View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositório Científico do Instituto Politécnico do Porto

Technical Report

Two Protocols Without Periodicity for the Global and Preemptive Scheduling Problem of Multi-Mode Real-Time Systems upon Multiprocessor Platforms

Vincent Nelis, Joël Goossens and Björn Andersson

HURRAY-TR-090501
Version: 0
Date: 05-04-2009

Two Protocols Without Periodicity for the Global and Preemptive Scheduling Problem of Multi-Mode Real-Time Systems upon Multiprocessor Platforms

Vincent Nelis, Joël Goossens and Björn Andersson

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: bandersson@dei.isep.ipp.pt

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

Abstract

We consider the global and preemptive scheduling problem of multi-mode real-time systems upon identical multiprocessor platforms. Since it is a multi-mode system, the system can change from one mode to another such that the current task set is replaced with a new task set. Ensuring that deadlines are met requires not only that a schedulability test is performed on tasks in each mode but also that (i) a protocol for transitioning from one mode to another is specified and (ii) a schedulability test for each transition is performed. We propose two protocols which ensure that all the expected requirements are met during every transition between every pair of operating modes of the system. Moreover, we prove the correctness of our proposed algorithms by extending the theory about the makespan determination problem.

Two Protocols for Scheduling Multi-Mode Real-Time Systems upon Identical Multiprocessor Platforms

Vincent Nelis[1,2]
vnelis@ulb.ac.be

Joël Goossens[1]
joel.goossens@ulb.ac.be

Björn Andersson[3]
bandersson@dei.isep.ipp.pt

Abstract

We consider the problem of scheduling a multi-mode real-time system upon identical multiprocessor platforms. Since it is a multi-mode system, the system can change from one mode to another such that the current task set is replaced with a new task set. Ensuring that deadlines are met requires not only that a schedulability test is performed on tasks in each mode but also that (i) a protocol for transitioning from one mode to another is specified and (ii) a schedulability test for each transition is performed. We propose two protocols which ensure that all the expected requirements are met during every transition between every pair of operating modes of the system. Moreover, we prove the correctness of our proposed algorithms by extending the theory about the makespan determination problem.

1. Introduction

Hard real-time systems require both functionally correct executions and *results that are produced on time*. Control of the traffic (ground or air), control of engines, control of chemical and nuclear power plants are just some examples of such systems. Currently, numerous techniques exist that enable engineers to design real-time systems while guaranteeing that all their temporal requirements are met. These techniques generally model each functionality of the system by a *recurrent* task, characterized by a computing requirement, a temporal deadline and an activation rate. Commonly, real-time systems are modeled by a set of such tasks. However, some applications exhibit multiple behaviors issued from several operating modes (e.g., an initialization mode, an emergency mode, a fault recovery mode, etc.), where each mode is characterized by its own set of functionalities, i.e., its set of tasks. During the execution of such *multi-mode* real-time systems, switching from the current mode (called the *old-mode*) to another one (the *new-mode* hereafter) requires to substitute the current executing task set with the set of tasks of the target mode. This substitution introduces a transient stage, where the tasks of the old- and new-mode may be scheduled *simultaneously*, thereby leading to an overload which can compromise the system schedulability.

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 (see e.g., [5, 9] for details about the different task requirements during mode transitions). For instance, an *old-mode task* may be immediately aborted, or it may require to complete the execution of its current instance in order to preserve data consistency. In this document, we assume that *every old-mode task must complete its current instance* when a mode change is requested. Indeed we will see that, while using scheduling algorithms such as the one considered in this paper, tasks which can be aborted upon a mode change request do not have any impact on the schedulability during the mode transitions. On the other hand, a *new-mode task* sometimes requires to be activated as soon as possible, or it may also have to delay its first activation until all the tasks of the old-mode are completed. Moreover, there may be some tasks (called *mode-independent* tasks) present in both the old- and new-mode, such that their periodic (or sporadic) execution pattern must not be influenced by the mode change in progress.

The existing transition scheduling protocols are classified with respect to (i) their ability to deal with the mode-independent tasks and (ii) the way they schedule the old- and new-mode tasks during the transitions. In the literature (see [13] for instance), a protocol is said to be *synchronous* if it does not schedule old- and new-mode tasks simultaneously, otherwise it is said to be *asynchronous*. Furthermore, a synchronous/asynchronous protocol is said to be *with periodicity* if it is able to deal with mode-independent tasks, otherwise it is said to be *without periodicity*.

[1]Université Libre de Bruxelles (U.L.B.) CP 212, 50 Av. F.D. Roosevelt, B-1050 Brussels, Belgium.

[2]Supported by the Belgian National Science Foundation (FNRS) under a FRIA grant.

[3]Polytechnic Institute of Porto, Rua Dr. Antonio Bernardino de Almeida 431, 4200-072 Porto, Portugal.

Related work. Numerous scheduling protocols have already been proposed in the *uni*processor case to ensure the transition between modes (a survey of the literature about this uniprocessor problem is provided in [13]). In synchronous protocols, one can cite the *Idle Time Protocol* [14] where the periodic activations of the old-mode tasks are suspended at the first idle time-instant occurring during the transition and then, the new-mode tasks are released. The *Maximum-Period Offset Protocol* proposed in [2] is a protocol *with periodicity* which delays the first activation of all the new-mode tasks for a time equal to the period of the less frequent task in both modes (mode-independent tasks are not affected). The *Minimum Single Offset Protocol* in [13] completes the last activation of all the old-mode tasks and then, releases the new-mode ones. This protocol exists in two versions, *with* and *without periodicity*. Concerning the *asynchronous* protocols, the authors of [15] propose a protocol *with periodicity* and the authors of [12, 11] propose a protocol *without periodicity*.

To the best of our knowledge, no protocol exists in the *multi*processors case (except for the "Work in progress" version of this paper [10]). This problem is much more complex, especially due to the presence of scheduling anomalies upon multiprocessors (see, e.g., Chapter 5 of [1] on page 51 for a definition). Nowadays, it is well-known that real-time multiprocessor scheduling problems are typically not solved by applying straightforward extensions of techniques used for solving similar uniprocessor problems.

This research. In this paper, we propose two protocols *without periodicity* (a *synchronous* protocol and an *asynchronous* one) for managing every mode transition during the execution of a multi-mode real-time system on a multiprocessor platform. Both protocols can be considered to be a generalization to the multiprocessor case of the Minimal Single Offset (MSO) protocol proposed in [13].

