Technical Report

A Synchronous Transition Protocol with Periodicity for Global Scheduling of Multimode Real-Time Systems on Multiprocessors

Vincent Nelis, Björn Andersson and Joel Goossens

HURRAY-TR-091101

Version: 0

Date: 11-01-2009

A Synchronous Transition Protocol with Periodicity for Global Scheduling of Multimode Real-Time Systems on Multiprocessors

Vincent Nelis, Björn Andersson and Joel Goossens

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

We consider the global scheduling problem of multimode real-time systems upon identical multiprocessor platforms. During the execution of a multimode system, the system can change from one mode to another such that the current task set is replaced with a new task set. Thereby, 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. In this paper, we extend the synchronous transition protocol SM-MSO in order to take into account mode-independent tasks [1], i.e., tasks of which the execution pattern must not be jeopardized by the mode changes.

A Synchronous Transition Protocol with Periodicity for Global Scheduling of Multimode Real-Time Systems on Multiprocessors

Vincent Nelis[1]
Computer Science Department
Université Libre de Bruxelles (U.L.B.)
Brussels, Belgium
vnelis@ulb.ac.be

Björn Andersson
CISTER Research unit
Polytechnic Institute of Porto
Porto, Portugal
bandersson@dei.isep.ipp.pt

Joël Goossens
Computer Science Department
Université Libre de Bruxelles (U.L.B.)
Brussels, Belgium
joel.goossens@ulb.ac.be

***Abstract*—We consider the global scheduling problem of multimode real-time systems upon identical multiprocessor platforms. During the execution of a multimode system, the system can change from one mode to another such that the current task set is replaced with a new task set. Thereby, 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. In this paper, we extend the synchronous transition protocol SM-MSO in order to take into account mode-independent tasks [1], i.e., tasks of which the execution pattern must not be jeopardized by the mode changes.**

***Keywords*-multimode scheduling; multiprocessor scheduling; real-time scheduling;**

I. Introduction

Hard real-time systems require both functionally correct executions and *results that are produced on time*. 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 as 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 *multimode* 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 new-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.

[1]Supported by the Belgian National Science Foundation (F.N.R.S.) under a F.R.I.A. grant.

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., [2], [3] 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 for instance). 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 jeopardized by the mode change in progress (such tasks are typically daemon functionalities). In the literature (see [4] for instance), a transition 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*.

Related work. Numerous scheduling protocols have already been proposed in the *uni*processor case to ensure the transition between modes (see [4] for a survey of the literature about this uniprocessor problem). Targeting *multi*processor environments, previous work [1] proposed two protocols *without periodicity*: a synchronous protocol called SM-MSO and an asynchronous one called AM-MSO. To the best of our knowledge, these two protocols are the only ones to be proposed for the multimode scheduling problem upon *multi*processor platforms.

This research. In this paper, we extend the protocols SM-MSO proposed in [1] to make it "*with periodicity*". We take into account the mode-independent tasks and we rewrite the validity test of SM-MSO in order to ensure that all the requirements are met during every mode transition.

However this research is a first step since we only consider synchronous protocols. Notice that in this document, we assume that every operating mode of the system is scheduled by a *global*, *preemptive*, *work-conserving* and *fixed job-level priority* scheduling algorithm.

II. Model of computation

A. System and platform specifications

We consider multiprocessor platforms composed of a known and fixed number m of *identical* processors $\{P_1, P_2, \ldots, P_m\}$ upon which a multimode 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 multimode 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 from 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 as 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 absolute deadline denoted $d_{i,j}^k \stackrel{\text{def}}{=} a_{i,j}^k + d_i^k$. 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. Hereafter, $\mathrm{active}(\tau^k, t)$ denotes the subsets of active tasks of τ^k at time 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 denote by $\mathrm{enabled}(\tau^k, t)$ and $\mathrm{disabled}(\tau^k, t)$ the subsets of *enabled* and *disabled* tasks of τ^k at time t, respectively.

