

# Real-Time Reconfigurable Scheduling of Sporadic Tasks

Hamza Gharsellaoui<sup>1,2(✉)</sup> and Samir Ben Ahmed<sup>3</sup>

<sup>1</sup> Higher School of Technology and Computer Science,  
University of Carthage, Carthage, Tunisia

<sup>2</sup> Al-Jouf College of Technology, Technical and Vocational Training Corporation,  
Al-Jouf, Kingdom of Saudi Arabia  
`gharsellaoui.hamza@gmail.com`

<sup>3</sup> Faculty of Mathematical, Physical and Natural Sciences of Tunis, FST,  
University of Tunis El Manar, Tunis, Tunisia  
`Samir.benahmed@fst.rnu.tn`

**Abstract.** This book chapter deals with the problem of scheduling multiprocessor real-time tasks by an optimal EDF-based scheduling algorithm. Two forms of automatic reconfigurations which are assumed to be applied at run-time: Addition-Removal of tasks or just modifications of their temporal parameters: WCET and/or deadlines. Nevertheless, when such a scenario is applied to save the system at the occurrence of hardware-software faults, or to improve its performance, some real-time properties can be violated at run-time. We define an Intelligent Agent that automatically checks the system's feasibility after any reconfiguration scenario was applied on a multiprocessor embedded system. Indeed, if the system is unfeasible, then the Intelligent Agent dynamically provides precious technical solutions for users to send sporadic tasks to idle times, by modifying the deadlines of tasks, the worst case execution times (WCETs), the activation time, by tolerating some non critical tasks, by sending some tasks from their current processors to be scheduled in other processors, or in the worst case by removing some soft tasks according to predefined heuristic. We implement the agent to support these services.

**Keywords:** Real-time reconfigurable sporadic tasks · Intelligent agent · Multiprocessor systems automatic reconfigurations · EDF-based scheduling algorithm

## 1 Introduction

Nowadays, the new generations of embedded control systems are addressing new criteria such as flexibility and agility [1]. For these reasons, there is a need to develop tools, methodologies in embedded software engineering and dynamic reconfigurable embedded control systems as an independent discipline. Each system is a subset of tasks. Each task is characterized by its worst case execution times (WCETs)  $C_i^{p,\psi_h}$ , an offset (release time)  $a_i^{p,\psi_h}$ , a period  $T_i^{p,\psi_h}$  and a deadline  $D_i^{p,\psi_h}$  for each reconfiguration scenario  $\psi_h$ , ( $h \in 1..M$ , we assume that we

have  $M$  reconfiguration scenarios) and on each processor  $p$ , ( $p \in 1..K$ , we assume that we have  $K$  identical processors numbered from 1 to  $K$ ), and  $n$  real-time tasks numbered from 1 to  $n$  that composed a feasible subset of tasks entitled  $\xi_{old}$ . The general goal of this work is to be reassured that any reconfiguration scenario  $\psi_h$  changing the implementation of the embedded system does not violate real-time constraints: i.e. the system is feasible and meets real-time constraints even if we change its implementation and to correctly allow the minimization of the response time of this system after any reconfiguration scenario [1]. To obtain this optimization (minimization of response time), we propose an intelligent agent-based architecture in which a software agent is deployed to dynamically adapt the system to its environment by applying reconfiguration scenarios. A reconfiguration scenario  $\psi_h$  means the addition, removal or update of tasks in order to save the whole system on the occurrence of hardware/software faults, or also to improve its performance when random disturbances happen at run-time. Sporadic task is described by minimum interarrival time  $P_i^{p,\psi_h}$  which is assumed to be equal to its relative deadline  $D_i^{p,\psi_h}$ , and a worst-case execution time (WCET)  $C_i^{p,\psi_h}$  for each reconfiguration scenario  $\psi_h$  and on each processor  $p$ . A random disturbance is defined in the current work as any random internal or external event allowing the addition of tasks that we assume sporadic or removal of sporadic/periodic tasks to adapt the system's behavior. Indeed, a hard real-time system typically has a mixture of off-line and on-line workloads and assumed to be feasible before any reconfiguration scenario  $\psi_h$ . The off-line requests support the normal functions of the system while the on-line requests are sporadic tasks to handle external events such as operator commands and recovery actions which are usually unpredictable. For this reason and in this original work, we propose a new optimal scheduling algorithm based on the dynamic priorities scheduling Earliest Deadline First (EDF) algorithm principles on each processor  $p$  and for each dynamic reconfiguration scenario  $\psi_h$  in order to obtain the feasibility of the system at run-time, meeting real-time constraints and for the optimization of the response time of this system. Indeed, for independent, preemptable tasks, on a uni-processor, EDF is optimal in the sense that if any algorithm can find a schedule where all tasks meet their deadlines, then EDF can meet the deadlines [2].

According to [3], a hyperperiod is defined as  $HP = [\zeta, 2 * LCM + \zeta]$ , where  $LCM$  is the well-known Least Common Multiple of the tasks periods and  $\zeta$  is the largest task offset. This algorithm, in our original work assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks  $HP^{(p,\psi_h)} = [\zeta^{(p,\psi_h)}, 2 * LCM + \zeta^{(p,\psi_h)}]$ , where  $LCM^{p,\psi_h}$  is the well-known Least Common Multiple of tasks periods and  $(\zeta^{p,\psi_h})$  is the largest task offset of all tasks  $\tau_k^{p,\psi_h}$  for each reconfiguration scenario  $\psi_h$  on each processor  $p$ . The problem is to find which solution proposed by the agent that reduces the response time. To obtain these results, the intelligent agent calculates the residual time  $R_i^{p,\psi_h}$  before and after each addition scenario and calculates the minimum of those proposed solutions in order to obtain  $Resp_k^{p,\psi_h}$  optimal noted  $Resp_k^{p,\psi_h,opt}$ .