Paper organization. In Section 2, we define the computational model used throughout the paper. In Section 3, we propose a synchronous protocol and we prove its correctness by extending the theory about the makespan determination problem. In Section 4, we propose an asynchronous protocols and we prove that it also meets all the expected requirements during every mode transition. Finally, Section 5 gives conclusions.

2. Model of computation

2.1. System and platform specifications

We consider multiprocessor platforms composed of a known and fixed number m of *identical* processors $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m\}$ upon which a multi-mode real-time system is executed. "Identical" means that all the processors have the same profile (in term of consumption, computational capabilities, etc.) and are interchangeable.

We define a multi-mode real-time system τ as a set of x operating modes noted $M^1, M^2, \ldots, M^x$ where each mode contains its own set of functionalities to execute. At any time during its execution, the system runs in only one of its modes, i.e., it executes only the set of tasks associated with the selected mode, or the system switches to one mode to another one. A mode M^k contains a set τ^k of n_k functionalities denoted $\left\{\tau_1^k, \tau_2^k, \ldots, \tau_{n_k}^k\right\}$. Every functionality τ_i^k is modeled by a sporadic constrained-deadline task characterized by three parameters (C_i^k, D_i^k, T_i^k) – a worst-case execution time C_i^k, a minimum inter-arrival separation T_i^k and a relative deadline $D_i^k \leq T_i^k$ – with the interpretation that, during the execution of the mode M^k, the task τ_i^k generates successive *jobs* $\tau_{i,j}^k$ (with $j = 1, \ldots, \infty$) arriving at times $a_{i,j}^k$ such that $a_{i,j}^k \geq a_{i,j-1}^k + T_i^k$ (with $a_{i,1}^k \geq 0$), each such job has an execution requirement of at most C_i^k, and must be completed at (or before) its deadline noted $D_{i,j}^k \stackrel{\text{def}}{=} a_{i,j}^k + D_i^k$. Since $D_i^k \leq T_i^k$, successive jobs of a task τ_i^k do not interfere with each other. In our study, 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.

At any time t during the system execution, a job $\tau_{i,j}^k$ is said to be *active* iff $a_{i,j}^k \leq t$ and it is not completed yet. Moreover, an *active* job is said to be *running* at time t if it is allocated to a processor. Otherwise, the active job is waiting in a ready-queue of the operating system and we say that it is *waiting*. We respectively denote by $\text{active}(\tau^k, t)$, $\text{run}(\tau^k, t)$ and $\text{wait}(\tau^k, t)$ the subsets of active, running and waiting jobs of the tasks of τ^k at time t. Notice that the following relation holds: $\text{active}(\tau^k, t) \stackrel{\text{def}}{=} \text{run}(\tau^k, t) \cup \text{wait}(\tau^k, t)$.

A task must be *enabled* to generate jobs, and the system is said to run in mode M^k only if every task of τ^k is enabled and all the tasks of the other modes are disabled. Thereby, disabling a task τ_i^k prevents future job arrivals from τ_i^k. In the following, we respectively denote by $\text{enabled}(\tau^k, t)$ and $\text{disabled}(\tau^k, t)$ the subsets of *enabled* and *disabled* tasks of τ^k at time t.

2.2. Scheduler specifications

We consider in this study the *global* scheduling problem of sporadic constrained-deadlines tasks on multiprocessor platforms. "Global" scheduling algorithms, on the contrary to partitioned algorithms, allow different tasks and *different* jobs of the same task to be executed upon *different* processors. Furthermore, we consider *preemptive*, *work-*

conserving and *fixed job-level priority assignment* according to the following interpretations:

- a *preemptive* global scheduler: every job can start its execution on any processor and *may migrate at run-time* to any other processor (with no loss or penalty) if it gets meanwhile preempted by a higher-priority job.
- a *work-conserving* global scheduler: a processor cannot be idle if there is a waiting job. In this paper, we assume that the scheduler assigns, at each time-instant during the system execution, the m highest priority active jobs (if any) to the m processors.
- a *fixed job-level priority* assignment: the scheduler assigns a priority to jobs as soon as they arrive and every job keeps its priority constant until it completes. Notice that, if the set of highest-priority jobs do not change during a time interval $[t_0, t_1)$ then no preemption or task migration occurs during the time interval $[t_0, t_1)$.

Moreover, we consider scheduling algorithms S for which the following property holds: for any set of tasks τ^ℓ and any integers $m > 0$ and $m' > m$, if S schedules τ^ℓ upon m processors without missing any deadline, then it also schedules τ^ℓ upon m' processors without missing any deadline. *Global Deadline Monotonic* and *Global Earliest Deadline First* [3] are just some examples of such scheduling algorithms.

Every mode M^k of the system uses its own scheduling algorithm noted S^k, which is *global*, *preemptive*, *work-conserving*, and which assigns *fixed job-level priorities*. In the remainder of this paper, we assume that every mode M^k can be scheduled by S^k on m processors without missing any deadline. This assumption allows us to only focus on the schedulability of the system *during the mode transitions*, and not during the executions of the modes. We denote by $J_i >_{S^k} J_j$ the fact that job J_i has a higher priority than J_j according to the scheduler S^k, and we assume that every assigned priority is distinct from the others, i.e., $\forall k, i, j$ such as $i \neq j$ we have either $J_i >_{S^k} J_j$ or $J_i <_{S^k} J_j$.

2.3. Mode transition specifications

While the system is running in a mode M^i, a mode change can be initiated by any task of τ^i or by the system itself, whenever it detects a change in the environment or in its internal state. This is performed by invoking a $\mathrm{MCR}(j)$ (i.e., a Mode Change Request), where M^j is the targeted destination mode (i.e., the new-mode). We denote by $t_{\mathrm{MCR}(j)}$ the invoking time of a $\mathrm{MCR}(j)$ and we assume that a MCR may only be invoked in the steady state of the system, and not during the transition between two modes.