During any mode change from mode M^i to mode M^j, we denote by $\tau_{i,j}^{\mathrm{mit}}$ the set of mode-independent tasks that belong to both modes M_i and M_j (i.e., $\tau_{i,j}^{\mathrm{mit}} = \tau^i \bigcap \tau^j$). These tasks are assumed to be sporadic and constrained-deadline. Each such task sporadically generates jobs during the entire mode transition and its sporadic execution pattern must not be influenced by the mode change in progress.

B. Scheduler specifications

We consider in this study that the scheduler is *global*, *preemptive*, *work-conserving* and it assigns *fixed job-level priority* according to the usual interpretations (see [1] for formal definitions). Notice that *Global Deadline Monotonic* and *Global Earliest Deadline First* [5] are some examples of such scheduling algorithms. We assume that every mode M^k of the system uses its own scheduling algorithm noted S^k and the tasks set τ^k of 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.

C. Mode transition specifications

While the system is running in a mode M^i (i.e., the old-mode), 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 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 that are not mode-independent (i.e., the tasks of $\tau^i \setminus \tau_{i,j}^{\mathrm{mit}}$), *hence preventing new job arrivals from these tasks*. At time $t_{\mathrm{MCR}(j)}$ the active jobs of these disabled tasks, henceforth called the *rem-jobs* (for *rem*aining *jobs*), may have two distinct behaviors: either *they can be aborted* or *they must complete their execution*. In previous work [1] we showed that aborting rem-jobs immediately do not jeopardize the system schedulability, that scheduling problem is straightforward. In this research we consider the more interesting case where all rem-jobs *must complete their execution*.

By assumption, we know that the set τ^j of new-mode tasks can be scheduled upon the m processors without missing any deadline. However, the rem-jobs may cause an overload if the tasks of τ^j are immediately enabled upon the mode change request $\mathrm{MCR}(j)$. As a result, transition protocols usually have to delay the enablement of these new-mode tasks until it is safe to do that. 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 enabled not after time $t_{\mathrm{MCR}(j)} + \mathcal{D}_k^j(M^i)$. The goal of a transition protocol is therefore to *(i) complete every rem-job, (ii) schedule every mode-independent tasks and (iii) 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.

III. The protocol SM-MSO

In this section, we present how the *synchronous* protocol SM-MSO proposed in [1] can be extended while considering mode-independent tasks. Notice that we do not consider asynchronous protocols in this document. The main idea of this extension is the following: *upon a* $\mathrm{MCR}(j)$*, every non-mode-independent task of the old-mode (say* M^i*) is disabled and both the rem-jobs and the mode-independent tasks continue to be scheduled by* S^i *upon the* m *processors. When* all *the rem-jobs are completed, all the non-mode-independent new-mode tasks (i.e., the tasks of* $\tau^j \setminus \tau^{\mathrm{mit}}_{i,j}$*) are simultaneously enabled*. From this instant, both the new-mode tasks and the mode-independent tasks are scheduled by S^j upon the m processors. Figure 1 depicts an example with a 2-processors platform. Both modes M^i and M^j contain 3 tasks and 1 mode-independent tasks ($\tau^{\mathrm{mit}}_{i,j} = \{\tau_1\}$), where the light gray, dark gray and black boxes are the old-mode, new-mode, and mode-independent tasks, respectively. Algorithm 1 gives the pseudo-code of this protocol.

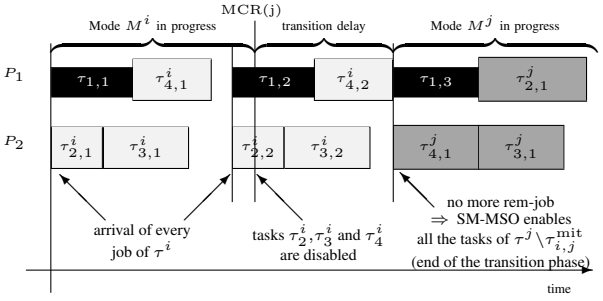

Figure 1. Illustration of a mode transition handled by SM-MSO.

