Technical Report

Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks

Vincent Nelis

Bjorn Andersson

José Marinho

Stefan M. Petters

HURRAY-TR-110703

Version:

Date: 07-13-2011

Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks

Vincent Nelis, Bjorn Andersson, José Marinho, Stefan M. Petters

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

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

Abstract

Embedded real-time systems often have to support the embedding system in very different and changing application scenarios. An aircraft taxiing, taking off and in cruise flight is one example. The different application scenarios are reflected in the software structure with a changing task set and thus different operational modes. At the same time there is a strong push for integrating previously isolated functionalities in single-chip multicore processors. On such multicores the behavior of the system during a mode change, when the systems transitions from one mode to another, is complex but crucial to get right. In the past we have investigated mode change in multiprocessor systems where a mode change requires a complete change of task set. Now, we present the first analysis which considers mode changes in multicore systems, which use global EDF to schedule a set of mode independent (MI) and mode specific (MS) tasks. In such systems, only the set of MS tasks has to be replaced during mode changes, without jeopardizing the schedulability of the MI tasks. Of prime concern is that the mode change is safe and efficient: i.e. the mode change needs to be performed in a predefined time window and no deadlines may be missed as a function of the mode change.

Global-EDF Scheduling of Multimode Real-Time Systems Considering Mode Independent Tasks

Vincent Nélis*, Björn Andersson†*, José Marinho* and Stefan M. Petters*

*CISTER-ISEP Research Centre, Polytechnic Institute of Porto, Portugal
†Software Engineering Institute, Carnegie Mellon University, USA
{nelis, baa, jmsm, smp}@isep.ipp.pt, baandersson@sei.cmu.edu

***Abstract*—Embedded real-time systems often have to support the embedding system in very different and changing application scenarios. An aircraft taxiing, taking off and in cruise flight is one example. The different application scenarios are reflected in the software structure with a changing task set and thus different operational modes. At the same time there is a strong push for integrating previously isolated functionalities in single-chip multicore processors. On such multicores the behavior of the system during a mode change, when the systems transitions from one mode to another, is complex but crucial to get right. In the past we have investigated mode change in multiprocessor systems where a mode change requires a complete change of task set. Now, we present the first analysis which considers mode changes in multicore systems, which use global EDF to schedule a set of mode independent (MI) and mode specific (MS) tasks. In such systems, only the set of MS tasks has to be replaced during mode changes, without jeopardizing the schedulability of the MI tasks. Of prime concern is that the mode change is *safe* and *efficient*: i.e. the mode change needs to be performed in a predefined time window and no deadlines may be missed as a function of the mode change.**

***Keywords*-Mode-change, multiprocessors, multicores, real-time systems, real-time scheduling.**

I. Introduction

In many real-time computer systems, the task set that executes must be changed throughout the operation and hence the process of replacing one task set for another becomes important. Sometimes this need arises because of changes in the physical environment, for example, in an avionics system, the tasks that must execute during taxing are different from the ones that must execute during flying.

Systems in which tasks may be replaced by other tasks are typically organized as a set of *modes* where each mode comprises a set of tasks and the entire system is called a *multimode* system. During the execution of such multimode systems, switching from the current mode (called the *old-mode*) to any other mode (called the *new-mode*) requires to substitute the currently executing task set with the set of tasks of the new-mode. This substitution introduces a *transient phase*, where tasks of both the old- and new-mode may be scheduled *simultaneously*, thereby leading to a possible overload that can compromise the system schedulability, even if both the old- and new-mode have been asserted schedulable by the schedulability analysis. Therefore, researchers have proposed protocols (so-called *mode-change protocols*) which govern when a task in a new mode can be enabled. Researchers have also proposed methods which can prove, for a given scheduling algorithm in each mode and for a given mode-change protocol, that all deadlines are met.

The scheduling problem during a transition between two modes has multiple aspects, depending on the behavior and requirements of the old- and new-mode tasks when a mode change is initiated. The research literature offers several results on mode-change protocols and proof techniques for a computer system with a single processor. Initial results have also been produced for multiprocessors. There are tasks however (called *mode-independent tasks* in the literature) which should execute in *every* mode and such that their activation pattern must not be jeopardized during the transition between those modes[1]. Unfortunately, the current research literature offers *no schedulability analysis for multiprocessor scheduling of multimode systems comprising mode-independent tasks*. Therefore, in this paper, we present a mode-change protocol and corresponding analysis for multiprocessor scheduling of multimode systems with mode-independent tasks. Our results apply to constrained-deadline sporadic tasks using global-EDF [1] (gEDF or simply EDF).

The remainder of this paper is organized as follows. Section II gives related work and Section III presents the system model we use. In Section IV, we make two observations which motivate our choice of mode-change protocol and the overall approach of our analysis. Section V states the mode-change protocol and Sections VI and VII analyze it. Finally, Section VIII gives our conclusions.

II. Related work and contribution

Transition scheduling protocols are often classified with respect to the way they schedule the old- and new-mode tasks during the transitions. In the literature (see for instance [2] which considers uniprocessor systems), the following definitions are used.

[1]In practice, mode-independent tasks typically allow to model daemon functionalities and low-level control loops.

Definition 1 (Synchronous/Asynchronous protocol): A mode-change protocol is said to be synchronous if it schedules new-mode tasks only when all the old-mode tasks have completed. Otherwise, it is said to be asynchronous.

Definition 2 (Protocol with/without periodicity): A mode-change protocol is said to be "with periodicity" if and only if it is able to deal with mode-independent tasks. Otherwise, it is said to be "without periodicity".

Numerous mode-change protocols have been proposed for *uni*processor platforms (a survey about this concern is presented in [2]). In such environments, existing work [2]–[4] has shown that even if two modes of the application have been proven feasible, the transition between the two modes can cause violation of timing constraints, hence needing explicit analyses. Such analyses have been proposed, considering the popular Deadline Monotonic Algorithm [5]. An analysis of sporadic tasks scheduled by EDF is known as well [6]. In [7], authors proposed an analysis which considers Fixed-Task-Priority scheduling (FTP), Earliest-Deadline-First [1] scheduling and arbitrary task activation pattern. Furthermore, for applications that were initially proven not schedulable during the transition phases, they derived the required offsets for delaying the initialization of transition between two modes in order to make the application schedulable.

Among the uniprocessor *synchronous* protocols, one can cite the *Minimum Single Offset* (MSO) protocol [2], the *Idle Time* protocol [8] and the *Maximum-Period Offset* protocol [9]. Among the uniprocessor *asynchronous* protocols, one can cite the protocol *without periodicity* proposed in [10], the protocol *with periodicity* proposed by Sha et al. [11] for Fixed-Task-Priority schedulers (and extended to EDF in [6]), and the particular protocol introduced in [5] that allows tasks to modify their parameters (period, execution time, etc.) during the mode changes.