Suppose that the system is running in mode M^i and a $\mathrm{MCR}(j)$ is invoked (with $j \neq i$). At time $t_{\mathrm{MCR}(j)}$, the system entrusts the scheduling decisions to a transition protocol. This protocol *immediately* disables all the old-mode tasks (i.e., the tasks of τ^i), *hence preventing new job arrivals from these tasks*. The active old-mode jobs at time $t_{\mathrm{MCR}(j)}$, henceforth called the *rem-jobs*, may have two distinct behaviors: either they can be aborted, or they must complete their execution. As it was introduced in Section 1, we assume in this work that every rem-jobs *must* complete. Indeed we will see in the following section that it is the *worst* scenario, since jobs which can be aborted do not have any impact on the system schedulability while considering schedulers such as the one introduced in the previous section. As a consequence, *every result proposed in this paper also holds while considering that some rem-jobs can be aborted*. Finally, notice that we do not consider mode-independent tasks in this paper.

Since the rem-jobs may cause an overload if the tasks of τ^j are immediately enabled upon a $\mathrm{MCR}(j)$, the transition protocols usually have to delay the enablement of these new-mode tasks until it is safe to enable them. We denote by $\mathcal{D}_k^j(M^i)$ *the relative enablement deadline of the task* τ_k^j *during the transition from the mode* M^i *to the mode* M^j, with the following interpretation: the transition protocol must ensure that τ_k^j is not enabled after time $t_{\mathrm{MCR}(j)} + \mathcal{D}_k^j(M^i)$. The goal of a transition protocol is therefore *to complete every rem-job and to enable every task of the new-mode* M^j*, while meeting all the job and enablement deadlines*. When all the rem-jobs are completed and all the tasks of τ^j are enabled, the system entrusts the scheduling decisions to the scheduler S^j of the new-mode M^j and the transition phase ends.

Definition 1 (a valid protocol) *A transition scheduling protocol is said to be* valid *for a given multi-mode real-time system if it always meets all the job and enablement deadlines during the transition from any mode of the system to any other one.*

3. The protocol SM-MSO

In this section, we present our synchronous protocol without periodicity called "Synchronous Multiprocessor Minimum Single Offset" (SM-MSO) protocol. The main idea is the following: *upon a* $\mathrm{MCR}(j)$*, every task of the current mode (say* M^i*) is disabled and the rem-jobs continue to be scheduled by* S^i *upon the* m *processors. When* all *of them are completed, all the new-mode tasks (i.e., the tasks of* τ^j*) are simultaneously enabled.* Figure 1 depicts an example with a 2-processors platform. The modes M^i and M^j respectively contain 4 and 3 tasks, where the light and

dark gray boxes are the old- and new-mode tasks, respectively. Algorithm 1 gives the pseudo-code of this protocol.

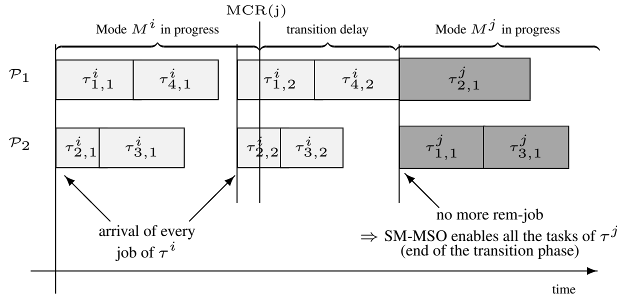

Figure 1. Illustration of a mode transition handled by SM-MSO.

Algorithm 1: SM-MSO

Input: t: current time; M^i: the old mode; M^j: the new-mode
begin
  Schedule all the jobs of τ^i by S^i ;

  At job (say J_k) completion time t :
  if $\text{active}(\tau^i, t) = \phi$ **then**
    enable all the tasks of τ^j ;
    enter mode M^j ;
end

In order to know if this protocol is valid for a given multi-mode system, we need to establish a *validity test*, i.e., a condition based on the tasks and platform characteristics which indicates *a priori* whether the given system will always comply with the expected requirements during every mode change. In the following, we first introduce the notion of *predictable* algorithms and we prove in Lemma 2 that disabling the old-mode tasks upon a mode change request does not jeopardize the schedulability of the rem-jobs, when they continue to be scheduled by the old-mode scheduler upon the m processors. Then in Lemma 5, we establish an upper bound on the maximal transition delay which could be produced by the completion of the rem-jobs. Finally, we use this upper bound to provide a *sufficient schedulability condition* that indicates, a priori, if all the enablement deadlines will be met during all possible mode changes.

Definition 2 (predictability [8]) *Let A denote a scheduling algorithm, and let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs, where each job $J_i = (a_i, c_i, d_i)$ is characterized by an arrival time a_i, a computing requirement c_i and an absolute deadline d_i. Let f_i denote the time at which job J_i completes its execution when J is scheduled by A. Now, consider any set $J' = \{J'_1, J'_2, \ldots, J'_n\}$ of n jobs obtained from J as follows. Job J'_i has an arrival time a_i, an execution requirement $c'_i \leq c_i$, and a deadline d_i (i.e., job J'_i has the same arrival time and deadline as J_i, and an execution requirement no larger than J_i's). Let f'_i denote the time at which job J'_i completes its execution when J' is scheduled by A. Algorithm A is said to be predictable if and only if for any set of jobs J and for any such J' obtained from J, it is the case that $f'_i \leq f_i \; \forall i$.*

The result from [6] that we will be using can be stated as follows.

Lemma 1 *Any global, preemptive, work-conserving and fixed job-level priority assignment scheduler is predictable.*

Lemma 2 *When a $\text{MCR}(j)$ occurs at time $t_{\text{MCR}(j)}$ while the system is running in mode M^i, every rem-job issued from the tasks of τ^i meets its deadline when scheduled by S^i upon m processors.*

Proof *From our assumptions, we know that the set of tasks τ^i of the mode M^i is schedulable by S^i upon m processors, and from Lemma 1 we know that S^i is* predictable. *When the $\text{MCR}(j)$ occurs at time $t_{\text{MCR}(j)}$, SM-MSO disables every task of τ^i. Disabling these tasks is equivalent to set the execution time of all their future jobs to zero, and since S^i is predictable the deadline of every rem-job is still met.* ■

From Lemma 2, every rem-job always meets its deadline while using SM-MSO during the transition phases. Thereby, SM-MSO is valid for a given multi-mode real-time system if, for every mode change, *the maximal transition delay which could be produced by the rem-jobs is not larger than the minimal enablement deadline of the new-mode tasks*. The transition delay is equal to the completion time of all the rem-jobs. Notice that, since (i) we consider fixed job-level priority assignment and (ii) the old-mode tasks are disabled upon any mode change request, there is no job arrival and therefore no preemption during the whole transition phase while using SM-MSO.