Where  $Resp_k^{p,\psi_h^{opt}}$  is the minimum of the response time of the current system under study calculated by the intelligent agent.

To calculate this previous value  $Resp_k^{p,\psi_h^{opt}}$ , we proposed a new theoretical concepts  $R_i^{p,\psi_h}$ ,  $S_i^{p,\psi_h}$ ,  $s_i^{p,\psi_h}$ ,  $f_i^{p,\psi_h}$  and  $L_i^{p,\psi_h}$  for the case of real-time sporadic operating system (OS) tasks. Where  $R_i^{p,\psi_h}$  is the residual time of task  $\sigma_i^{p,\psi_h}$ ,  $S_i^{p,\psi_h}$  denotes the first release time of task  $\sigma_i^{p,\psi_h}$ ,  $s_i^{p,\psi_h}$  is the last release time of task  $\sigma_i^{p,\psi_h}$ ,  $f_i^{p,\psi_h}$  denotes the estimated finishing time of task  $\sigma_i^{p,\psi_h}$ , and  $L_i^{p,\psi_h}$  denotes the laxity of task  $\sigma_i^{p,\psi_h}$  for each reconfiguration scenario  $\psi_h$  and on each processor  $p$ .

The organization of this work is as follows. Section 2 introduces the related work of the proposed approach and gives the basic guarantee algorithm. In Sect. 3, we present the new approach with deadline tolerance for optimal scheduling theory. Section 4 presents the performance study, showing how this work is a significant extension to the state of the art of EDF scheduling and discusses experimental results of the proposed approach research. Section 5 summarizes the main results and presents the conclusion of the proposed approach and describes the intended future works.

## 2 Background

We present related works dealing with reconfigurations and real-time scheduling of embedded systems. Today, real-time embedded systems are found in many diverse application areas including; automotive electronics, avionics, telecommunications, space systems, medical imaging, and consumer electronics. In all of these areas, there is rapid technological progress. Companies building embedded real-time systems are driven by a profit motive. To succeed, they aim to meet the needs and desires of their customers by providing systems that are more capable, more flexible, and more effective than their competition, and by bringing these systems to market earlier. This desire for technological progress has resulted in a rapid increase in both software complexity and the processing demands placed on the underlying hardware [3].

To address demands for increasing processor performance, silicon vendors no longer concentrate wholly on the miniaturisation needed to increase processor clock speeds, as this approach has led to problems with both high power consumption and excessive heat dissipation. Instead, there is now an increasing trend towards using multiprocessor platforms for high-end real-time applications [3].

For these reasons, we will use in our work the case of real-time scheduling on homogeneous multiprocessor platforms. Before presenting our original contribution, we will present some definitions below. According to [1], each periodic task is described by an initial offset  $a_i$  (activation time), a worst-case execution time (WCET)  $C_i$ , a relative deadline  $D_i$  and a period  $T_i$ .

According to [4], each sporadic task is described by minimum interarrival time  $P_i$  which is assumed to be equal to its relative deadline  $D_i$ , and a worst-case execution time (WCET)  $C_i$ . Hence, a sporadic task set will be denoted as follows:

$Sys_2 = \{\sigma_i(C_i, D_i)\}$ ,  $i=1$  to  $m$ . Reconfiguration policies in the current paper are classically distinguished into two strategies: static and dynamic reconfigurations. Static reconfigurations are applied off-line to modify the assumed system before any system cold start, whereas dynamic reconfigurations are dynamically applied at run-time, which can be further divided into two cases: manual reconfigurations applied by users and automatic reconfigurations applied by intelligent agents [1,5]. This book chapter work focuses on the dynamic reconfigurations of assumed mixture of off-line and on-line workloads that should meet deadlines defined according to user requirements. The extension of the proposed algorithm should be straightforward, when this assumption does not hold and its running time is  $O(n + m)$  [6].

## 2.1 State of the Art

Nowadays, several interesting studies have been published to develop reconfigurable embedded control systems. In [7] Marian et al. propose a static reconfiguration technique for the reuse of tasks that implement a broad range of systems. The work in [11] proposes a methodology based on the human intervention to dynamically reconfigure tasks of considered systems. In [10], an ontology-based agent is proposed by Vyatkin et al. to perform system reconfigurations according to user requirements and also the environment evolution. Window-constrained scheduling is proposed in [8], which is based on an algorithm named dynamic window-constrained scheduling (DWCS). The research work in [9] provides a window-constrained-based method to determine how much a task can increase its computation time, without missing its deadline under EDF scheduling. In [9], a window-constrained execution time can be assumed for reconfigurable tasks in  $n$  among  $m$  windows of jobs. In the current paper, a window constrained schedule is used to separate old and new tasks that assumed sporadic on each processor  $p$  and after each reconfiguration scenario  $\psi_h$ . Old and new tasks are located in different windows to schedule the system with a minimum response time. In [5], a window constrained schedule is used to schedule the system with a low power consumption.

In the following, we only consider periodic and sporadic tasks. Few results have been proposed to deal with deadline assignment problem. Baruah, Buttazo and Gorinsky in [1] propose to modify the deadlines of a task set to minimize the output, seen as secondary criteria of this work. So, we note that the optimal scheduling algorithm based on the EDF principles and on the dynamic reconfiguration scenario  $\psi_h$  is that we propose in the current original work in which we give solutions computed and presented by the intelligent agent for users to respond to their requirements.

## 2.2 Formalization

To illustrate the key point of the proposed dynamically approach, we assume that there are  $K$  identical processors numbered from 1 to  $K$ , and  $n$  real-time tasks numbered from 1 to  $n$  that composed a feasible subset of tasks entitled  $\xi_{old}$  and

need to be scheduled. At time  $t$  and before the application of the reconfiguration scenario  $\psi_h$ , each one of the tasks of  $\xi_{old}$  is feasible, e.g. the execution of each instance in each processor is finished before the corresponding deadline and the tasks are not assumed to be arranged in any specific order.

