An EDF-based Scheduling Algorithm for Real-time Reconfigurable Sporadic Tasks

Hamza Gharsellaoui[1], Mohamed Khalgui[1] and Samir Ben Ahmed[1,2]
[1]*INSAT Institute, University of Carthage, Tunis, Tunisia*
[2]*FST Faculty, University of Tunis El Manar, Tunis, Tunisia*

Keywords: EDF-based Scheduling Algorithm, Real-time Reconfigurable Sporadic Tasks, Intelligent Agent, Automatic Reconfigurations.

Abstract: This paper deals with the problem of scheduling the mixed workload of both homogeneous multiprocessor on-line sporadic and off-line periodic tasks in a hard reconfigurable real-time environment 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 to verify if all tasks meet the required deadlines after a reconfiguration scenario ψ_h (h $\in$ 1..M, we assume that we have M reconfiguration scenarios), was applied on a multiprocessor embedded system in the case of shared memory. 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 m among n according to the (m,n) firm and a reasonable cost, 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 predifined heuristic. We implement the agent to support these services in order to demonstrate the effectiveness and the excellent performance of the new optimal algorithm in normal and overload conditions.

1 INTRODUCTION

Nowadays, due to the growing class of portable systems, such as personal computing and communication devices, embedded and real-time systems contain new complex software which are increasing by the time. This complexity is growing because many available software development models don't take into account the specific needs of embedded and systems development. The software engineering principles for embedded system should address specific constraints such as hard timing constraints, limited memory and power use, predefined hardware platform technology, and hardware costs. On the other hand, the new generations of embedded control systems are adressing new criteria such as flexibility and agility (H. Gharsellaoui and BenAhmed, 2012). 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 caracterized by its worst case execution times (WCETs) C_i^{p,ψ_h}, an offset (release time) a_i^{p,ψ_h}, a period T_i^{p,ψ_h} and a deadline D_i^{p,ψ_h} for each reconfiguration scenario ψ_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 ξ_{old} and need to be scheduled. The general goal of this paper is to be reassured that any reconfiguration scenario ψ_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 (H. Gharsellaoui and BenAhmed, 2012). To obtain this optimization (minimization of response time),

Gharsellaoui H., Khalgui M. and Ben Ahmed S..
An EDF-based Scheduling Algorithm for Real-time Reconfigurable Sporadic Tasks.
DOI: 10.5220/0004432003770388
In *Proceedings of the 8th International Joint Conference on Software Technologies* (ICSOFT-EA-2013), pages 377-388
ISBN: 978-989-8565-68-6

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 ψ_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,ψ_h} which is assumed to be equal to its relative deadline D_i^{p,ψ_h}, and a worst-case execution time (WCET) C_i^{p,ψ_h} for each reconfiguration scenario ψ_h and on each processor p. A random disturbance is defined in the current paper 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 ψ_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 ψ_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, many real-time systems rely on the EDF scheduling algorithm in the case of uni-processor scheduling theory. This algorithm has been shown to be optimal under many different conditions. For example, 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 (Dertouzos, 1974).

According to (Brocal V., 2011), a hyperperiod is defined as $HP = [\zeta, 2 * LCM + \zeta]$, where LCM is the well-known Least Common Multiple of the tasks periods and ζ is the largest task offset. This algorithm, in our original paper assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks $HP^{(p,\psi_h)} = [\zeta^{(p,\psi_h)}, 2*\text{LCM} + \zeta^{(p,\psi_h)}]$, where LCM^{p,ψ_h} is the well-known Least Common Multiple of tasks periods and (ζ^{p,ψ_h}) is the largest task offset of all tasks τ_k^{p,ψ_h} for each reconfiguration scenario ψ_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,ψ_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 given by the following equation: $Resp_k^{p,\psi_h{}^{opt}} = \min(Resp_{k,1}^{p,\psi_h}, Resp_{k,2}^{p,\psi_h}, Resp_{k,3}^{p,\psi_h}, Resp_{k,4}^{p,\psi_h}, Resp_{k,5}^{p,\psi_h}, Resp_{k,6}^{p,\psi_h}, Resp_{k,7}^{p,\psi_h})$. To calculate these previous values $Resp_{k,1}^{p,\psi_h}$, $Resp_{k,2}^{p,\psi_h}$, $Resp_{k,3}^{p,\psi_h}$, $Resp_{k,4}^{p,\psi_h}$, $Resp_{k,5}^{p,\psi_h}$, $Resp_{k,6}^{p,\psi_h}$, and $Resp_{k,7}^{p,\psi_h}$, we proposed a new theoretical concepts R_i^{p,ψ_h}, S_i^{p,ψ_h}, s_i^{p,ψ_h}, f_i^{p,ψ_h} and L_i^{p,ψ_h} for the case of real-time sporadic operating system (OS) tasks. Where R_i^{p,ψ_h} is the residual time of task σ_i^{p,ψ_h}, S_i^{p,ψ_h} denotes the first release time of task σ_i^{p,ψ_h}, s_i^{p,ψ_h} is the last release time of task σ_i^{p,ψ_h}, f_i^{p,ψ_h} denotes the estimated finishing time of task σ_i^{p,ψ_h}, and L_i^{p,ψ_h} denotes the laxity of task σ_i^{p,ψ_h} for each reconfiguration scenario ψ_h and on each processor p.

A tool RT-Reconfiguration is developed at INSAT institute in university of Carthage, Tunisia, to support all the services offered by the agent. The minimization of the response time is evaluated after each reconfiguration scenario ψ_h and on each processor p to be offered by the agent.