The completion time of a given set of n jobs (all ready at a same time) executed upon a given number m of processors is called the *makespan* in the literature and hereafter. Unfortunately, authors usually address the NP-hard problem of determining a job priority assignment which minimizes the makespan. This problem, of finding priorities to minimize the makespan, can be cast as a bin-packing problem for which a vast literature base is available. However in this work, we focus on the *maximal makespan* that could be produced by any set of n jobs (all ready at a same time) executed upon m processors, without relying on any specific job-level fixed priority.

In the following, $J = \{J_1, J_2, \ldots, J_n\}$ denotes a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processor by a scheduler such as the one described in Section 1. In Lemma 3, we first determine the latest instant in the schedule of J at which job J_i

may start its execution. Then, we determine in Lemma 5 an upper bound on the makespan that could be produced by J. To help the reader, Figure 2 illustrates the main notations used in the following proofs. In this figure, 7 jobs $\{J_1, J_2, \ldots, J_7\}$ ready at time 0 are scheduled upon 4 processors by a scheduler S (with $J_1 >_S J_2 >_S \ldots >_S J_7$) such as the one described in Section 1. We shall use the notations:

- W_i^j denotes the exact amount of processing times assigned to the processor $\mathcal{P}_j$, after having scheduled every job with a higher priority than J_i.
- t_i^{alloc} denotes the smallest instant in the schedule at which job J_i is assigned to a CPU. Notice that job J_i is always assigned to a processor at time $t_i^{\text{alloc}} = \min_{k=1}^{m}\{W_i^k\}$ while using a scheduler such as the one considered in this paper.
- t_i^{comp} denotes the exact completion time of the job J_i. For instance, we have $t_1^{\text{comp}} = t_7^{\text{alloc}}$ and $t_3^{\text{comp}} = t_6^{\text{alloc}}$.
- t_i^{idle} (with $i = 1, \ldots, m$) denote the smallest instant in the schedule such that at least i CPUs are idle.
- "makespan" denotes the time needed to complete all the rem-jobs. Notice that the makespan is always equal to t_m^{idle}.

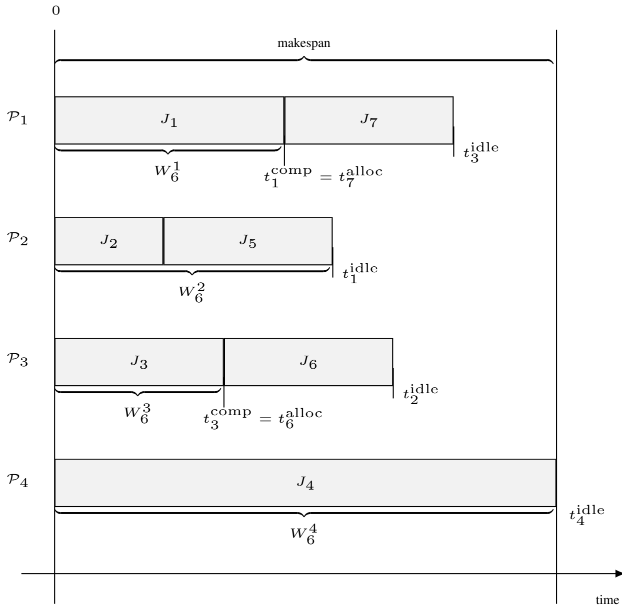

Figure 2. Illustration of our notations.

Lemma 3 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processor by a scheduler S such as the one described in Section 1. Then, an upper bound on the smallest instant t_i^{alloc} in the schedule at which the job J_i is assigned to a CPU is given by*

$$\hat{t}_i^{\text{alloc}} \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } m \geq n, \\ \frac{1}{m}\sum_{J_j >_S J_i} c_j & \text{otherwise.} \end{cases} \quad (1)$$

Proof *The proof is obvious for $m \geq n$. Otherwise, the proof is obtained by contradiction. Suppose that*

$$t_i^{\text{alloc}} > \hat{t}_i^{\text{alloc}}$$

Since we know that job J_i is assigned to a processor at time $t_i^{\text{alloc}} = \min_{k=1}^{m}\{W_i^k\}$ while scheduling the n jobs by S, we get

$$\min_{k=1}^{m}\{W_i^k\} > \hat{t}_i^{\text{alloc}}$$

and thus,

$$W_i^k > \hat{t}_i^{\text{alloc}} \quad \forall k = 1, 2, \ldots, m$$

By summing these m inequalities we obtain

$$\begin{array}{rlcl} & \sum_{k=1}^{m} W_i^k & > & m \cdot \hat{t}_i^{\text{alloc}} \\ \Leftrightarrow & \sum_{k=1}^{m} W_i^k & > & m \cdot \frac{1}{m}\sum_{J_j >_S J_i} c_j \\ \Leftrightarrow & \sum_{k=1}^{m} W_i^k & > & \sum_{J_j >_S J_i} c_j \end{array}$$

The above inequality means that the total amount of processing time associated to the processors after having scheduled every job with a higher priority than J_i is strictly larger than the amount of processing time of jobs with higher priority than J_i. This result being fallacious, the contradiction shows the property. ■

Lemma 4 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processor by a scheduler such as the one described in Section 1. Then for every $J_i \in J$, an upper bound on its completion time t_i^{comp} is given by:*

$$\hat{t}_i^{\text{comp}} \stackrel{\text{def}}{=} \hat{t}_i^{\text{alloc}} + c_i \quad (2)$$

where $\hat{t}_i^{\text{alloc}}$ is defined by Equation 1.