Every processor  $p$  assigns a set of periodic tasks  $TS^p = \{\tau_1^p, \tau_2^p, \dots, \tau_n^p\}$ . This allocation is made with an allowance algorithm at the time of the design, for example by using one of the well known techniques: first-fit (FF), next-fit (NF), best-fit (BF), worst-fit (WF). These tasks are independent and can be interrupted at any time. Every task  $\tau_i^p$  has an execution time (Worst Case Execution Time)  $C_i^p$ , one period  $T_i^p$ , a deadline  $D_i^p$  which is assumed to be less than or equal to its period, e.g.  $D_i^p \leq T_i^p$ . Every task instance  $k$  has to respect its absolute deadline, namely the  $k^{th}$  authority of the task  $\tau_i^p$ , named  $\tau_{i,k}^p$  must be completed before time  $D_{i,k}^p = (k-1)T_i^p + D_i^p$ . These tasks are handled by a global scheduler (GS), which assigns them to processors by using the state informations of the local schedulers. Moreover, under EDF scheduling, a task will fit on a processor as long as the total utilization of all tasks assigned to that processor does not exceed unity (the total utilization factor = 1). Finally, for reasons of simplicity, we assume that the migration cost of the tasks are equal to zero.

We assume now the arrival at run-time of a second subset  $\xi_{new}$  which is composed of  $m$  real-time tasks at time  $t_1$  ( $t_1 = t + \Delta t$ ). We have a system  $Current_{Sys}(t_1)$  composed of  $n + m$  tasks. In this case a reconfiguration scenario  $\psi_h$  is applied. The reconfiguration of the system  $Sys^{\psi_h}$  means the modification of its implementation that will be as follows at time  $t_1$ :

$$\xi^{\psi_h} = Current_{Sys}^{\psi_h}(t_1) = \xi_{old} \cup \xi_{new}^{\psi_h}$$

where  $\xi_{old}$  is a subset of old tasks which are not affected by the reconfiguration scenario  $\psi_h$  (e.g. they implement the system before the time  $t_1$ ), and  $\xi_{new}^{\psi_h}$  a subset of new tasks in the system. We assume that an updated task is considered as a new one at time  $t_1$ . When the reconfiguration scenario  $\psi_h$  is applied at time  $t_1$ , two cases exist:

- If tasks of  $\xi^{\psi_h} = \xi_{old} \cup \xi_{new}^{\psi_h}$  are feasible, then no reaction should be done by the agent,
- otherwise, the agent should provide different solutions for users in order to re-obtain the system's feasibility.

## Running Example

In this section, we demonstrate the performance of our proposed approach for both periodic synchronous and asynchronous, and sporadic tasks. The simulation runs on our tool RT-Reconfiguration and proved by the real-time simulator Cheddar [12] with a task set composed of old tasks ( $\xi_{old}$ ) and new tasks ( $\xi_{new}^{\psi_h}$ ) on the processor  $p$  for each reconfiguration scenario  $\psi_h$ . We illustrate with a simplified example to ease the understanding of our approach. The task set considered for this example is given in Table 1 and is composed of 10 tasks. The sum of utilization of all tasks is given in Table 1 and is equal to 426.1%.

**Table 1.** Task parameters of running example.

Tasks	$C_i$	$D_i$	$T_i = P_i$
$\tau_1$	2	9	7
$\tau_2$	3	21	20
$\tau_3$	2	9	9
$\tau_4$	2	13	10
$\tau_5$	3	15	9
$\tau_6$	14	21	19
$\tau_7$	10	24	16
$\tau_8$	8	18	18
$\tau_9$	13	16	17
$\tau_{10}$	5	11	12

We have 3 identical processors in our system to schedule these tasks. In this case, we assume that each task's deadline is less than or equal to its period. The worst case execution times, deadlines, and the time periods of all tasks are generated randomly. In this experiment, the system runs for time units equal to hyper-period of periodic tasks.

In this experiment, our task set example is initially implemented by 5 characterized old tasks ( $\xi_{old} = \{\tau_1; \tau_2; \tau_3; \tau_4; \tau_5\}$ ). These tasks are feasible because the processor utilization factor  $U = 1.19 \leq 3$ . These tasks should meet all required deadlines defined in user requirements and we have  $Feasibility(Current_{\xi_{old}}(t)) \equiv True$ .

Firstly, tasks are partitioned; task  $\tau_1$  is partitioned on first processor,  $\tau_2$  and  $\tau_3$  are partitioned on processor 2 while task  $\tau_4$  and  $\tau_5$  are partitioned on processor 3. We have three sets of local tasks. As there is only one task on first processor then task  $\tau_1$  utilization factor is the same as the first processor 1 utilization factor ( $u^{1,0} = 0.285 \leq 1$ ) while utilization factors of processor 2 and processor 3 are calculated as follows:

$$U^{2,0} = \sum_{i=1}^{(2)^2} \frac{C_i^2}{T_i^2} = 0.372 < 1,$$

$$U^{3,0} = \sum_{i=1}^{(2)^3} \frac{C_i^3}{T_i^3} = 0.533 < 1.$$

We suppose that a first reconfiguration scenario  $\psi_1$  ( $h=1$ ) is applied at time  $t_1$  to add 5 new tasks  $\xi_{new}^{\psi_1} = \{\tau_6; \tau_7; \tau_8; \tau_9; \tau_{10}\}$ . The new processor utilization becomes  $U^{\psi_1} = 4.261 > 3$  time units. Therefore the system is unfeasible.  $Feasibility(Current_{\xi}^{\psi_1}(t_1)) \equiv False$ . Indeed, if the number of tasks increases, then the overload of the system increases too. Our optimal earliest deadline first (OEDF) algorithm is based on the Guarantee Algorithm presented by Buttazo and Stankovic in [4]. Indeed, OEDF algorithm is an extended and

ameliorate version of Guarantee Algorithm that usually guarantee the system's feasibility.