It is worth noticing that the problem of scheduling multimode applications upon *multiprocessor* platforms is much more complex than upon uniprocessor platforms, especially due to the presence of scheduling anomalies, and it is now well known that real-time multiprocessor scheduling problems are typically not solved by applying straightforward extensions of techniques used for solving similar uniprocessor problems. Assuming *identical* multiprocessor platforms, the authors of [12] proposed two protocols *without periodicity* for managing mode transitions, namely, SM-MSO (which is *synchronous*) and AM-MSO (which is *asynchronous*). Then, these two protocols were extended to uniform platforms in [13]. These extensions however, are still not considering mode-independent tasks. The interested reader can consult [14] for a complete description of these two protocols (and their associated schedulability analyses) for both identical and uniform platforms, assuming in turn Fixed-Job-Priority and Fixed-Task-Priority schedulers. Finally, this paper presents some similarities with [15], in which the authors address the problem of scheduling tasks that can be "reweighted" (i.e., their workload can be modified) at runtime. However, unlike our work, [15] focuses on soft real-time systems (i.e., deadline misses are tolerated) and implicit-deadline tasks (the deadlines are equal to the periods).

Contribution of this work: This study introduces a new multiprocessor synchronous mode-change protocol with periodicity: SM-MDO. It assumes *identical* multiprocessor platforms and the proposed schedulability analysis assumes that the modes are scheduled using Earliest-Deadline-First algorithm. Furthermore, we show through some examples that multiprocessor mode-change protocols with periodicity cannot be straightforwardly extended from protocols without periodicity. Indeed, while synchronous multiprocessor protocols without periodicity require only that a schedulability test is performed (i) on the tasks of each mode and (ii) for each mode transition, the problem of scheduling multimode systems comprising mode-independent tasks also requires that an additional schedulability test is performed on the whole system.

III. Models of computation and specifications

A. Application specifications

We define a multimode real-time application as a set τ of x operating modes denoted by $M^1, M^2, \ldots, M^x$ where each mode M^i has to execute its associated task set $\widetilde{\tau}^i$, i.e., $\tau \stackrel{\text{def}}{=} \{\widetilde{\tau}^1, \widetilde{\tau}^2, \ldots, \widetilde{\tau}^x\}$. The task set $\widetilde{\tau}^i$ of each mode M^i is composed of two disjoint subsets of tasks τ^i and τ^{mit}, i.e., $\forall k \in [1, x]$: $\widetilde{\tau}^i = \tau^i \cup \tau^{\text{mit}}$ and $\tau^i \cap \tau^{\text{mit}} = \phi$.

- $\tau^i \stackrel{\text{def}}{=} \{\tau^i_1, \tau^i_2, \ldots, \tau^i_{n_i}\}$ contains n_i tasks that belong exclusively to mode M^i, i.e., $\forall j \neq i$: $\tau^i \cap \tau^j = \phi$. Hereafter, these tasks will be referred to as the Mode-Specific (MS) tasks of mode M^i.
- $\tau^{\text{mit}} \stackrel{\text{def}}{=} \{\tau^{\text{mit}}_1, \tau^{\text{mit}}_2, \ldots, \tau^{\text{mit}}_y\}$ is the set of *Mode-Independent* (MI) tasks. On the contrary to the MS tasks, all the task of τ^{mit} belong to every mode.

At run-time, the application is either running in one and only one mode (say M^i), i.e., it is executing only the task set $\widetilde{\tau}^i$ associated to that mode (which includes the MI tasks), or it is switching from one mode to another one. We will explain in Section V how the mode transitions are managed in this paper.

In every mode M^i, each MS task $\tau^i_k \in \tau^i$ is modeled by a *sporadic* and *constrained-deadline* task which is characterized by three parameters $\langle C^i_k, D^i_k, T^i_k \rangle$—a worst-case execution time C^i_k, a minimum inter-arrival time T^i_k and a relative deadline $D^i_k \leq T^i_k$. The MI tasks are modeled in the same way, i.e., $\forall k \in [1, y]$: τ^{mit}_k is characterized by the 3-tuple $\langle C^{\text{mit}}_k, D^{\text{mit}}_k, T^{\text{mit}}_k \rangle$ according to the interpretation given above.

B. Definitions and notations

Definition 3 (Active job): At run-time, we say that a job is *active* at time-instant t if it has been released at or before time t and it is not completed at time t.

Since we assume $D_i^k \leq T_i^k \; \forall i, k$, there cannot be two jobs of a same task τ_i^k active at a same time in any feasible[2] schedule. All the tasks are assumed to be independent, i.e., there is no communication, no precedence constraint and no shared resource (except the processors) between them. In [12], we introduced the following concept of enabled/disabled tasks.

Definition 4 (Enabled/disabled tasks [12]): At run-time, any task τ_k^i of the application can release jobs if and only if τ_k^i is enabled. Symmetrically, a disabled task cannot release jobs.

As such, disabling a task τ_k^i prevents future job releases from τ_k^i. Notice that the MI tasks are enabled at system boot and are never disabled afterwards. When all the MS tasks of a mode (say M^i) are *enabled* and all the MS tasks of all the other modes are *disabled*, the application is said to be running in mode M^i (since only the tasks of mode M^i can release jobs).

The following notations will be used. For any task τ_k, we define the density λ_k of τ_k as $\lambda_k \stackrel{\text{def}}{=} C_k/D_k$. That is, the density of any MS task τ_k^i of any mode M^i will be denoted by λ_k^i, and the density of any MI task τ_k^{mit} will be denoted as λ_k^{mit}. Also, for any task τ_k, we will use in the proof of Section VII the popular notion of "Demand Bound Function" DBF, as well as the "*Forced-Forward Demand Bound Function*" FF-DBF defined in [16] (and for which an intuitive interpretation is given below). These two functions as defined as follows.

$$\text{DBF}(\tau_k, t) \stackrel{\text{def}}{=} \left(\left\lfloor \frac{t - D_k}{T_k} \right\rfloor + 1 \right) \times C_k \tag{1}$$

$$\text{FF-DBF}(\tau_k, t, \sigma) \stackrel{\text{def}}{=}$$

$$q_k \cdot C_k + \begin{cases} C_k & \text{if } r_k \geq D_k \\ C_k - (D_k - r_k)\sigma & \text{if } D_k > r_k \geq D_k - \frac{C_k}{\sigma} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $q_k \stackrel{\text{def}}{=} \lfloor \frac{t}{T_k} \rfloor$ and $r_k \stackrel{\text{def}}{=} t \bmod T_k$.

One can summarize the intuition behind the concept of FF-DBF as follows: the *minimum demand* of any job over a time interval of length t is the minimal amount of execution that the job must execute within this interval if it is to meet its deadline. Then, the *maxmin demand* of a sporadic task τ_k over a time interval of length t is the largest minimum demand of any collection of jobs that could be legally generated by τ_k within this interval. This largest minimum demand is reached by the scenario in which τ_k releases jobs *exactly* T_k time units apart and each job consumes its worst-case execution time C_k. Finally, the Forced-Forward Demand Bound Function denotes the maxmin demand of a task over a time interval of length t, when executing outside the interval occurs on a processor of speed σ (see [16] for more details about FF-DBF). Figure 1 gives a visualization of $\text{FF-DBF}(\tau_k, t, \sigma)$ for any task τ_k.