Proof *The proof is a direct consequence of the fact that there is no preemption during the schedule of the n jobs.* ■

Lemma 5 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ (assuming $c_1 \leq c_2 \leq \ldots \leq c_n$) that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processors by a scheduler S such as the one described in Section 1. Then, whatever the priority assignment of jobs, an upper bound on the makespan is given by:*

$$\text{upms}(J, m) \stackrel{\text{def}}{=} \begin{cases} c_n & \text{if } m \geq n \\ \frac{1}{m}\sum_{i=1}^{n} c_i + (1 - \frac{1}{m})\, c_n & \text{otherwise} \end{cases} \quad (3)$$

Proof *The proof is obvious for $m \geq n$. We will now prove the lemma for $m < n$. Using Lemma 3 and Lemma 4 we get for every job $J_i \in J$ that*

$$\hat{t}_i^{\text{comp}} \stackrel{\text{def}}{=} \frac{1}{m} \sum_{J_j >_S J_i} c_j + c_i$$

One can observe that the set $\cup_{k=1}^{n} \{J_k \mid J_k >_S J_i\} \subseteq J \setminus \{J_i\}$. Hence we have:

$$\hat{t}_i^{\text{comp}} \leq \frac{1}{m} \sum_{J \setminus \{J_i\}} c_j + c_i$$

Rewriting this yields:

$$\hat{t}_i^{\text{comp}} \leq \frac{1}{m} \sum_{j=1}^{n} c_j + (1 - \frac{1}{m}) c_i$$

We know that $c_i \leq c_n \ \forall i$. Applying this gives us:

$$\hat{t}_i^{\text{comp}} \leq \frac{1}{m} \sum_{j=1}^{n} c_j + (1 - \frac{1}{m}) c_n$$

which states the lemma. ■

The two following corollaries are direct consequences of Expression 3.

Corollary 1 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processors by a scheduler such as the one described in Section 1. Then, for any job $J_i \notin J$ of processing time $c_i \geq 0$ and ready at time 0, we have*

$$\text{upms}(J \cup \{J_i\}, m) \geq \text{upms}(J, m)$$

where $\text{upms}(J, m)$ is defined by Equation 3.

Notice that we know from Corollary 1 that aborting a rem-job upon a MCR can only reduce the largest makespan. As a result, only the case where every task must complete its current active job upon a MCR is studied in this paper, since it represents the *worst* scenario.

Corollary 2 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processors by a scheduler such as the one described in Section 1. Then, for any job $J_i \in J$ and for any job $J'_i \notin J$ ready at time 0 and such that $c'_i \geq c_i$, we have*

$$\text{upms}(J \setminus \{J_i\} \cup \{J'_i\}, m) \geq \text{upms}(J, m)$$

where $\text{upms}(J, m)$ is defined by Equation 3.

In the framework of our problem, we respectively know from Corollary 1 and 2 that the largest makespan is reached when (i) every task of τ^i has an active job at time $t_{\text{MCR}(j)}$ and (ii) every rem-job has a processing time equal to the worst-case execution time of its task. From Lemma 5, a *sufficient* schedulability condition may therefore be formalized as follows.

Validity test 1 *For any multi-mode real-time system τ, SM-MSO is* valid *provided:*

$$\text{upms}(\tau^i, m) \leq \min_{k=1}^{n_j} \{\mathcal{D}_k^j(M^i)\} \qquad \forall i, j \text{ with } i \neq j$$

where

$$\text{upms}(\tau^i, m) \stackrel{\text{def}}{=} \begin{cases} C_{\max} & \text{if } m \geq n_i \\ \frac{1}{m} \sum_{\tau_k^i \in \tau^i} C_k^i + (1 - \frac{1}{m}) C_{\max} & \text{otherwise} \end{cases}$$

and

$$C_{\max} \stackrel{\text{def}}{=} \max_{\tau_k^i \in \tau^i} \{C_k^i\}$$

4. The protocol AM-MSO

The main idea of this second protocol is to reduce the enablement delay applied to the new-mode tasks, by enabling them as soon as possible. On the contrary to SM-MSO, rem-jobs and new-mode tasks can be scheduled *simultaneously* during the transition phases.

The priorities of the rem-jobs are assigned according to the *old-mode* scheduler, whereas those of the new-mode tasks are assigned according to the *new-mode* scheduler. However during the whole transition phases, *every rem-job always get a higher priority than every new-mode task*. Formally, let M^{old} and M^{new} be the old- and new-mode respectively, let J_i and J_j be two active jobs during the mode change from M^{old} to M^{new}, and let S^{trans} be the scheduler used by AM-MSO during every mode transition. According to these notations we have

$$\begin{aligned} J_j >_{S^{\text{trans}}} J_i \quad \text{iff} \quad & (J_j \in M^{\text{old}} \wedge J_i \in M^{\text{new}}) \\ & \vee (J_j \in M^{\text{old}} \wedge J_i \in M^{\text{old}} \wedge J_j >_{S^{\text{old}}} J_i) \\ & \vee (J_j \in M^{\text{new}} \wedge J_i \in M^{\text{new}} \wedge J_j >_{S^{\text{new}}} J_i) \end{aligned}$$

Notice that in some cases, it could be more interesting to assign a higher priority to a new-task than to a rem-job (e.g., if the rem-job has a large deadline and the new-mode task has a small enablement deadline). However since this work is a first-step study, we did not implement this solution and we leave this problem open for future work.

AM-MSO works as follows: upon a $\text{MCR}(j)$, all the old-mode tasks are disabled and the rem-jobs continue to be scheduled by S^i (assuming that M^i is the old-mode). Whenever a processor completes a rem-job (say at time t)

and *there is no more waiting rem-jobs*, AM-MSO immediately enables some new-mode tasks; contrary to SM-MSO which waits for the completion of all the rem-jobs. In order to select the new-mode tasks to enable at time t, AM-MSO uses the following heuristic: it considers every disabled new-mode task in increasing order of their enablement deadline and it enables those which can be scheduled by S^j upon the current number of available CPUs (i.e., the number of CPUs which are not running a rem-job and which are therefore available for executing some new-mode tasks).

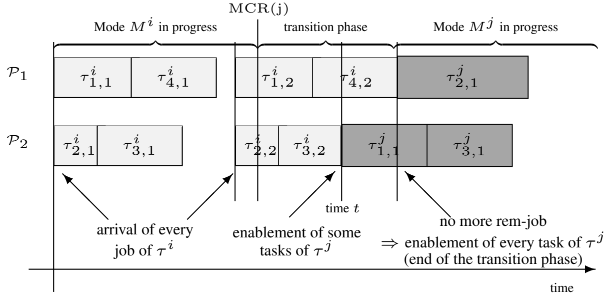

Figure 3. Illustration of a mode transition handled by AM-MSO.

Figure 3 depicts an example on a 2-processors platform (the same example as in Figure 1). At time t, $\mathcal{P}_2$ completes a rem-job and there is no more waiting rem-jobs to execute. Notice that time t is equivalent to time t_1^{idle}, defined as previously. AM-MSO therefore considers every disabled task of τ^j (in increasing order of their enablement deadline) and enables some of them in such a way that the resulting set of enabled tasks can be scheduled by S^j upon 1 processor (since at this time, only this processor $\mathcal{P}_2$ is available for executing some new-mode tasks).