### 3 New Approach with Deadline Tolerance

In this section we will present some preliminaries concepts and we will describe our contribution after.

In [4], Buttazo and Stankovic present the Guarantee Algorithm without the notion of deadline tolerance, and then we will extend the algorithm in our new proposed approach by including tolerance indicator and task rejection policy. For this reason, and in order to more explain these notions we will present some preliminaries.

#### 3.1 Preliminaries

$\xi$  denotes a set of active sporadic tasks  $\sigma_i$  ordered by increasing deadline in a linked list,  $\sigma_1$  being the task with the shortest absolute deadline.

$a_i$  denotes the arrival time of task  $\sigma_i$ , i.e., the time at which the task is activated and becomes ready to execute.

$C_i$  denotes the maximum computation time of task  $\sigma_i$ , i.e., the worst case execution time (WCET) needed for the processor to execute task  $\sigma_{i,k}$  without interruption.

$c_i$  denotes the dynamic computation time of task  $\sigma_i$ , i.e., the remaining worst case execution time needed for the processor, at the current time, to complete task  $\sigma_{i,k}$  without interruption.

$d_i$  denotes the absolute deadline of task  $\tau_i$ , i.e., the time before which the task should complete its execution, without causing any damage to the system.

$D_i$  denotes the relative deadline of task  $\sigma_i$ , i.e., the time interval between the arrival time and the absolute deadline.  $S_i$  denotes the first start time of task  $\sigma_i$ , i.e., the time at which task  $\sigma_i$  gains the processor for the first time.  $s_i$  denotes the last start time of task  $\sigma_i$ , i.e., the last time, before the current time, at which task  $\sigma_i$  gained the processor.

$f_i$  denotes the estimated finishing time of task  $\sigma_i$ , i.e., the time according to the current schedule at which task  $\sigma_i$  should complete its execution and leave the system.

$L_i$  denotes the laxity of task  $\sigma_i$ , i.e., the maximum time task  $\sigma_i$  can be delayed before its execution begins.

$R_i$  denotes the residual time of task  $\sigma_i$ , i.e., the length of time between the finishing time of  $\sigma_i$  and its absolute deadline. Baruah et al. [13] present a necessary and sufficient feasibility test for synchronous systems with pseudo-polynomial complexity. The other known method is to use response time analysis, which consists of computing the worst-case response time (WCRT) of all tasks in a system and ensuring that each task WCRT is less than its relative deadline. To avoid these problems, and to have a feasible system in this paper, our proposed tool RT-Reconfiguration can be used. For this reason, we present the following relationships among the parameters defined above:

$$d_i = a_i + D_i \quad (1)$$

$$L_i = d_i - a_i - C_i \quad (2)$$

$$R_i = d_i - f_i \quad (3)$$

$$f_1 = t + c_1; \quad f_i = f_{i-1} + c_i \quad \forall i > 1 \quad (4)$$

The basic properties stated by the following lemmas and theorems are used to derive an efficient  $O(n + m)$  algorithm for analyzing the schedulability of the sporadic task set whenever a new task arrives in the systems.

**Lemma 1.** Given a set  $\xi = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$  of active sporadic tasks ordered by increasing deadline in a linked list, the residual time  $R_i$  of each task  $\sigma_i$  at time  $t$  can be computed by the following recursive formula:

$$R_1 = d_1 - t - c_1 \quad (5)$$

$$R_i = R_{i-1} + (d_i - d_{i-1}) - c_i. \quad (6)$$

**Proof.** By the residual time definition (Eq. 3) we have:

$$R_i = d_i - f_i.$$

By the assumption on set  $\xi$ , at time  $t$ , the task  $\sigma_1$  in execution and cannot be preempted by other tasks in the set  $\xi$ , hence its estimated finishing time is given by the current time plus its remaining execution time:

$$f_1 = t + c_1$$

and, by Eq. 3, we have:

$$R_1 = d_1 - f_1 = d_1 - t - c_1.$$

For any other task  $\sigma_i$ , with  $i > 1$ , each task  $\sigma_i$  will start executing as soon as  $\sigma_{i-1}$  completes, hence we can write:

$$f_i = f_{i-1} + c_i \quad (7)$$

and, by Eq. 3, we have:

$$\begin{aligned} R_i &= d_i - f_i = d_i - f_{i-1} - c_i = \\ &= d_i - (d_{i-1} - R_{i-1}) - c_i = R_{i-1} + (d_i - d_{i-1}) - c_i \end{aligned}$$

and the lemma follows.

**Lemma 2.** A task  $\sigma_i$  is guaranteed to complete within its deadline if and only if  $R_i \geq 0$  [4].



**Theorem 3.** A set  $\xi = \{\sigma_i, i = 1 \text{ to } m\}$  of  $m$  active sporadic tasks ordered by increasing deadline is feasibly schedulable if and only if  $R_i \geq 0$  for all  $\sigma_i \in \xi$ , [4].

### 3.2 Feasibility Analysis for Tasks

By considering real-time tasks and as we mentioned before, the schedulability analysis should be done in the hyperperiod  $HP^{(p, \psi_h)} = [\zeta^{(p, \psi_h)}, 2 * LCM + \zeta^{(p, \psi_h)}]$ , where  $LCM^{p, \psi_h}$  is the well-known Least Common Multiple of tasks periods and  $(\zeta^{p, \psi_h})$  is the largest task offset of all tasks  $\tau_k^{p, \psi_h}$  for each reconfiguration scenario  $\psi_h$  on each processor  $p$ .