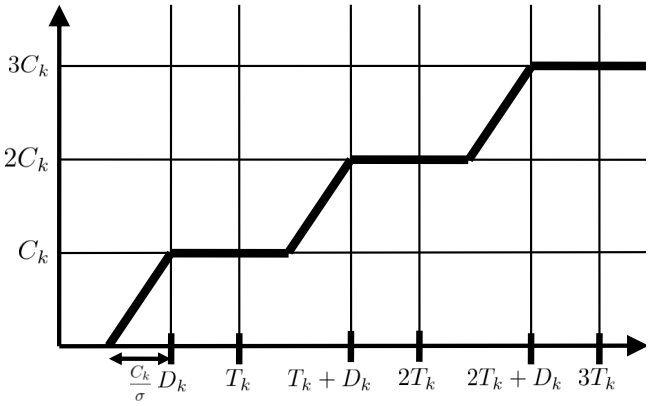

Figure 1. Illustration of $\text{FF-DBF}(\tau_k, t, \sigma)$ (from [16]).

Then, for any set τ^i of tasks (this also holds for τ^{mit}), we define

$$\lambda_{\max}(\tau^i) \stackrel{\text{def}}{=} \max_{\tau_k^i \in \tau^i} \{\lambda_k^i\} \tag{3}$$

$$\lambda_{\text{sum}}(\tau^i) \stackrel{\text{def}}{=} \sum_{\tau_k^i \in \tau^i} \lambda_k^i \tag{4}$$

$$\text{DBF}(\tau^i, t) \stackrel{\text{def}}{=} \sum_{\tau_k^i \in \tau^i} \text{DBF}(\tau_k^i, t) \tag{5}$$

$$\text{FF-DBF}(\tau^i, t, \sigma) \stackrel{\text{def}}{=} \sum_{\tau_k^i \in \tau^i} \text{FF-DBF}(\tau_k^i, t, \sigma) \tag{6}$$

$$\text{LOAD}(\tau^i) \stackrel{\text{def}}{=} \max_{t>0} \left\{ \frac{\text{DBF}(\tau^i, t)}{t} \right\} \tag{7}$$

$$\text{FF-LOAD}(\tau^i, \sigma) \stackrel{\text{def}}{=} \max_{t>0} \left\{ \frac{\text{FF-DBF}(\tau^i, t, \sigma)}{t} \right\} \tag{8}$$

The $\text{DBF}, \text{FF-DBF}, \text{LOAD}$ and FF-LOAD functions can be computed *exactly* (by the methods proposed in [17] for instance) or *approximately* (by those proposed in [18]) to any arbitrary degree of accuracy in pseudo-polynomial time or polynomial time, respectively.

C. Platform and scheduler specifications

We assume identical multiprocessor platform model. In such platforms, all the CPUs have the same computational capabilities, with the interpretation that in any interval of time two CPUs execute the same amount of work (assuming that none of them is idling). Also, we assume that the system is scheduled by Global-EDF. This scheduler assigns priority

[2]A schedule is said to be feasible iff no deadline is missed in that schedule.

to jobs at run-time according to their absolute deadlines: the earlier the deadline, the higher the priority (ties are broken arbitrarily). Then, at any time, EDF assigns the m highest priority jobs to the m CPUs. This algorithm is work-conserving, fully preemptive and allows job migrations, according to the usual definitions. Furthermore, EDF has been proven sustainable [19] according to the following definition.

Definition 5 (Sustainability): A scheduling algorithm is sustainable if the schedulability of a task set is not jeopardized by decreasing job execution times or by increasing task periods.

D. Mode transition specifications

While the application is running in any mode M^i, a mode change can be initiated by any task of $\widetilde{\tau}^i$ or by the system itself, whenever it detects a change in the environment or in its internal state for instance. This is performed by invoking a $\mathrm{mcr}(j)$ (i.e., a Mode Change Request), where M^j is the destination mode. At any time t, we denote by $t_{\mathrm{mcr}(j)} \leq t$ the invoking time of the *last* $\mathrm{mcr}(j)$. From the time at which a mode change is requested to the time at which the transition phase ends, M^i and M^j are referred to as the old- and new-mode, respectively.

At run-time, mode transitions are managed as follows. Suppose that the application is running in mode M^i and the system (or any task of $\widetilde{\tau}^i$) comes to request a mode change to mode M^j, with $j \neq i$. At time $t_{\mathrm{mcr}(j)}$, the system enters the transition phase and *immediately* disables all the MS tasks of the old-mode, i.e., the tasks of τ^i, thus preventing them from releasing new jobs. That is, from this time and until the transition phase ends, only the MI tasks can still release jobs. At time $t_{\mathrm{mcr}(j)}$, the active jobs issued from the disabled tasks of τ^i, henceforth called the *rem-jobs* (for "*rem*aining *jobs*"), may have two distinct behaviors: either they can be *immediately* aborted upon the $\mathrm{mcr}(j)$, or *they have to complete execution*[3]. From a schedulability point of view, aborting some (or all) rem-jobs upon a mode change request does not jeopardize the system schedulability during the transition phase[4]. Consequently, we assume the worst-case scenario for every mode transition, i.e., the scenario in which *every MS task of the old-mode has to complete its last released job (if any) during every mode transition*.

The fact that the rem-jobs have to complete their execution upon the $\mathrm{mcr}(j)$ brings the following problem: even if both task sets $\widetilde{\tau}^i$ and $\widetilde{\tau}^j$ (from the old- and new-mode, respectively) have been separately asserted to be EDF-schedulable upon the m CPUs at system design-time, enabling all the MS tasks of the new-mode *immediately* upon the $\mathrm{mcr}(j)$ may cause an *overload* that can possibly compromise the system schedulability (because the schedulability analysis performed offline on the new-mode tasks of $\widetilde{\tau}^j$ did not take into account the additional amount of execution requested by the rem-jobs).

To solve this problem, mode-change protocols usually *delay* the enablement of each MS task of the new-mode until it is safe to enable them. However, multimode systems assume that these delays are also subject to hard constraints. More precisely, we denote by $\mathcal{D}_k^j(M^i)$ *the relative transition deadline of every MS task* $\tau_k^j \in \tau^j$ *during every transition from mode* M^i *to mode* M^j, with the following interpretation: the mode-change protocol must ensure that τ_k^j is enabled not later than time $t_{\mathrm{mcr}(j)} + \mathcal{D}_k^j(M^i)$. Finally, when all the rem-jobs are completed and all the MS tasks of the new-mode are enabled, the transition phase ends and the system is considered as running in mode M^j.

In short, the goal of any mode-change protocol is to fulfill the following requirements during every mode transition:

1) Complete every job by its absolute deadline.
2) Enable every MS task τ_k^j of the new-mode M^j by its absolute transition deadline $t_{\mathrm{mcr}(j)} + \mathcal{D}_k^j(M^i)$, assuming that M^i is the old-mode.

IV. Two interesting observations

Upon a mode change request, the most intuitive idea to safely perform the desired mode transition is to keep scheduling the rem-jobs together with the MI tasks until a time-instant t such that both of the following conditions are true:

1) all the rem-jobs have completed execution by time t;
2) enabling all the MS tasks of the new-mode at time t does not jeopardize the schedulability of the system.

Intuitively, the first condition seems to imply the second one, but we observed the two following phenomenons.

Observation 1: After disabling a task τ_ℓ, it may be the case that former executions of τ_ℓ have affected the schedule permanently.

