARTS Lab 95-02, Scuola Superiore S.Anna, 1996.

[9] C. Chaouiya, S. Lefebvre-Barbaroux, and A. Jean-Marie. Real-time scheduling of periodic tasks. In P. Chrétienne, E.G. Coffman, J.K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its Applications*, chapter 8, pages 167–191. John Wiley and Sons Ltd, 1995.

[10] R.L. Cruz. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):132–142, January 1991.

[11] R. Davis. Dual priority scheduling: A means of providing flexibility in hard real-time systems. Technical Report YCS230, Univerity of York, May 1994.

[12] M. Dertouzos. Control robotics: the procedural control of physical processors. In *IFIP CONGRESS*, pages 807–813, 1974.

[13] L. George, N. Rivierre, and M. Spuri. Preemptive and non-reemptive real-time uni-procesor scheduling. Technical Report 2966, INRIA, september 1996.

[14] R. Gerber, S. Hong, and M. Saksena. Guaranteeing real-time requirements with resource-based calibration of periodic processes. *IEEE Transactions on Software Engineering*, 21(7), July 1995.

[15] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignments for hard real-time offset free systems. Technical Report 299, Université Libre de Bruxelles, January 1995.

[16] M.G. Härbour, M.H. Klein, and Lehoczky. Timing analysis of fixed-priority scheduling of hard real-time systems. *IEEE Transactions on Software Engineering*, 20(1), January 1994.

[17] J.F. Hermant, L. Leboucher, and N. Rivierre. Real-time fixed and dynamic prority driven scheduling algorithms : theory and experience. Technical Report 3081, INRIA, December 96.

[18] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *12 th IEEE Real-Time Systems Symposium*, pages 129–139. IEEE, 1991.

[19] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[20] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

[21] J.Y.T Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2(4):237–250, December 1982.

[22] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of ACM*, 20(1):40–61, February 73.

[23] J.M. Migge. *Scheduling of recurrent tasks on one processor: A trajectory based Model.* PhD thesis, Université Nice Sophia-Antipolis, January 1999.

[24] A. Mok and D. Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, october 1997.

[25] P. Pushner. Worst-case execution-time analysis at low cost. *Control Eng. Practice*, 6:129–135, 1998.

[26] M. Spuri. Analysis of deadline scheduled real-time systems. Technical Report 2772, INRIA, January 1996.

[27] M. Spuri and J. Stankovic. How to integrate precedence constraints and shared resources in real-time scheduling. *IEEE Transactions on Computers*, 1994.

[28] J. Stankovic, M. Spuri, K. Ramamritham, and C. Buttazzo. *Deadline Scheduling for Real-Time Systems.* Kluwer Academic Publishers, 1998.

[29] Jun Sun, Mark K. Gardner, and Jane W. S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Transactions on Software Engineering*, 23(10):603–615, October 1997.

[30] K. Tindell, A Burns, and A. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proceedings IEEE Real-Time Systems Symposium*, December 1992.

[31] K. Tindell, A. Burns, and A.J. Wellings. An extendible approach for analysing fixed priority hard real time sytems. *Real-Time Systems*, 6(2), 1994.

[32] K.W. Tindell. *Fixed Priority Scheduling of Hard Real-Time Systems.* PhD thesis, University of York, December 1993.

[33] K.W. Tindell. Adding time-offsets to schedulability analysis. Technical Report YCS221, Department of Computer Science, University of York, 94.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399