Let  $n + m$  be the number of tasks respectively in  $\xi_{old}$  and  $\xi_{new}^{\psi_h}$ . By assuming unfeasible system at time  $t_1$ , and every processor  $p$  will execute its tasks in local by using EDF, the following formula is satisfied:

$$\sum_{i=1}^{n+m} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} > K, \text{ where } K \text{ is the number of identical processors.}$$

Our proposed algorithm provides guarantees to both old and new tasks in each processor  $p$  if and only if,

$$\sum_{i=1}^{n-j} \frac{C_i^{p, \psi_h}}{T_i^{p, \psi_h}} + \sum_{i=n-j+1}^{n+m} \frac{C_i^{p, \psi_h}}{T_i^{p, \psi_h}} \leq 1$$

where  $\sum_{i=1}^{n-j} \frac{C_i^{p, \psi_h}}{T_i^{p, \psi_h}}$  denotes sum of utilization factor of  $n$  old tasks in processor  $p$  for each reconfiguration scenario  $\psi_h$  and,  $\sum_{i=n-j+1}^{n+m} \frac{C_i^{p, \psi_h}}{T_i^{p, \psi_h}}$  denotes sum of utilization factor of new arrival  $m$  tasks to the processor  $p$  for each reconfiguration scenario  $\psi_h$ .

We propose, for each reconfiguration scenario  $\psi_h$ , to add the tasks of  $\xi_{old}$  to a linked list  $L_{old}^{\psi_h}$  that we sort on the increasing order of their utilization factor values.

### 3.3 Contribution: An Algorithm for Feasibility Testing with Respect to Sporadic Task Systems

In the current book chapter, we suppose that on each processor  $p$ , each system  $\xi^{(p)}$  can be automatically and repeatedly reconfigured at each reconfiguration scenario  $\psi_h$ .  $\xi^{(p)}$  is initially considered as  $\xi^{(p, 0)}$  and after the  $h_{th}$  reconfiguration  $\xi^{(p)}$  turns into  $\xi^{(p, \psi_h)}$ , where  $h \in 1..M$ . We define  $VP_1^{p, \psi_h}$  and  $VP_2^{p, \psi_h}$  two virtual processors to virtually execute old and new sporadic tasks, implementing the system after the  $h_{th}$  reconfiguration scenario for each processor  $p$ . In  $\xi^{(p, \psi_h)}$ , all old tasks from  $\xi^{(p, \psi_{h-1})}$  are executed by the newly updated  $VP_1^{(p, \psi_h)}$  and the added sporadic tasks are executed by  $VP_2^{(p, \psi_h)}$ . The proposed intelligent agent is trying to minimize the response time  $Resp_k^{p, \psi_h, opt}$  of  $\xi^{(\psi_h)}$  after each reconfiguration scenario  $\psi_h$  and for each processor  $p$ .

For example, after the first addition scenario,  $\xi^{(p,0)}$  turns into  $\xi^{(p,1)}$ .  $\xi^{(p,1)}$  is automatically decomposed into  $VP_1^{(p,1)}$  and  $VP_2^{(p,1)}$  for old and new tasks with the processor utilization factors  $UVP_1^{(p,1)}$  and  $UVP_2^{(p,1)}$  respectively on each processor p.

After each addition scenario, the proposed intelligent agent proposes to modify the virtual processors, to modify the deadlines of old and new tasks, the WCETs and the activation time of some tasks, to send some tasks from processor i to another processor j, or to remove some soft tasks as following:

- **Solution 1:** Moving some arrival tasks to be scheduled in idle times for each reconfiguration scenario  $\psi_h$  and on each processor p. (idle times are caused when some tasks complete before its worst case execution time) (S1)
- **Solution 2:** maximize the  $d_i^{p,\psi_h}$  for each reconfiguration scenario  $\psi_h$  and on each processor p (S2)

By applying Eq. 3 that notices:

$R_i = d_i - f_i$ , we have:

$$R_i^{p,\psi_h} = d_i^{p,\psi_h} - t - C_i^{p,\psi_h}.$$

Or, to obtain a feasible system after a reconfiguration scenario  $\psi_h$ , the following formula must be enforced:

$$R_i^{p,\psi_h} \geq 0 \text{ on each processor p.}$$

By this result we can write:  $d_{inew}^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0$ , where  $d_{inew}^{p,\psi_h} = d_i^{p,\psi_h} + \theta_i^{p,\psi_h}$ . So,  $d_i^{p,\psi_h} + \theta_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0 \Rightarrow$

$$\theta_i^{p,\psi_h} \geq t + C_i^{p,\psi_h} - d_i^{p,\psi_h}.$$

- **Solution 3:** minimize the  $c_i$  for each reconfiguration scenario  $\psi_h$  and on each processor p (S3)

By applying Eq. 3 that notices:

$R_i = d_i - f_i$ , we have:

$$R_i^{p,\psi_h} = d_i^{p,\psi_h} - t - C_i^{p,\psi_h}.$$

Or, to obtain a feasible system after a reconfiguration scenario, the following formula must be enforced:

$$R_i^{p,\psi_h} \geq 0.$$

By this result we can write:  $d_i^{p,\psi_h} - t - C_{inew}^{p,\psi_h} \geq 0$ , where  $C_{inew}^{p,\psi_h} = C_i^{p,\psi_h} + \beta_i^{p,\psi_h}$ . So,  $d_i^{p,\psi_h} - t - C_i^{p,\psi_h} - \beta_i^{p,\psi_h} \geq 0 \Rightarrow d_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq \beta_i^{p,\psi_h}$

$$\Rightarrow \beta_i^{p,\psi_h} \leq d_i^{p,\psi_h} - t - C_i^{p,\psi_h}$$

- **Solution 4.** Enforcing the release time to come back:  $a_i^{p,\psi_h} \rightarrow a_{i_{new}}^{p,\psi_h} \rightarrow (a_{i_{new}}^{p,\psi_h} = a_i^{p,\psi_h} + \Delta^{p,\psi_h}t)$  for each reconfiguration scenario  $\psi_h$  and on each processor p (S4)