This phenomenon clearly appears in the following example. Consider the task system described in Table (a). The upper schedule of Figure 2 is the schedule of τ_1^{mit} (in gray), τ_2^{mit} (in dark) and τ_3^{mit} (in white) produced by EDF on a 2-processors platform, assuming that jobs of every task τ_ℓ are released exactly T_ℓ time units apart and execute for C_ℓ time units. The lower schedule of this figure is the schedule produced by EDF of $\tau_1^{\mathrm{mit}}, \tau_2^{\mathrm{mit}}, \tau_3^{\mathrm{mit}}$ and τ_4 (hatched pattern), where τ_4 is disabled at time 28. Here again, all the jobs are assumed to be released periodically and executed for their WCET. As we can observe, if the job-release pattern of $\tau_1^{\mathrm{mit}}, \tau_2^{\mathrm{mit}}$ and τ_3^{mit} never changes after time 28 (as well as the execution time of the released jobs), then the lower schedule will permanently conserve the "print" of the former executions of τ_4.

[3] Aborting a job consists in suddenly stopping its execution and removing it from the system memory. But in the real world, suddenly killing a process may cause system failures and the rem-jobs often have to complete their execution.

[4] Assuming EDF, this property is a direct consequence of the sustainability, because disabling a task is equivalent to set the execution time of all its next released jobs to zero.

τ	C_i	D_i	T_i
τ_1^{mit}	3	9	10
τ_2^{mit}	10	19	20
τ_3^{mit}	10	20	20
τ_4	1	9	10
$\lambda_{\text{sum}}(\tau)$	14/9		
$\lambda_{\text{max}}(\tau)$	0.5		

(a) System used in Observation 1 (Figure 2).

τ_1^{mit}	$C_1^{\text{mit}} = 10,\ D_1^{\text{mit}} = 20$									
τ_2^{mit}	$C_2^{\text{mit}} = 10,\ D_2^{\text{mit}} = 20$									
	mode 1: τ^1		mode 2: τ^2		mode 3: τ^3		mode 4: τ^4		mode 5: τ^5	
	C_i^1	D_i^1	C_i^2	D_i^2	C_i^3	D_i^3	C_i^4	D_i^4	C_i^5	D_i^5
τ_1^k	5	20	7	20	8	20	9	20	2	10
τ_2^k	5	20	2	20	1	20	1	20	2	10

$\lambda_{\text{sum}}(\widetilde{\tau^k})$	1.5	29/20	29/20	1.5	7/5
$\lambda_{\text{max}}(\widetilde{\tau^k})$	0.5	0.5	0.5	0.5	0.5

(b) System used in Observation 2 (Figure 3). $\forall i, k$: $D_i^k = T_i^k$.

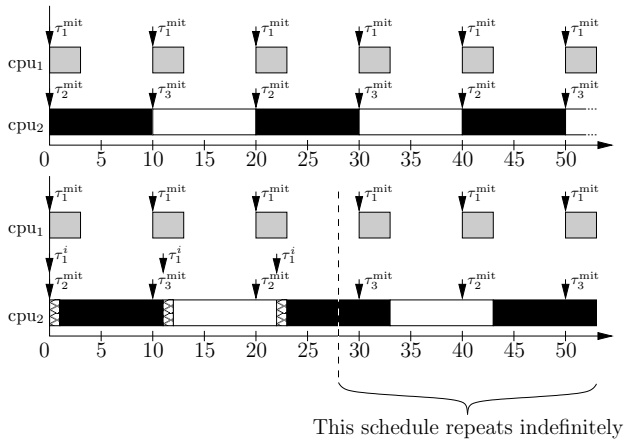

Figure 2. The fact that τ_4 has been scheduled over $[0, 28]$ has an endless impact on the whole schedule.

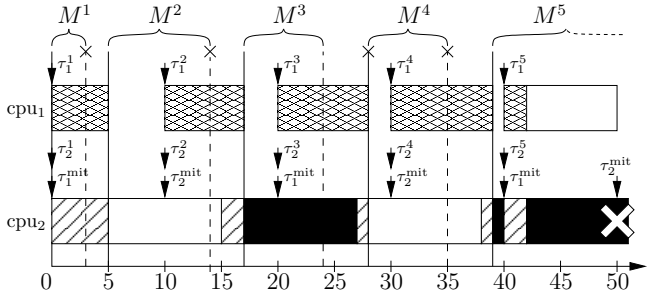

Figure 3. A deadline is missed at time 50 in the schedule of the multimode system described in Table (b).

Informally speaking, one can consider that the lower schedule is "late" compared to the upper schedule, with the interpretation that at any time-instant t, the remaining processing time of every task is larger (or equal) in the lower schedule than that in the upper schedule; and this lateness is never caught up in the above example. Consequently, the most natural question at this stage is: "Is this lateness able to jeopardize the system schedulability while switching from one mode to another?". We answered this question in our second observation.

Observation 2: Even if every mode has been separately asserted to be EDF-schedulable, it may be the case that the lateness produced by the successive executions of the modes propagates through the schedule and ultimately leads to a deadline miss.

This second phenomenon appears in Figure 3, where we illustrate the schedule of the task system given in Table (b) on a 2-processors platform, using EDF. In this figure: τ_1^{mit} and τ_2^{mit} are colored in plain white and black respectively, the first and second MS tasks for all modes (τ_1^i and τ_2^i, $\forall i$), are depicted with hatched and striped patterns respectively. The vertical dotted lines are the time-instants at which the mode changes are *requested* whereas the vertical plain lines are the instants at which those mode changes are *performed* (i.e., the new-mode MS tasks are enabled). Here we assume that mode changes are performed as soon as the rem-jobs have finished execution. This mechanism is used, for instance, by the protocol "without periodicity" SM-MSO introduced in [12] for identical multiprocessor platforms and extended in [13] to uniform multiprocessor platforms. In this example, although it may be verified that each mode of this system is EDF-schedulable[5], assuming specific job release patterns and execution times leads to a deadline miss at time 50.

According to Observations 1 and 2, after any mode change, the "lateness" generated by the MS tasks of the previous executed modes can (i) propagate indefinitely through the schedule of the following modes and (ii) compromise their schedulability. Therefore, upon a mode change request, it might be irrelevant to keep scheduling the rem-jobs together with the MI tasks until reaching a time-instant at which enabling all the MS tasks of the new-mode is safe in terms of schedulability. Indeed, there could be some particular situations for which, after a mode change, the lateness produced by the execution of the former modes (i) propagates indefinitely through the schedule of the MI tasks (from Observations 1) and (ii) continuously prevents the new-mode tasks from being enabled without jeopardizing the system schedulability (from Observations 2). The conclusion

[5]using, for example, the simple and well-known EDF-schedulability test proposed in [20] (for sporadic implicit-deadline tasks) and extended in [21] to sporadic constrained-deadline tasks, i.e., $\frac{\lambda_{\text{sum}}(\tau)-\lambda_{\text{max}}(\tau)}{1-\lambda_{\text{max}}(\tau)} \leq m$.