We denote by $\text{CPU}^S(\tau^\ell)$ *the function returning a sufficient number of processors so that the set of tasks* τ^ℓ *can be scheduled by* S *without missing any deadline*. Unfortunately, no *efficient necessary and sufficient* schedulability test is known for most of multiprocessor scheduling algorithms in order to determine the *minimal* number of required CPUs to schedule a given task set (Theodore Baker has proposed [4] a necessary and sufficient schedulability test for arbitrary-deadline sporadic tasks scheduled by Global-EDF but its time-complexity is very high so only small systems can be tested). Fortunately, *sufficient* tests exist. For instance, examples of such functions can be found in [7] for the scheduling algorithms Global-DM and Global-EDF. The pseudo-code of the protocol AM-MSO is given by Algorithm 2.

In order to know if AM-MSO is valid for a given multi-mode system, we established a *validity algorithm* that indicates if all the job and enablement deadlines are met, while considering the worst-case scenario for every mode transition. This algorithm returns "true" only if the enablement deadlines will be always met, otherwise it returns "false". The main idea of this method is to simulate the behavior of Algorithm 2 for every mode transition, *while considering the largest instants at which the new-mode tasks could be enabled*.

Algorithm 2: AM-MSO

Input: M^i: the old mode; M^j: the new-mode
begin
    Assign priorities to the jobs according to S^{trans} ;
    Sort $\text{disabled}(\tau^j, t)$ by increasing order of enablement deadline ;

    At job (say J_k) completion time t :
    if $J_k \in \tau^i$ **and** $\text{wait}(\tau^i, t) = \phi$ **then**
        foreach $\tau_r^j \in \text{disabled}(\tau^j, t)$ **do**
            if $\text{CPU}^{S^j}(\text{enabled}(\tau^j, t) \cup \{\tau_r^j\}) \leq m - \#(\text{run}(\tau^i, t))$ **then**
                enable τ_r^j ;
        if $\text{active}(\tau^i, t) = \phi$ **then** enter mode M^j;
end

Since S_{trans} always assigns higher priorities to the rem-jobs than to new-mode jobs, the arrivals of the new-mode jobs do not influence the schedule of the rem-jobs. As a result, the largest enablement instants of the new-mode tasks can be determined by only considering the schedule of the rem-jobs. Actually, *these largest instants correspond to the largest instants* t_k^{idle} *as defined previously*. Hence, we establish in Lemma 7 an upper bound on the smallest instants t_k^{idle} in the transition phase at which at least k processors are idle (while ignoring the new-mode job arrivals). Finally, we prove the correctness our the validity algorithm in Lemma 8.

Lemma 6 *Let* $J = \{J_1, J_2, \ldots, J_n\}$ *be a set of* n *jobs with processing times* $c_1, c_2, \ldots, c_n$ *(assuming* $c_1 \leq c_2 \leq \ldots \leq c_n$*) that are ready for execution at time* 0*. Suppose that these jobs are scheduled upon* m *identical processors (with* $m < n$*) by a scheduler* S *such as the one described in Section 1. Then, whatever the priority assignment of jobs, we have* $\forall j, k \in [1, m]$ *such that* $j < k$*:*

$$t_j^{\text{idle}} \geq t_k^{\text{idle}} - c_{n-m+k}$$

Proof *The proof is obtained by contradiction. Suppose that there are* j *and* k *in* $[1, m]$ *such that* $j < k$ *and*

$$t_j^{\text{idle}} < t_k^{\text{idle}} - c_{n-m+k}$$

By definition of the instants t_i^{idle}*'s, we know that* $t_i^{\text{idle}} \leq t_{i+1}^{\text{idle}}$ $\forall i \in [1, m]$*. Therefore, we know from the above in-*

equality that the following $(m-k+1)$ inequalities hold:

$$\begin{aligned} t_j^{\text{idle}} &< t_k^{\text{idle}} - c_{n-m+k} \\ \Rightarrow t_j^{\text{idle}} &< t_{k+1}^{\text{idle}} - c_{n-m+k} \\ \Rightarrow t_j^{\text{idle}} &< t_{k+2}^{\text{idle}} - c_{n-m+k} \\ \Rightarrow t_j^{\text{idle}} &< \ldots \\ \Rightarrow t_j^{\text{idle}} &< t_m^{\text{idle}} - c_{n-m+k} \end{aligned}$$

According to the specifications of S, we know that a processor (say $\mathcal{P}_i$) gets idle at time t_j^{idle} only if there is no more waiting job at time t_j^{idle}, i.e., each one of the $(m-j)$ remaining active jobs is running *on another processor than $\mathcal{P}_i$. However, the above $(m-k+1)$ inequalities suggest that there remain $(m-k+1)$ running jobs at time t_j^{idle} with a remaining processing time larger than c_{n-m+k}. This leads to a contradiction since there can only be at most $(m-k)$ jobs with a processing time larger than c_{n-m+k} (since $c_1 \leq c_2 \leq \ldots \leq c_n$). The property follows.* ■

Lemma 7 *Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs with processing times $c_1, c_2, \ldots, c_n$ (assuming $c_1 \leq c_2 \leq \ldots \leq c_n$) that are ready for execution at time 0. Suppose that these jobs are scheduled upon m identical processors by a scheduler such as the one described in Section 1. Then, whatever the priority assignment of jobs, an upper bound on the smallest instant t_k^{idle} in the schedule such that at least k CPUs (with $k = 1, \ldots, m$) are idle is given by $t(J,m,k)$ where*

$$t(J,m,k) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } (n \leq m) \wedge (m-n \geq k) \\ c_{k-m+n} & \text{if } (n \leq m) \wedge (m-n < k) \end{cases} \quad (4)$$

and $t(J,m,k) \stackrel{\text{def}}{=}$

$$\max_{i=0}^{n-m+k-1} \left\{ \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+m-k+1} c_j}{m} + \frac{\sum_{j=i+1}^{i+m-k+1} c_j}{m-k+1} \right\}$$

otherwise $(n > m)$.