By applying Eq. 1 that notices:

$d_i = a_i + D_i$ , we have:

$$R_i^{p,\psi_h} = a_i^{p,\psi_h} + D_i^{p,\psi_h} - t - C_i^{p,\psi_h}.$$

Or, to obtain a feasible system after a reconfiguration scenario, the following formula must be enforced:

$$R_i^{p,\psi_h} \geq 0 \Rightarrow a_i^{p,\psi_h} + D_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0.$$

By this result we can write:

$$a_{i_{new}}^{p,\psi_h} + D_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0, \text{ where } a_{i_{new}}^{p,\psi_h} = a_i^{p,\psi_h} + \Delta^{p,\psi_h}t.$$

So, we obtain:  $a_i^{p,\psi_h} + \Delta^{p,\psi_h}t + D_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0.$

$$\Rightarrow \Delta^{p,\psi_h}t \geq t + C_i^{p,\psi_h} - a_i^{p,\psi_h} - D_i^{p,\psi_h}.$$

- **Solution 5:** Tolerate some non critical Tasks  $m_1^p$  among  $(n+m)^p$  (according to the (m, n) firm model), on each processor p for a reasonable cost, and for each reconfiguration scenario  $\psi_h$  (S5)

$$\xi^p = \{\tau_i^p(C_i^p, D_i^p, m_i^p, I_i^p), i = 1 \text{ to } n^p\}.$$

$m_i^p = 1$ , it tolerates missing deadline,

$m_i^p = 0$ , it doesn't tolerate missing deadline,

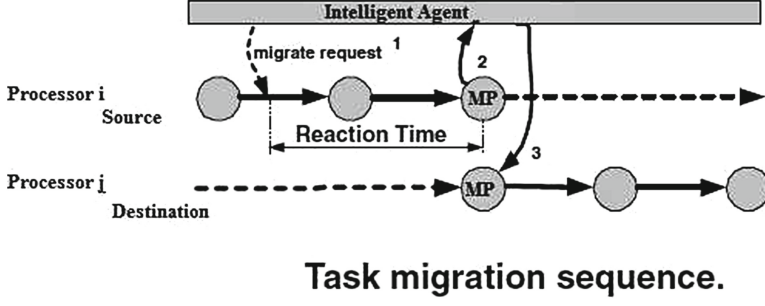
$I_i^p = H$ , Hard task,

$I_i^p = S$ , Soft task.

- **Solution 6:** Migration of some tasks from a processor source i in order to be scheduled on another processor destination j for each reconfiguration scenario  $\psi_h$  (S6)

The agent proceeds now as a sixth solution to migrate some tasks of  $\xi_{new}^{p,\psi_h}$  and  $\xi_{old}^p$  on the processor p for each reconfiguration scenario  $\psi_h$ . Indeed, the agent is responsible for allocating the tasks to the K computing processors in an optimal way (Fig. 1).

Run-time task migration can be defined as the relocation of an executing task from its current location, the source processor i, to a new location, the destination processor j ( $i \neq j$ ;  $i, j = 1..K$ ) that must belong to the inclusion set. We need by inclusion set in paper, the set of processors in which tasks can be scheduled after any reconfiguration scenario  $\psi_h$  when a migration request has done and in this case all the relevant state information of that migration is transferred to the new processor. Otherwise, it is called exclusion set.



**Fig. 1.** The task migration sequence.

This allows the OS to e.g., minimize energy savings and response time of the whole system. It also enables processors management by moving tasks away from processors with a high amount of workload or which have their utilization factors  $>1$ . The architectural differences between the source processor  $i$  and destination source processor  $j$  are masked by capturing and transferring the logical task state, shown by Fig. 2. In order to relocate a task, the intelligent agent notifies the task by means of a migration request signal<sup>(1)</sup>. Whenever that signaled task reaches a migration point (MP), it checks if there is a pending migration request or the destination processor  $j$  belongs to the exclusion group of the current migrated task for each reconfiguration scenario  $\psi_h$ . In such case of these two reasons, all the relevant state information of that migration point is transferred to the intelligent agent<sup>(2)</sup>. Consequently, the intelligent agent will instantiate the same task on a different processor. The new task instantiation will be initialized using the state information previously captured by the intelligent agent<sup>(3)</sup>. Finally, the task resumes execution at the corresponding migration point (MP).

- **Solution 7:** Removal of some non critical tasks (to be rejected) for each reconfiguration scenario  $\psi_h$  and on each processor  $p$  (S7)

$$\xi^p = \{\tau_i^p(C_i^p, D_i^p, m_i^p, I_i^p), i = 1 \text{ to } n^p\}.$$

$m_i^{p,\psi_h} = 1$ , it tolerates missing deadline,

$m_i^{p,\psi_h} = 0$ , it doesn't tolerate missing deadline,

$I_i^{p,\psi_h} = H$ , Hard task,

$m_i^{p,\psi_h} = S$ , Soft task.

For every solution the corresponding response time is:

$Resp_{k,1}^{p,\psi_h}$  = the response time calculated by the first solution,

$Resp_{k,2}^{p,\psi_h}$  = the response time calculated by the second solution,

$Resp_{k,3}^{p,\psi_h}$  = the response time calculated by the third solution,

$Resp_{k,4}^{p,\psi_h}$  = the response time calculated by the fourth solution,

$Resp_{k,5}^{p,\psi_h}$  = the response time calculated by the fifth solution,  
 $Resp_{k,6}^{p,\psi_h}$  = the response time calculated by the sixth solution,  
 $Resp_{k,7}^{p,\psi_h}$  = the response time calculated by the seventh solution.