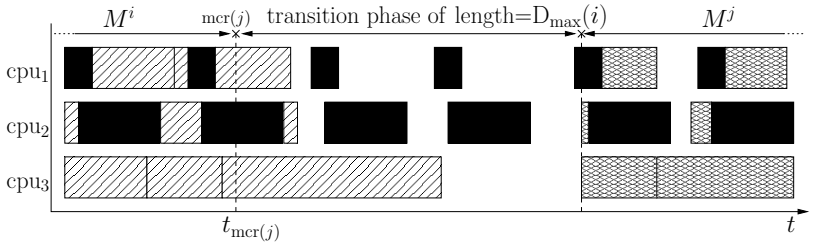

Figure 4. Illustration of how SM-MDO manages mode transitions.

is inescapable: the impact that every task of every mode has on the schedule must be taken into account in the schedulability analysis of each mode. In former studies related to mode changes without the consideration of MI tasks, ensuring that all the deadlines are met required only that a schedulability test was performed (i) on the tasks of each mode and (ii) for each mode transition. When MI tasks are part of the system, ensuring that all the deadlines are met requires also that a schedulability test is performed in order to guarantee that the lateness introduced above does not jeopardize the schedulability, i.e., it must be proven that scheduling the different feasible task sets successively does not jeopardize the schedulability of each one. In order to clearly distinguish the role of each test, let us introduce (or refine) the following two definitions.

Definition 6 (Validity test): For a given multimode system τ, multiprocessor platform π and mode-change protocol P, a validity test $\mathcal{V}(\tau, \pi, P)$ is a condition that indicates *a priori* whether any transition between any pair (M^i, M^j) of modes of τ meets all the transition deadlines $\mathcal{D}_k^j(M^i)$, $1 \leq k \leq n_j$.

Definition 7 (Schedulability test): For a given multimode system τ, multiprocessor platform π and mode-change protocol P, a schedulability test $\mathcal{S}(\tau, \pi, P)$ is a condition that indicates *a priori* whether the execution of τ upon π, assuming that P is used to manage the mode transitions, meets all the deadlines D_k^i and D_k^{mit} ($\forall i, k$).

The problem of extending existing mode-change protocols (such as SM-MSO for instance) so that they support MI tasks has revealed to be particularly challenging, and especially extending their schedulability analyses to a *system-wide* schedulability analysis. For this reason, we propose a new mode-change protocol, SM-MDO, in the following section. A validity analysis for this protocol is provided in Section VI and a system-wide schedulability analysis in Section VII.

V. The synchronous protocol SM-MDO

The main idea of the protocol SM-MDO (which stands for "Synchronous Multiprocessor Maximum Deadline Offset") is simple: upon a $\mathrm{mcr}(j)$, all the tasks of the old-mode (say M^i) are disabled and the rem-jobs continue to be scheduled upon the m CPUs, together with the MI tasks. Then, $\mathrm{D}_{\max}(i) \stackrel{\mathrm{def}}{=} \max_{k=1}^{n_i}\{D_k^i\}$ time units after the mode change was requested, *all* the MS tasks of the new-mode (i.e., the tasks of τ^j) are simultaneously enabled. At that time $t_{\mathrm{mcr}(j)} + \mathrm{D}_{\max}(i)$, all the rem-jobs have finished execution by their respective deadline. This is due to the sustainability of EDF and because the old-mode MS tasks stopped releasing jobs from time $t_{\mathrm{mcr}(j)}$.

Notice that, at any time-instant during any mode transition, if all the CPUs idle simultaneously then it can easily be shown that it is safe to directly enter the new-mode without waiting for $\mathrm{D}_{\max}(i)$ time units to elapse (because the behavior of earlier jobs will not impact the new-mode tasks after such an instant).

Example 1: Let us consider a platform composed of 3 CPUs and an application composed of 2 modes M^i and M^j. Figure 4 depicts an example of a schedule in which the system is switching from mode M^i to mode M^j. All the MI tasks are displayed in black whereas all the MS tasks of modes M^i and M^j are depicted with striped and hatched pattern, respectively. Note that this example does not rely on any specific scheduler or task parameters and is introduced only to clarify the mechanism defined by SM-MDO. In this picture, the system requests a mode change at time $t_{\mathrm{mcr}(j)}$. Here starts the transition phase from mode M^i to mode M^j. As specified by SM-MDO, all the MS tasks of the old-mode are immediately disabled and the rem-jobs continue to be scheduled, together with the MI tasks. After $\mathrm{D}_{\max}(i) \stackrel{\mathrm{def}}{=} \max_{k=1}^{n_i}\{D_k^i\}$ time units have elapsed, SM-MDO immediately enables all the new-mode tasks. Here ends the transition phase and the system is then running in mode M^j.

Notice that during any mode transition, SM-MDO allows the system (or any task) to request any other mode change at any time. In that case, the system records the last mode change to be requested and enables all the tasks of the specified mode at the end of the transition phase.

VI. Validity test for SM-MDO

In order to guarantee the validity of SM-MDO, it must be the case that the relative deadline D_k^i of any MS task τ_k^i of any mode M^i is not larger than the relative transition

1: **for** $i \leftarrow 2, 3, \ldots$ **do**
2: Let j_i denote a job that
- is released at some time-instant $t_i < t_{i-1}$;
- has a deadline after time t_{i-1};
- has not completed execution by time t_{i-1};
- has executed for strictly less than $(t_{i-1} - t_i) \times \sigma$ units over the time interval $[t_{i-1}, t_i)$.

3: **if** there is no such job **then**
4: $k \leftarrow i - 1$
5: break (exit the for loop)
6: **end if**
7: **end for**

Figure 5. Algorithm for constructing a sequence of jobs j_i [16].

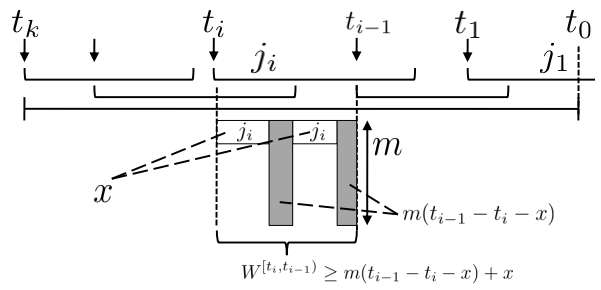

Figure 6. Notations of Lemma 1

deadline $\mathcal{D}_r^j(M^i)$ of any task τ_r^j of any other mode M^j. Indeed, since SM-MDO enables all the MS tasks of the new-mode exactly $\mathrm{D}_{\max}(i) \stackrel{\text{def}}{=} \max_{k=1}^{n_i}\{D_k^i\}$ time units after the mode change request, it follows from this case that all transition deadlines will be met during every possible mode change. Formally, a validity test for SM-MDO can be formulated as follows.

Validity Test 1 (For SM-MDO*):* For any multimode real-time application τ and any identical multiprocessor platform, protocol SM-MDO is valid provided that, for every mode M^i,

$$\max_{k=1}^{n_i}\left\{D_k^i\right\} \leq \min_{\substack{j=1 \\ j\neq i}}^{x}\left\{\min_{r=1}^{n_j}\left\{\mathcal{D}_r^j(M^i)\right\}\right\} \tag{9}$$

VII. Schedulability test for SM-MDO

In this section, we design a schedulability test specific to the protocol SM-MDO and the scheduling algorithm EDF. This schedulability test is a direct consequence of Theorem 1 proved below. We want to stress that the proof of this theorem has been widely inspired from Theorem 1 of [16].