Algorithm 1: SM-MSO (revisited)

Input: $t_{\mathrm{MCR}(j)}$: current time; M^i: the old mode; M^j: the new-mode
begin
  Disable all the tasks $\tau_k \in \tau^i \setminus \tau^{\mathrm{mit}}_{i,j}$;
  Schedule all the jobs of τ^i according to S^i ;
  At job completion time t :
  if $\mathrm{active}(\tau^i \setminus \tau^{\mathrm{mit}}_{i,j}, t) = \phi$ **then**
    enable all the tasks of $\tau^j \setminus \tau^{\mathrm{mit}}_{i,j}$;
    enter mode M^j ;
end

It is well-known that proposing a new scheduling algorithm requires to also provide an associated *schedulability test*, i.e., a condition based on the tasks and platform characteristics which indicates *a priori* whether the given system will meet *every job deadline*. In a similar way, proposing a new mode transition protocol requires to also provide an associated *validity test*, i.e., a condition based on the tasks and platform characteristics that indicates *a priori* whether the given system will meet *every job and enablement deadline during every transition between every pair of operating modes of the system*. In the following, we focus on designing a *validity test* for the protocol SM-MSO. First, Lemma 1 establishes an upper bound on the completion time of any rem-job during any mode transition, while considering the interference due to the execution of the mode-independent tasks. Then in Corollary 1, we determine the *largest makespan*, where the makespan is defined as follows.

Definition 1 (makespan): 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. Let τ^{mit} be a set of sporadic constrained-deadline tasks that are scheduled during the schedule of J. Suppose that τ^{mit} and the n jobs of J are scheduled upon m identical processors by a global, preemptive, work-conserving and fixed-job priority scheduler. We define the makespan as *the earliest time in the schedule at which all the jobs of* J *are completed*.

By using a similar proof as Lemma 2 of [1], we can easily show that every rem-job and every job generated by any mode-independent tasks always meets its absolute deadline $d_{i,j}$ while using SM-MSO during the transition phases. Thereby, for a given multimode real-time system, the protocol SM-MSO will comply with every expected requirement if all the *enablement deadlines* $\mathcal{D}^j_k(M^i)$ are also met during every mode transition, i.e., if *the makespan is not larger than the minimal enablement deadline of the non-mode-independent new-mode tasks*. This upper bound on the makespan thus allows us to design a *sufficient validity test* that indicates, a priori, if all the enablement deadlines will be met during all possible mode changes. Notice that we do not assume specific scheduling algorithms in this document. Every result proposed here hold for any global, preemptive, work-conserving and fixed-job priority scheduler.

Definition 2 (processed work): At any time t in any schedule, the processed work $\mathrm{w}_k(t)$ denotes the amount of work executed on processor P_k in the time interval $[0, t]$.

Lemma 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. Let τ^{mit} be a set of sporadic constrained-deadline tasks that are scheduled during the schedule of J. Suppose that τ^{mit} and the n jobs of J are scheduled upon m identical processors by a global, preemptive, work-conserving and fixed-job priority scheduler. Then, an upper

bound $\hat{R}_i$ on the time at which job $J_i \in J$ completes is given by the first fixed point of the following iterative process:

$$\begin{aligned} \hat{R}_i^{(0)} &\leftarrow \frac{1}{m}\sum_{\substack{j=1\\ j\neq i}}^{n} c_j + c_i \\ \hat{R}_i^{(k+1)} &\leftarrow \frac{1}{m}\left(\sum_{\substack{j=1\\ j\neq i}}^{n} c_j + \sum_{\tau_j \in \tau^{\text{mit}}} W(\tau_j, \hat{R}_i^{(k)})\right) + c_i \quad (1) \end{aligned}$$

where $W(\tau_j, t)$ denotes an upper bound on the amount of work that can be generated by the task τ_j in the time interval $[0, t]$. Notice that in [6], the authors showed that

$$W(\tau_j, t) = N_j(t)C_j + \min(C_j, t + D_j - C_j - N_j(t)T_j)$$

where $N_j(t) \stackrel{\text{def}}{=} \left\lfloor \frac{t+D_j-C_j}{T_j} \right\rfloor$