We define now,  $Resp_k^{p,\psi_h}$  optimal noted  $Resp_k^{p,\psi_h^{opt}}$  according to the previous seven solutions calculated by the intelligent Agent (Solution 1, Solution 2, Solution 3, Solution 4, Solution 5, Solution 6 and Solution 7) by the following expression:  $Resp_k^{p,\psi_h^{opt}} = \min(Resp_{k,1}^{p,\psi_h}, Resp_{k,2}^{p,\psi_h}, Resp_{k,3}^{p,\psi_h}, Resp_{k,4}^{p,\psi_h}, Resp_{k,5}^{p,\psi_h}, Resp_{k,6}^{p,\psi_h} \text{ and } Resp_{k,7}^{p,\psi_h})$  (the minimum of the seven values). So, the calculation of  $Resp_k^{p,\psi_h^{opt}}$  allows us to obtain and to calculate the minimizations of response times values and to get the optimum of these values. In conclusion, we can deduce that by arrival of  $\xi_{new}^{\psi_h}$  tasks at run-time and the whole system become unfeasible, the following formula is satisfied for each reconfiguration scenario  $\psi_h$ :

$$\sum_{i=1}^{(n+m)^{\psi_h}} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} > K, \text{ where } K \text{ is the number of identical processors.}$$

Then, after the reconfiguration scenario  $\psi_h$  was applied at run-time to the whole system by the intelligent agent, our proposed algorithm provides guarantees to both old and new tasks if and only if, we have in each processor p for each reconfiguration scenario  $\psi_h$ :

$$\sum_{i=1}^{(n+m)^{(p,\psi_h)}} \frac{C_i^{(p,\psi_h)}}{T_i^{(p,\psi_h)}} \leq 1, \text{ in each processor p for each reconfiguration scenario } \psi_h.$$

Moreover, we have calculated  $R_k^{(p,\psi_h)^{opt}} = \min(R_{k,1}^{(p,\psi_h)}, R_{k,2}^{(p,\psi_h)}, R_{k,3}^{(p,\psi_h)}, R_{k,4}^{(p,\psi_h)}, R_{k,5}^{(p,\psi_h)}, R_{k,6}^{(p,\psi_h)} \text{ and } R_{k,7}^{(p,\psi_h)})$ ; so we obtain also:

$$\sum_{i=1}^{(n+m)^{(p,\psi_h)}} \frac{C_i^{(p,\psi_h)}}{T_i^{(p,\psi_h)}} < 1, \text{ with } 1 \leq p \leq K, 1 \leq h \leq M.$$

We can observe that all tasks meet their deadlines after a reconfiguration scenario  $\psi_h$  was applied at run-time. We can also observe that our proposed algorithm outperforms other scheduling multiprocessor algorithms and a number of scheduling events are much lower than appearing in others.

### 3.4 The General OEDF Scheduling Strategy

When dealing with the deadline tolerance factor  $m_i$ , each task has to be computed with respect to the deadline tolerance factor  $m_i$ .

**Algorithm GUARANTEE**( $\xi; \sigma_a$ )  
**For each**  $h$  **in**  $[1..M]$  **Do**  
**begin**  $t = \text{get current time}()$ ;  
 $R_0^{p, \psi_h} = 0$ ;  
 $d_0^{p, \psi_h} = t$ ;  
Insert  $\sigma_a$  in the ordered task list;  
 $\xi^{p, \psi_h} = \xi^{p, \psi_h} \cup \sigma_a$ ;  
 $k = \text{position of } \sigma_a \text{ in the task set } \xi^{p, \psi_h}$ ;  
**for each** task  $\sigma_i^{p, \psi_h}$  **such that**  $i \geq k$  **do**  
{  
 $R_i^{p, \psi_h} = R_{i-1}^{p, \psi_h} + (d_i^{p, \psi_h} - d_{i-1}^{p, \psi_h}) - c_i^{p, \psi_h}$ ;  
**if** ( $R_i^{p, \psi_h} \geq 0$ ) **then**  
{  
**return** ("Guaranteed");  
}  
} **else return**  
("You can try by using solution 1, or,  
You can try by using solution 2, or,  
You can try by using solution 3, or,  
You can try by using solution 4, or,  
You can try by using solution 5, or,  
You can try by using solution 6, or,  
You can try by using solution 7 !");  
}

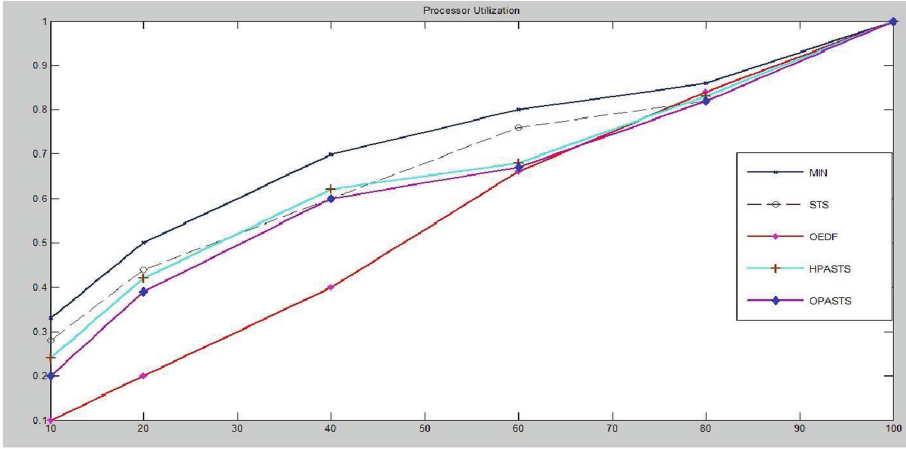
- Compute( $Resp_{k,1}^{p, \psi_h}$ );
- Compute( $Resp_{k,2}^{p, \psi_h}$ );
- Compute( $Resp_{k,3}^{p, \psi_h}$ );
- Compute( $Resp_{k,4}^{p, \psi_h}$ );
- Compute( $Resp_{k,5}^{p, \psi_h}$ );
- Compute( $Resp_{k,6}^{p, \psi_h}$ );
- Compute( $Resp_{k,7}^{p, \psi_h}$ );
- Generate( $Resp_k^{p, \psi_h^{opt}}$ );

**end**

We show the results of our optimal proposed algorithm by means of experimental result's evaluation.