Theorem 1: Suppose that a multimode constrained-deadline sporadic task system τ is not EDF-schedulable upon a platform composed of m processors. Then it holds for all $\sigma \geq \max_{k=1}^{x}\{\lambda_{\max}(\widetilde{\tau}^k)\}$ that[6]

$$\max_{k=1}^{x}\left\{\mathrm{LOAD}(\tau^k)\right\} + \text{FF-LOAD}(\tau^{\text{mit}}, \sigma) > m - (m-1)\sigma$$

Proof: Let $\sigma \geq \max_{k=1}^{x}\{\lambda_{\max}(\widetilde{\tau}^k)\}$. Let t_0 denote the first instant at which a deadline is missed. Let j_1 denote a job that missed its deadline at time t_0, and let t_1 denote the release time of j_1. From the definitions of s and j_1, we know that j_1 has executed for strictly less than $(t_0 - t_1) \times \sigma$ execution units over the time interval $[t_1, t_0)$. Let us define the sequence of jobs j_i, time-instants t_i, and an index k, according to the pseudo-code given by Figure 5.

Let L denote the length of the interval $[t_k, t_0)$, i.e., $L \stackrel{\text{def}}{=} t_0 - t_k$, and let $W^{[t_k,t_0)}$ denote the total amount of execution performed by the m processors within $[t_k, t_0)$.

Lemma 1: The amount of work executed over $[t_k, t_0)$ is bounded from *below* by $L \times (m - (m-1)\sigma)$, i.e.,

$$W^{[t_k,t_0)} > L \times (m - (m-1)\sigma)$$

Proof: For each $i \in [1, k]$, we denote by $W^{[t_i,t_{i-1})}$ the total amount of execution performed by the m processors within $[t_i, t_{i-1})$. Figure 6 provides a visualization of the notations used in this proof. Let us focus on any single time interval $[t_i, t_{i-1})$, $1 \leq i \leq k$, and let x denote the total amount of time during which j_i executes within this interval. By definition of j_i, we know that

$$x < (t_{i-1} - t_i) \times \sigma$$

and because j_i has not completed execution at time t_{i-1}, it is the case that no processor idles whenever j_i is *not* executing.

$$\begin{aligned} W^{[t_i,t_{i-1})} &\geq m(t_{i-1} - t_i - x) + x \\ &= m(t_{i-1} - t_i) - (m-1)x \\ &> m(t_{i-1} - t_i) - (m-1)(t_{i-1} - t_i)\sigma \\ &= (m - (m-1)\sigma) \times (t_{i-1} - t_i) \end{aligned}$$

Finally, summing the $W^{[t_i,t_{i-1})}$ for all $i \in [1, k]$ yields

$$\begin{aligned} \sum_{i=1}^{k} W^{[t_i,t_{i-1})} &> \sum_{i=1}^{k}(m - (m-1)\sigma) \times (t_{i-1} - t_i) \\ &= (m - (m-1)\sigma) \times \sum_{i=1}^{k}(t_{i-1} - t_i) \\ &= (m - (m-1)\sigma) \times (t_0 - t_k) \\ &= (m - (m-1)\sigma) \times L \end{aligned}$$

and since $\sum_{i=1}^{k} W^{[t_i,t_{i-1})} = W^{[t_k,t_0)}$, it holds that

$$W^{[t_k,t_0)} > (m - (m-1)\sigma) \times L$$

and Lemma 1 is proven. ■

Lemma 2: The amount of work executed over $[t_k, t_0)$ is bounded from *above* by

$$L \times \max_{k=1}^{x}\left\{\mathrm{LOAD}(\tau^k)\right\} + \text{FF-DBF}(\tau^{\text{mit}}, L, \sigma)$$

[6]Recall that $\lambda_{\max}(\widetilde{\tau}^k)$ denotes the maximal density of the task set $\widetilde{\tau}^k$ (see page 3).

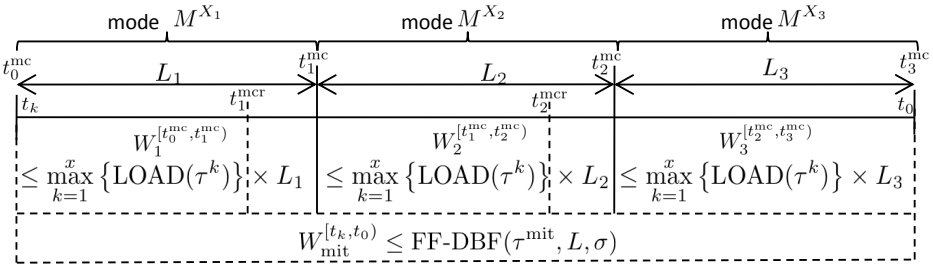

Figure 7. Notations of Lemma 2

where $L \stackrel{\text{def}}{=} t_0 - t_k$.

Proof: Consider the scenario depicted in Figure 7. Two mode changes occur in the time interval $[t_k, t_0)$. First, the system is running in mode M^{X_1} at time t_k. Then, a mode change is requested to mode M^{X_2} at time t_1^{mcr} and the mode change is performed at time t_1^{mc}, after a transition delay corresponding to the maximum relative deadline of the MS tasks of τ^{X_1}. Similarly, a mode change is requested at time t_2^{mcr} and the mode change is performed at time $t_2^{\mathrm{mc}} = t_2^{\mathrm{mcr}} + \mathrm{D}_{\max}(X_2)$, where $\mathrm{D}_{\max}(X_2) \stackrel{\text{def}}{=} \max_{k=1}^{n_{X_2}}\{D_k^{X_2}\}$. Notice that in the offline analysis, we don't know how many mode changes are actually requested in the time interval $[t_k, t_0)$, neither from/to which modes.

Let us define the following notations. Some of them are depicted in Figure 7.

- z denotes the number of mode changes that occur in the time interval $[t_k, t_0)$. In Figure 7, $z = 2$.
- t_j^{mc}, $1 \leq j \leq z$, denotes each time-instant at which a mode change is performed within $[t_k, t_0)$. We assume $t_0^{\mathrm{mc}} = t_k$ and $t_{z+1}^{\mathrm{mc}} = t_0$.
- L_j, $1 \leq j \leq z+1$, denotes the length of each interval of time $\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)$, i.e., $L_j \stackrel{\text{def}}{=} t_j^{\mathrm{mc}} - t_{j-1}^{\mathrm{mc}}$. During each of these intervals, the system is executing the tasks of only one mode.
- X_j denotes the mode running over $\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)$.
- $W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)}$, $1 \leq j \leq z+1$, denotes the total amount of work executed by the m processors within $\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)$, considering only the jobs issued from the MS tasks of τ^{X_j}.
- $W_{\mathrm{mit}}^{[t_k,t_0)}$ denotes the total amount of work executed by the m processors within $[t_k, t_0)$, considering only the jobs issued from the task set τ^{mit}.

Note that according to these notations,

$$W^{[t_k,t_0)} = \sum_{j=1}^{z+1} W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)} + W_{\mathrm{mit}}^{[t_k,t_0)} \tag{10}$$

The remainder of this proof is divided into two parts. The first one determines an upper-bound on $W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)}$, $\forall j \in [1, z+1]$, and the second one establishes an upper-bound on $W_{\mathrm{mit}}^{[t_k,t_0)}$.