Proof: Suppose that the job J_i completes at time R_i on processor P_k. Since the scheduler is work-conserving, the processed work $\mathrm{w}_k(R_i)$ on P_k is R_i and the processed work $\mathrm{w}_j(R_i)$ on the other processors P_j (with $j \neq k$) is at least $R_i - c_i$. Formally we have

$$\begin{cases} \mathrm{w}_j(R_i) = R_i & \text{if } j = k \\ \mathrm{w}_j(R_i) \geq R_i - c_i & \forall j = 1, \ldots, m \text{ and } j \neq k \end{cases}$$

By summing these m expressions, we get

$$\sum_{j=1}^{m} \mathrm{w}_j(R_i) \geq mR_i - (m-1)c_i \quad (2)$$

where $\sum_{j=1}^{m} \mathrm{w}_j(R_i)$ denotes the amount of work executed on the m processors from time 0 to time R_i. On the other hand, we know that this cumulative processed work cannot be larger than the amount of requested work in the system, i.e.,

$$\sum_{j=1}^{m} \mathrm{w}_j(R_i) \leq \sum_{j=1}^{n} c_j + \sum_{\tau_j \in \tau^{\text{mit}}} W(\tau_j, R_i) \quad (3)$$

By using Inequalities 2 and 3, we get

$$mR_i - (m-1)c_i \leq \sum_{j=1}^{n} c_j + \sum_{\tau_j \in \tau^{\text{mit}}} W(\tau_j, R_i)$$

Rewriting this yields

$$R_i \leq \frac{1}{m}\left(\sum_{\substack{j=1\\ j\neq i}}^{n} c_j + \sum_{\tau_j \in \tau^{\text{mit}}} W(\tau_j, R_i)\right) + c_i$$

As a result, R_i is upper bounded by $\hat{R}_i$ defined as in Expression 1.

■

Corollary 1: Assuming the same notations as in Lemma 1, an upper bound on the makespan is given by

$$\widehat{\mathrm{ms}}(J, \tau^{\text{mit}}, m) \stackrel{\text{def}}{=} \max_{i=1}^{n}\{\hat{R}_i\} \quad (4)$$

The proof is a direct consequence of Lemma 1. As a result, a *sufficient* validity condition may be formalized as follows.

Validity test 1: For any multimode real-time system τ, SM-MSO meets every job and enablement deadline during every transition between every pair of operating modes of τ if, $\forall M^i, M^j$ with $M^i \neq M^j$,

$$\widehat{\mathrm{ms}}(J, \tau_{i,j}^{\text{mit}}, m) \leq \min_{\tau_k^j \in \tau^j \setminus \tau_{i,j}^{\text{mit}}} \{\mathcal{D}_k^j(M^i)\}$$

where J is the set of jobs composed of one job issued from each task of $\tau^i \setminus \tau_{i,j}^{\text{mit}}$.

IV. Conclusion and future work

In this paper, we extended the *synchronous* protocol SM-MSO proposed in [1] in order to take into account the mode-independent tasks during the execution of sporadic multimode real-time systems on multiprocessor platforms. Moreover, we established a validity test which allows the system designer to predict whether the given system can meet all the expected requirements during every mode transition. In our future work, we aim to design an *asynchronous* protocol with the consideration of mode-independent tasks.

References

[1] V. Nelis, 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*, Dublin, Ireland, July 2009, pp. 151–160.

[2] G. J. Fohler, "Flexibility in statically scheduled hard real-time systems," Ph.D. dissertation, Technische Universität Wien, 1994.

[3] 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*, 1988, pp. 479–489.

[4] 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, March 2004.

[5] T. Baker, "Multiprocessor EDF and deadline monotonic schedulability analysis," in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, December 2003, pp. 120–129.

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