Proof *Both cases where $n \leq m$ are obvious. Otherwise, we prove the lemma by contradiction. Let t_k^{idle} denote the smallest instant in the schedule such that at least k CPUs are idle and suppose that there is a $k \in [1,m]$ such that $t_k^{\text{idle}} > t(J,m,k)$. The following properties hold:*

(a) $\forall\, i > k$: $t_i^{\text{idle}} \geq t_k^{\text{idle}}$ *(by definition of t_i^{idle}'s).*

(b) $\forall\, i < k$: $t_i^{\text{idle}} \geq t_k^{\text{idle}} - c_{n-m+k}$ *(from Lemma 6).*

Obviously, we know that:

$$\sum_{i=1}^{m} t_i^{\text{idle}} = \sum_{i=1}^{k-1} t_i^{\text{idle}} + t_k^{\text{idle}} + \sum_{i=k+1}^{m} t_i^{\text{idle}}$$

By applying properties (a) and (b) on the right-hand side, we get

$$\begin{aligned} \sum_{i=1}^{m} t_i^{\text{idle}} &\geq \sum_{i=1}^{k-1} (t_k^{\text{idle}} - c_{n-m+k}) + t_k^{\text{idle}} + \sum_{i=k+1}^{m} t_k^{\text{idle}} \\ &\geq (k-1)(t_k^{\text{idle}} - c_{n-m+k}) + t_k^{\text{idle}} + (m-k) t_k^{\text{idle}} \\ &\geq m\, t_k^{\text{idle}} - (k-1) c_{n-m+k} \end{aligned}$$

Since by assumption $t_k^{\text{idle}} > t(J,m,k)$ we can replace t_k^{idle} by $t(J,m,k)$ in the above inequality, which leads to

$$\sum_{i=1}^{m} t_i^{\text{idle}} > m\, t(J,m,k) - (k-1) c_{n-m+k}$$

Let ℓ be a value of i which maximizes $t(J,m,k)$ in Expression 4. The above inequality becomes

$$\begin{aligned} \sum_{i=1}^{m} t_i^{\text{idle}} > \; & m \left(\frac{\sum_{j=1}^{n} c_j - \sum_{j=\ell+1}^{\ell+m-k+1} c_j}{m} + \frac{\sum_{j=\ell+1}^{\ell+m-k+1} c_j}{m-k+1} \right) \\ & -(k-1) c_{n-m+k} \end{aligned}$$

Rewriting this yields:

$$\begin{aligned} \sum_{i=1}^{m} t_i^{\text{idle}} &> \sum_{j=1}^{n} c_j - \sum_{j=\ell+1}^{\ell+m-k+1} c_j + \frac{m}{m-k+1} \sum_{j=\ell+1}^{\ell+m-k+1} c_j \\ &\quad -(k-1) c_{n-m+k} \\ &> \sum_{j=1}^{n} c_j + \left(\frac{m}{m-k+1} - 1\right) \sum_{j=\ell+1}^{\ell+m-k+1} c_j - (k-1) c_{n-m+k} \\ &> \sum_{j=1}^{n} c_j + \frac{k-1}{m-k+1} \sum_{j=\ell+1}^{\ell+m-k+1} c_j - (k-1) c_{n-m+k} \quad (5) \end{aligned}$$

By definition of ℓ, we know from Expression 4 that $\forall i = 0, \ldots, n-m+k-1$ we have:

$$\begin{aligned} & \frac{\sum_{j=1}^{n} c_j - \sum_{j=\ell+1}^{\ell+m-k+1} c_j}{m} + \frac{\sum_{j=\ell+1}^{\ell+m-k+1} c_j}{m-k+1} \\ \geq \; & \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+m-k+1} c_j}{m} + \frac{\sum_{j=i+1}^{i+m-k+1} c_j}{m-k+1} \end{aligned}$$

Rewriting this yields:

$$\sum_{j=\ell+1}^{\ell+m-k+1} c_j \geq \sum_{j=i+1}^{i+m-k+1} c_j$$

and consequently,

$$\begin{aligned} \frac{(k-1)}{m-k+1} \sum_{j=\ell+1}^{\ell+m-k+1} c_j &\geq \frac{(k-1)}{m-k+1} \sum_{j=i+1}^{i+m-k+1} c_j \\ &\geq \frac{(k-1)}{m-k+1} (m-k+1) c_{i+1} \\ &\geq (k-1) c_{i+1} \end{aligned}$$

$\forall i = 0, \ldots, n-m+k-1$. *Replacing i by $n-m+k-1$ in the above inequality leads to*

$$\frac{(k-1)}{m-k+1} \sum_{j=\ell+1}^{\ell+m-k+1} c_j \geq (k-1)c_{n-m+k}$$

Therefore, rewriting Inequality 5 yields:

$$\sum_{i=1}^{m} t_i^{\text{idle}} > \sum_{j=1}^{n} c_j$$

which leads to a contradiction since we know that $\sum_{i=1}^{m} t_i^{\text{idle}} = \sum_{i=1}^{n} c_i$. The lemma follows. ■

Important notes.

- Expression 4 can be considered to be a generalization of Expression 3. Indeed, the makespan is the particular case where $k = m$ in Expression 4, leading to

  $$t(J, m, m) \stackrel{\text{def}}{=} \begin{cases} c_n & \text{if } n \leq m \\ \max_{i=0}^{n-1} \left\{ \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+1} c_j}{m} + \sum_{j=i+1}^{i+1} c_j \right\} & otherwise \end{cases}$$

  With $n > m$, it is easy to show (using Lemma 9 in Appendix) that the above expression is maximized for $i = n-1$. In that case, we get

  $$\begin{aligned} & \max_{i=0}^{n-1} \left\{ \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+1} c_j}{m} + \sum_{j=i+1}^{i+1} c_j \right\} \\ = & \frac{\sum_{j=1}^{n} c_j - c_n}{m} + c_n \\ = & \frac{\sum_{j=1}^{n} c_j}{m} + (1 - \frac{1}{m})c_n \end{aligned}$$

  which is equivalent to Expression 3.

- Notice that both Expressions 3 and 4 *are tight*. In some particular cases, these upper bounds can even be reached. For instance, the maximal makespan produced by the set of jobs $J = \{J_1, J_2, J_3, J_4, J_5\}$ with respective processing times 2, 3, 3, 4, 8 on a 2-processors platform is 14 (e.g., by choosing the priority assignment $J_1 > J_2 > J_4 > J_3 > J_5$); and from Expression 3 we get

  $$\text{upms}(J, 2) = \frac{1}{2}(2+3+3+4+8) + (1 - \frac{1}{2}) \cdot 8 = 14$$

  Similar examples can easily be found concerning Expression 4.