Part 1: Let us focus on any single time interval $\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)$. Since the MS tasks of the mode M^{X_j} can release jobs only in the subinterval of time $\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mcr}}\right)$, the amount of execution that jobs of any task $\tau_\ell \in \tau^{X_j}$ contribute to $W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)}$ is bounded from above by

$$\left(\left\lfloor \frac{t_j^{\mathrm{mcr}} - t_{j-1}^{\mathrm{mc}}}{T_\ell} \right\rfloor + 1\right) \times C_\ell$$

By definition of protocol SM-MDO, we have $t_j^{\mathrm{mcr}} = t_j^{\mathrm{mc}} - \mathrm{D}_{\max}(X_j)$ and the above expression becomes

$$\left(\left\lfloor \frac{t_j^{\mathrm{mc}} - t_{j-1}^{\mathrm{mc}} - \mathrm{D}_{\max}(i)}{T_\ell} \right\rfloor + 1\right) \times C_\ell$$

By definition, $D_\ell \leq \mathrm{D}_{\max}(X_j)$ and this upper-bound can be rewritten as

$$\left(\left\lfloor \frac{t_j^{\mathrm{mc}} - t_{j-1}^{\mathrm{mc}} - D_\ell}{T_\ell} \right\rfloor + 1\right) \times C_\ell$$

which corresponds to $\mathrm{DBF}(\tau_\ell, t_j^{\mathrm{mc}} - t_{j-1}^{\mathrm{mc}})$ (see Expression 1). Therefore, we have

$$\begin{aligned} W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)} &\leq \sum_{\tau_\ell \in \tau^{X_j}} \mathrm{DBF}(\tau_\ell, L_j) \\ &= \mathrm{DBF}(\tau^{X_j}, L_j) \\ &\leq \max_{k=1}^{x} \{\mathrm{DBF}(\tau^k, L_j)\} \\ &\leq \max_{k=1}^{x} \{\mathrm{LOAD}(\tau^k) \times L_j\} \\ &= \max_{k=1}^{x} \{\mathrm{LOAD}(\tau^k)\} \times L_j \end{aligned}$$

Let $\mathrm{load}_{\max} \stackrel{\text{def}}{=} \max_{k=1}^{x} \{\mathrm{LOAD}(\tau^k)\}$. The above inequality can be rewritten as

$$W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)} \leq \mathrm{load}_{\max} \times L_j$$

and thus,

$$\begin{aligned} \sum_{j=1}^{z+1} W_j^{\left[t_{j-1}^{\mathrm{mc}}, t_j^{\mathrm{mc}}\right)} &\leq \sum_{j=1}^{z+1} \mathrm{load}_{\max} \times L_j \\ &= \mathrm{load}_{\max} \times \sum_{j=1}^{z+1} L_j \\ &= \mathrm{load}_{\max} \times L \end{aligned} \tag{11}$$

Part 2: Concerning the amount $W_{\text{mit}}^{[t_k,t_0)}$ of execution, we use exactly the same reasoning as the one used in Lemma 1.1 of [16]. This reasoning is as follows: the contribution of any MI task $\tau_\ell^{\text{mit}} \in \tau^{\text{mit}}$ to $W_{\text{mit}}^{[t_k,t_0)}$ is bounded from above by the scenario in which a job of τ_ℓ^{mit} has its deadline exactly at time t_0, and prior jobs have arrived exactly T_ℓ^{mit} time-units apart. Under this scenario, the contribution of τ_ℓ^{mit} to $W_{\text{mit}}^{[t_k,t_0)}$ includes:

- at least $q_\ell^{\text{mit}} \stackrel{\text{def}}{=} \lfloor (t_0 - t_k)/T_\ell^{\text{mit}} \rfloor$ jobs of τ_ℓ^{mit} that lie entirely within $[t_k, t_0)$. For each such job its contribution is C_ℓ^{mit}.
- (perhaps) an additional job that has its deadline at time-instant $t_k + r_\ell^{\text{mit}}$, where $r_\ell^{\text{mit}} \stackrel{\text{def}}{=} (t_0 - t_k) \bmod T_\ell^{\text{mit}}$. The maximal contribution of this job depends on r_ℓ^{mit}.
  - if $r_\ell^{\text{mit}} \geq D_\ell^{\text{mit}}$ then this additional job arrives at or after t_k and its maximal contribution is C_ℓ^{mit}.
  - if $r_\ell^{\text{mit}} < D_\ell^{\text{mit}}$ then it arrives before time t_k. Therefore, since t_k does not denote its arrival time then this job does not belong to the sequence of jobs j_i defined by the pseudo-code of Figure 5; and since its deadline is posterior to t_k, the reason must be that this job has completed at least $(D_\ell^{\text{mit}} - r_\ell^{\text{mit}}) \times \sigma$ units of execution before time t_k. Hence, its remaining execution time at time-instant t_k is at most $\max(0, C_\ell^{\text{mit}} - (D_\ell^{\text{mit}} - r_\ell^{\text{mit}}) \times \sigma)$.

In either case, one can see that the maximal contribution of τ_ℓ^{mit} to $W_{\text{mit}}^{[t_k,t_0)}$ corresponds to $\text{FF-DBF}(\tau_\ell^{\text{mit}}, t_0 - t_k, \sigma)$ (see Equation 2). Therefore, it follows that

$$\begin{aligned} W_{\text{mit}}^{[t_k,t_0)} &\leq \sum_{\tau_\ell^{\text{mit}} \in \tau^{\text{mit}}} \text{FF-DBF}(\tau_\ell^{\text{mit}}, t_0 - t_k, \sigma) \\ &= \text{FF-DBF}(\tau^{\text{mit}}, L, \sigma) \end{aligned} \quad (12)$$

By summing the two Inequalities 11 and 12 obtained from Part 1 and 2, we get

$$\begin{aligned} &\sum_{j=1}^{z+1} W_j^{\left[t_{j-1}^{\text{mc}}, t_j^{\text{mc}}\right)} + W_{\text{mit}}^{[t_k,t_0)} \\ \leq\ & L \times \text{load}_{\max} + \text{FF-DBF}(\tau^{\text{mit}}, L, \sigma) \end{aligned}$$

and finally, from Equality 10, the left-hand side of the above inequality can be rewritten as

$$W^{[t_k,t_0)} \leq L \times \text{load}_{\max} + \text{FF-DBF}(\tau^{\text{mit}}, L, \sigma)$$

which states the lemma. ■

From Lemmas 1 and 2, if the system is not EDF-schedulable then there exists a time interval of length L such that

$$L \times \text{load}_{\max} + \text{FF-DBF}(\tau^{\text{mit}}, L, \sigma) > (m - (m-1)\sigma) \times L$$

Dividing both sides of this inequality by L yields

$$\text{load}_{\max} + \frac{\text{FF-DBF}(\tau^{\text{mit}}, L, \sigma)}{L} > m - (m-1)\sigma \quad (13)$$

and since it holds from Expression 8 (page 3) that, $\forall L$,

$$\frac{\text{FF-DBF}(\tau^{\text{mit}}, L, \sigma)}{L} \leq \text{FF-LOAD}(\tau^{\text{mit}}, \sigma)$$

then Inequality 13 can be rewritten as

$$\text{load}_{\max} + \text{FF-LOAD}(\tau^{\text{mit}}, \sigma) > m - (m-1)\sigma$$

and Theorem 1 is proven. ■

The contrapositive of the unschedulability condition of Theorem 1 provides a sufficient schedulability test. This test is given below.