## 4 Experimental Results

In order to evaluate our optimal OEDF algorithm, we consider the following experiments applied to our running example.



**Fig. 2.** Processor utilization.

#### 4.1 Simulations

To quantify the benefits of the proposed approach (OEDF algorithm) over the predictive system shutdown (PSS) approach, over the MIN algorithm, the OPASTS algorithm and over the HPASTS algorithm. We performed a number of simulations to compare the response time and the utilization processor under the four strategies. The PSS technique assumes the complete knowledge of the idle periods while the MIN algorithm assumes the complete knowledge of the arrivals of sporadic tasks. For more details about both four techniques, you can see [14]. The OEDF scheduling result is shown in figure (Fig. 2).

#### 4.2 Discussion

We observe that our approach, by the solutions of the OEDF algorithm gives us the minimum bound for response time and utilization factor. This observation was proven by the results given by OEDF algorithm which are lower (better) than these of the solutions given by the predictive system shutdown approach, the MIN algorithm, the OPASTS algorithm and the HPASTS algorithm. Also, we observe that, when we have no knowledge of the arrival of sporadic tasks, our proposed algorithm is optimal and gives better results than others for a big number of arrival sporadic tasks and in overload conditions, but in a small number of tasks or light workload, OEDF algorithm is optimal but not strictly since it gives results close to that of the solutions of MIN, OPASTS and HPASTS algorithms, but it is efficient and effective.

### 5 Conclusions

This book chapter deals with reconfigurable homogeneous multiprocessor systems to be implemented by hybrid systems composed of a mixture of periodic

and sporadic tasks that should meet real-time constraints. In this work, we propose an optimal scheduling algorithm based on the EDF principles and on the dynamic reconfiguration for the minimization of the response time of sporadic and periodic constrained deadline real-time tasks on multiprocessor systems and proven it correct.

## References

1. Gharsellaoui, H., Khalgui, M., BenAhmed, S.: Feasible Automatic Reconfigurations of Real-Time OS Tasks. IGI-Global Knowledge, London (2012)
2. Dertouzos, M.: Control robotics: the procedural control of physical processes. In: Proceedings of the IFIP Congress (1974)
3. Balbastre, P., Ballester, R., Brocal V., Ripoll, L.: Task period selection to minimize hyperperiod, emerging technologies and factory automation. In: 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–4. IEEE Press, Toulouse, France (2011)
4. Buttazzo, G., Stankovic, J.: RED: robust earliest deadline scheduling. In: 3rd International Workshop On Responsive Computing Systems, Austin (1993)
5. Wang, X., Khalgui, M., Li, Z.W.: Dynamic low power reconfigurations of real-time embedded systems. In: 16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–4. IEEE Press, Toulouse, France (2011)
6. Tia, T., Liu, J.W.-S., Sun, J., Ha, R.: A linear-time optimal acceptance test for scheduling of hard real-time tasks, Technical report. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign (1994)
7. Marian, N., Angelov, C., Sierszecki, K.: Design models for reusable and reconfigurable state machines. In: Yang, L.T., et al. (eds.) Proceedings of Embedded Ubiquitous Computing (2005)
8. Schwan, K., West, R.: Dynamic window-constrained scheduling for multimedia applications. In: 6th IEEE International Conference on Multimedia Computing and Systems (1999)
9. Balbastre, P., Ripoll, I., Crespo, A.: Schedulability analysis of window-constrained execution time tasks for real-time control. In: 14th IEEE International Conference on Euromicro Conference Real-Time Systems (ECRTS) (2002)
10. Al-Safi, Y., Vyatkin, V.: An ontology-based reconfiguration agent for intelligent mechatronic systems. In: Mařík, V., Vyatkin, V., Colombo, A.W. (eds.) HoloMAS 2007. LNCS (LNAI), vol. 4659, pp. 114–126. Springer, Heidelberg (2007)
11. Rooker, M.N., Subder, C., Strasser, T., Zoitl, A., Hummer, O., Ebenhofer, G.: Zero downtime reconfiguration of distributed automation systems: the CEDAC approach. In: 3rd IEEE International Conference on Industrial Applications of Holonic and Multi-Agent Systems, Regensburg (2007)
12. Legrand, J., Singhoff, L.M.F.: Cheddar : a flexible real time scheduling framework. In: ACM SIGAda Ada Letters, vol. 24, no 4, pp. 1–8. ACM Press, ISSN:1094–3641 (2004)
13. Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D.: On-line scheduling in the presence of overload. In: IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico (1991)
14. Hong, I., Potkonjak, M., Srivastava, B.M.: On-line scheduling of hard real-time tasks on variable voltage processor. In: 8th International Conference on Computer-Aided Design, San Jose, California, USA (1998)



Software Technologies

8th International Joint Conference, ICSOFT 2013,

Reykjavik, Iceland, July 29-31, 2013, Revised Selected

Papers

Cordeiro, J.; van Sinderen, M. (Eds.)

2014, XII, 323 p. 95 illus., Softcover

ISBN: 978-3-662-44919-6