The validity algorithm for AM-MSO is given by Algorithm 3, where the largest instants at which the new-mode tasks are enabled are determined in line 7 thanks to Expression 4.

Algorithm 3: Validity algorithm for AM-MSO

```
   Output: boolean schedulable (or not)
1  begin
2      foreach i, j ∈ [1, x] such as i ≠ j do
3          τ^disabled ← τ^j ;
4          sort(τ^disabled) by increasing order of enablement
           deadline ;
5          τ^enabled ← φ ;
6          for (k = 1; k ≤ m; k++) do
7              t_k^idle ← t(τ^i, m, k) ;
8              foreach τ_r ∈ τ^disabled do
9                  if (D_k^j(M^i) < t_k^idle) then return false ;
10                 if CPU^{S^j}(τ^enabled ∪ τ_r) ≤ k then
11                     τ^enabled ← τ^enabled ∪ {τ_r} ;
12                     τ^disabled ← τ^disabled \ {τ_r} ;

13     return true;
14 end
```

Lemma 8 *Algorithm 3 provides a sufficient validity test.*

Proof *This proof is a direct consequence of the fact that every instant at which some new-mode tasks are enabled in Algorithm 3 (cf. line 7) is large as possible. Indeed, let M^i and M^j be two operating modes of the given system, and let t_r^{idle} be the smallest instant such that r CPUs are idle during the transition from M^i to M^j in the system execution. Suppose that at this time t_r^{idle}, a new-mode task τ_k^j misses its enablement deadline $\mathcal{D}_k^j(M^i)$ (i.e., $t_r^{\text{idle}} > \mathcal{D}_k^j(M^i)$). Since $t_r^{\text{idle}} < t(\tau^i, m, r)$ from Lemma 7, τ_k^j also misses its enablement deadline while executing Algorithm 3.* ■

5. Conclusion and future work

In this paper, we proposed two protocols which handle every mode transition during the execution of a sporadic multi-mode real-time system. The first one (called MS-MSO) is a synchronous protocol in the sense that the tasks of the old- and new-mode are not scheduled simultaneously. On the other hand, our second protocol (called AM-MSO) allows old- and new-mode tasks to be scheduled together. Both protocols can be considered to be a generalization to the multiprocessor case of the "Minimal Single Offset" (MSO) protocol proposed in [13]. For both of them, we established a schedulability test which allows the system designer to predict whether the given system can meet all the expected requirements. This study led us to extend the theory about the makespan determination problem, by establishing an upper bound on the maximal makespan that can be produced by a given set of jobs.

In our future work, we aim to take into account mode-independent tasks, i.e., tasks whose the periodic (or sporadic) activation pattern is not affected by the mode changes. Moreover, instead of scheduling the rem-jobs by

using the scheduler of the old-mode during the transition, it could be better, in term of the enablement delays applied to the new-mode tasks, to propose a *dedicated priority assignment* which meets the deadline of every rem-job, *while* minimizing the makespan. To the best of our knowledge, the problem of minimizing the makespan while meeting job deadlines together is not yet addressed in the literature and remains for future work.

References

[1] B. Andersson. *Static-priority scheduling on multiprocessors*. PhD thesis, Chalmers University of Technology, 2003.

[2] C. M. Bailey. Hard real-time operating system kernel. investigation of mode change. Technical report, Task 14 Deliverable on ESTSEC Contract 9198/90/NL/SF, British Aerospace Systems Ltd., 1993.

[3] T. Baker. Multiprocessor EDF and deadline monotonic schedulability analisys. In *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, pages 120–129, December 2003.

[4] T. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *Proceedings of OPODIS 2007, The 10th International Conference on Principles of Distributed Systems*, number 4878, pages 62–75, Guadeloupe, December 2007. Springer Lecture Notes in Computer Science.

[5] G. J. Fohler. *Flexibility in statically scheduled hard real-time systems*. PhD thesis, Technische Universität Wien, 1994.

[6] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on uniform multiprocessors. *Real-Time Systems*, 25:187–205, 2003.

[7] J. Goossens, D. Milojevic, and V. Nelis. Power-aware real-time scheduling upon dual CPU type multiprocessor platforms. In T. Baker, A. Bui, and S. Tixeuil, editors, *12th International Conference On Principles Of Distributed Systems (OPODIS)*, number 5401 in LNCS, pages 388–407, Luxor, Egypt, December 2008. Springer.

[8] R. Ha and J. W. S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 162–171, 1994.

[9] F. Jahanian, R. Lee, and A. Mok. Semantics of modechart in real time logic. In *Proceedings of the 21st Hawaii International Conference on Systems Sciences*, pages 479–489, 1988.

[10] V. Nelis and J. Goossens. Mode change protocol for multi-mode real-time systems upon identical multiprocessors. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (Work in Progress session - RTSS08-WiP)*, pages 9–12, Barcelona Spain, December 2008.

[11] P. Pedro. *Schedulability of mode changes in flexible real-time distributed systems*. PhD thesis, University of York, Department of Computer Science, 1999.

[12] 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*, pages 172–179, 1998.

[13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, March 2004.

[14] K. Tindell and A. Alonso. A very simple protocol for mode changes in priority preemptive systems. Technical report, Universidad Politécnica de Madrid, 1996.

[15] K. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proceedings Real-Time Systems Symposium*, pages 100–109, Phoenix, Arizona, 1992.

Appendix

Lemma 9 *If $c_1 \leq c_2 \leq \ldots \leq c_n$ then*

$$\max_{i=0}^{n-1} \left\{ \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+1} c_j}{m} + \sum_{j=i+1}^{i+1} c_j \right\}$$

is maximal for $i = n - 1$.

Proof *Consider the function*

$$\max_{i=0}^{n-1} \left\{ \frac{\sum_{j=1}^{n} c_j - \sum_{j=i+1}^{i+1} c_j}{m} + \sum_{j=i+1}^{i+1} c_j \right\}$$

We can rewrite this function to

$$\max_{i=0}^{n-1} \left\{ \frac{\sum_{j=1}^{n} c_j}{m} + c_{i+1} - \frac{c_{i+1}}{m} \right\}$$

Consider the following term (which is dependent on i):

$$\max_{i=0}^{n-1} \left\{ (1 - \frac{1}{m}) c_{i+1} \right\}$$

Since we know that $c_1 \leq c_2 \leq \ldots \leq c_n$, the above expression is maximal for $i = n - 1$. This is the statement of the lemma.