Schedulability Test 1 (for SM-MDO *and EDF):* A multimode constrained-deadline sporadic task system τ is EDF-schedulable upon m identical processors, provided

$$\text{load}_{\max} + \text{FF-LOAD}(\tau^{\text{mit}}, \lambda_{\max}^{\max}) \leq m - (m-1) \cdot \lambda_{\max}^{\max} \quad (14)$$

where

$$\begin{aligned} \text{load}_{\max} &\stackrel{\text{def}}{=} \max_{k=1}^{x} \left\{ \text{LOAD}(\tau^k) \right\} \\ \lambda_{\max}^{\max} &\stackrel{\text{def}}{=} \max_{k=1}^{x} \{ \lambda_{\max}(\widetilde{\tau}^k) \} \end{aligned}$$

Observation 3: Schedulability Test 1 above can be seen as a generalization[7] of the GFB test [20] extended in [21] to constrained-deadline tasks. According to this extension, a task set τ is EDF-schedulable on m identical processors, provided

$$\frac{\lambda_{\text{sum}}(\tau) - \lambda_{\max}(\tau)}{1 - \lambda_{\max}(\tau)} \leq m \quad (15)$$

Proof: Since we know that for any task set τ, $\text{LOAD}(\tau) \leq \lambda_{\text{sum}}(\tau)$, it holds for all set $\{\tau^1, \tau^2, \ldots, \tau^x\}$ of task sets that

$$\max_{k=1}^{x} \left\{ \text{LOAD}(\tau^k) \right\} \leq \max_{k=1}^{x} \left\{ \lambda_{\text{sum}}(\tau^k) \right\}$$

and since $\forall \sigma > 0$, $\text{FF-LOAD}(\tau^{\text{mit}}, \sigma) \leq \lambda_{\text{sum}}(\tau^{\text{mit}})$, the schedulability condition 14 can be rewritten as

$$\max_{k=1}^{x} \left\{ \lambda_{\text{sum}}(\tau^k) \right\} + \lambda_{\text{sum}}(\tau^{\text{mit}}) \leq m - (m-1) \cdot \lambda_{\max}^{\max}$$

leading to

$$\max_{k=1}^{x} \left\{ \lambda_{\text{sum}}(\widetilde{\tau}^k) \right\} \leq m - (m-1) \cdot \max_{k=1}^{x} \left\{ \lambda_{\max}(\widetilde{\tau}^k) \right\}$$

If the system is composed of a single task set τ then the above inequality becomes

$$\lambda_{\text{sum}}(\tau) \leq m - (m-1) \cdot \lambda_{\max}(\tau)$$

which can be rewritten as

$$\frac{\lambda_{\text{sum}}(\tau) - \lambda_{\max}(\tau)}{1 - \lambda_{\max}(\tau)} \leq m \quad (16)$$

and the observation follows. ■

[7] By "generalization", we mean that Schedulability Test 1 is equivalent to the extension of the GFB test if they both assume the same system model, i.e., a system composed of a single set of tasks, which corresponds to a multimode system composed of a single mode.

Acknowledgements

This work is financed by FEDER funds (EU) through the Operational Programme "Thematic Factors of Competitiveness" - COMPETE, by National Funds (PT) through FCT - Portuguese Foundation for Science and Technology, the ARTEMIS-JU and the Luso-American Development Foundation (FLAD), under the projects RECOMP (ARTEMIS/0202/2009) and REJOIN.

VIII. Conclusions

In this paper, we proposed a new synchronous multiprocessor mode-change protocol SM-MDO that supports mode-independent tasks, and in particular, we designed a schedulability analysis for this protocol. As we showed through examples, this must be achieved by introducing a system-wide schedulability test that complements the validity analysis, previously described in the literature. This system-wide analysis ensures that scheduling the modes successively does not jeopardize the schedulability of each one.

As future line of work, we intend to relax the assumption that the modes share the same set of mode-independent tasks. This should enable us to apply the results to several other problems such as component scheduling/reservation based systems.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[2] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Systems*, vol. 26, no. 2, pp. 161–197, 2004.

[3] R. Henia and R. Ernst, "Scenario aware analysis for complex event models and distributed systems," in *Proceedings of the 28th IEEE International Real-Time Systems Symposium*, 2007, pp. 171–180.

[4] P. Pedro and A. Burns, "Schedulability analysis for mode changes in flexible real-time systems," in *proceedings of the 10th Euromicro Workshop on Real-Time Systems*, 1998, pp. 172–179.

[5] K. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority pre-emptively scheduled systems," in *Proceedings of the 13th Real-Time Systems Symposium*, 1992, pp. 100–109.

[6] B. Andersson, "Uniprocessor EDF scheduling with mode change," in *Proceedings of the 12th International Conference on Principles of Distributed Systems*, 2008, pp. 572–577.

[7] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable mode changes in real-time systems with fixed priority or EDF scheduling," in *proceedings of the Conference on Design, Automation and Test in Europe*, 2009, pp. 99–104.

[8] K. Tindell and A. Alonso, "A very simple protocol for mode changes in priority preemptive systems," Universidad Politécnica de Madrid, Tech. Rep., 1996.

[9] C. M. Bailey, "Hard real-time operating system kernel. Investigation of mode change," Task 14 Deliverable on ESTSEC Contract 9198/90/NL/SF, British Aerospace Systems Ltd., Tech. Rep., 1993.

[10] P. Pedro, "Schedulability of mode changes in flexible real-time distributed systems," Ph.D. dissertation, University of York, Department of Computer Science, 1999.

[11] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Systems*, vol. 1, pp. 243–264, 1989.

[12] V. Nélis, J. Goossens, and B. Andersson, "Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms," in *Proceedings of the 21st Euromicro Conference on Real-Time Systems (ECRTS'09)*, 2009, pp. 151–160.

[13] P. Meumeu Yomsi, V. Nélis, and J. Goossens, "Scheduling multi-mode real-time systems upon uniform multiprocessor platforms," in *Proceedings of the 15th IEEE International Conference on Emerging Technologies and Factory Automation*, 2010.

[14] V. Nélis, "Energy-aware real-time scheduling in multiprocessor embedded systems," Ph.D. dissertation, Université Libre de Bruxelles, 2010.

[15] A. Block, J. H. Anderson, and U. C. Devi, "Task reweighting under global scheduling on multiprocessors," *Real-Time Systems*, vol. 39, pp. 123–167, August 2008.

[16] S. Baruah, V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller, "Implementation of a speedup-optimal global EDF schedulability test," in *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, 2009, pp. 259–268.

[17] I. Ripoll, A. Crespo, and A. K. Mok, "Improvement in feasibility testing for real-time tasks," *Real-Time Systems*, vol. 11, no. 1, pp. 19–39, 1996.

[18] N. Fisher, T. P. Baker, and S. Baruah, "Algorithms for determining the demand-based load of a sporadic task system," in *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006, pp. 135–146.

[19] T. P. Baker and S. K. Baruah, "Sustainable multiprocessor scheduling of sporadic task systems," in *Proceedings of the 21st Euromicro Conference on Real-Time Systems*, 2009, pp. 141–150.

[20] J. Goossens, S. Funk, and S. Baruah, "Priority-driven scheduling of periodic task systems on multiprocessors," *Real-Time Systems*, vol. 25, pp. 187–205, September 2003.

[21] M. Bertogna, M. Cirinei, and G. Lipari, "Improved schedulability analysis of EDF on multiprocessor platforms," in *Proceedings of the 17th Euromicro Conference on Real-Time Systems*, 2005, pp. 209–218.