The organization of the paper is as follows. Section 2 introduces the related work of the proposed approach and gives the basic guarantee algorithm. In Section 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 cus-

tomers 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 (Brocal V., 2011). 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 (Brocal V., 2011).

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 (H. Gharsellaoui and BenAhmed, 2012), 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 (Buttazzo and Stankovic, 1993), 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 (H. Gharsellaoui and BenAhmed, 2012), (X. Wang and L, 2011). This paper 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) (T. Tia, 1994).

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 ξ_{old} and need to be scheduled. At time t and before the application of the reconfiguration scenario ψ_h, each one of the tasks of ξ_{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 $TS^p = \{\tau_1^p, \tau_2^p, ..., \tau_n^p\}$. 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 τ_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 τ_i^p, named $\tau_{i,k}^p$ must be completed before time $D_{i,k}^p$ = (k-1)T_i^p + D_i^p. We express all the measures of time (e.g. periods, the deadlines, the calculations) as being multiple of the tick of the processor clock. Every processor p will execute its tasks in local by using EDF, it means that the priorities P_i^p of periodic tasks are dynamic and the scheduler guarantees that every instance of every task will run before its deadline. 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 all the overheads of context exchange, scheduling of tasks, the preemption of the tasks and the migration cost of the tasks are equal to zero.

We assume now the arrival at run-time of a second subset ξ_{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 ψ_h is applied. The reconfiguration of the system Sys^{ψ_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 ξ_{old} is a subset of old tasks which are not affected by the reconfiguration scenario ψ_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 ψ_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.

2.1 State of the Art

Nowadays, several interesting studies have been published to develop reconfigurable embedded control systems. In (C. Angelov and Marian, 2005) Marian et al. propose a static reconfiguration technique for the reuse of tasks that implement a broad range of systems. The work in (M. N. Rooker and Ebenhofer, 2007) proposes a methodology based on the human intervention to dynamically reconfigure tasks of considered systems. In (Al-Safi and Vyatkin, 2007), 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 (West and Schwan, 1999), which is based on an algorithm named dynamic window-constrained scheduling (DWCS). The research work in (P. Balbastre and Crespo, 2002) 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 (P. Balbastre and Crespo, 2002), 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 ψ_h. Old and new tasks are located in different windows to schedule the system with a minimum response time. In (X. Wang and L, 2011), 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 (H. Gharsellaoui and BenAhmed, 2012) 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 ψ_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.

Running Example 1:
To illustrate the key point of the proposed dynamic reconfiguration approach, we consider $\xi = Sys_1 \cup Sys_2$ a set of 5 characterized tasks, shown in Table 1 as a motivational example. $Sys_1 = \tau_A$, τ_B, and $Sys_2 = \sigma_C$, σ_D, and σ_E. τ_A and τ_B are periodic tasks and all the rest (σ_C, σ_D, and σ_E) are sporadic tasks. Each task can be executed immediately after its arrival and must be finished by its deadline. First, at time t, Sys_1 is feasible because the processor utilization factor U = 0.30 $\leq$ 1. We suppose after, that a reconfiguration scenario is applied at time t1 to add 3 new sporadic tasks σ_C, σ_D, and σ_E. The new processor utilization becomes U = 1.21 $>$ 1 time units. Therefore the system is unfeasible. In the following tables (table 1 and table 2), the first column represents the task identifier, the second column represents the release time, the third column represents the deadline of each task which is less than or equal to its period in these examples of real time tasks, the fourth column represents the period and the five column represents the worst case execution time (WCET) of each task.
* P_i is the inter-arrival time.

Table 1: The characteristics of the 5 tasks.

Task	a_i	D_i	$T_i = P_i^*$	C_i
A	0	10	10	2
B	0	20	20	2
C	5	15	-	5
D	5	8	-	4
E	11	12	-	1

Running Example 2:
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 (J. Legrand, 2004) with a task set composed of old tasks (ξ_{old}) and new tasks (ξ_{new}^{p,ψ_h}) on the processor p for each reconfiguration scenario ψ_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 2 and is composed of 10 tasks. The sum of utilization of all tasks is given in table 2 and is equal to 426.1%. 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 τ_1 is partioned on first processor, τ_2 and τ_3 are partitioned on processor 2 while task τ_4 and τ_5 are partitioned on processor 3. We have three sets of local tasks. As there is only one task on first processor then task τ_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 ψ_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. $Feasibility(Current_{\xi}^{\psi_1}(t1)) \equiv False$. Indeed, if the number of tasks increases, then the overload of the system increases too. Our optimal earliest deadline

Table 2: Task Parameters.

Task	C_i	D_i	$T_i = P_i^*$
τ_1	2	9	7
τ_2	3	21	20
τ_3	2	9	9
τ_4	2	13	10
τ_5	3	15	9
τ_6	14	21	19
τ_7	10	24	16
τ_8	8	18	18
τ_9	13	16	17
τ_{10}	5	11	12

first (OEDF) algorithm is based on the following Guarantee Algorithm which is presented by Buttazo and Stankovic in (Buttazzo and Stankovic, 1993). Indeed, OEDF algorithm is an extended and ameliorate version of Guarantee Algorithm that usually guarantee the system's feasibility.

2.2 Guarantee Algorithm

The dynamic, on-line, guarantee test in terms of residual time, which is a convenient parameter to deal with both normal and overload conditions is presented here.
Algorithm GUARANTEE(ξ; σ_a)
begin t = get current time();
R_0 = 0;
d_0 = t;
Insert σ_a in the ordered task linked list;
ξ` = ξ σ_a;
k = position of σ_a in the task set ξ`;
for each task σ_i` such that i $\geq$ k do {
$R_i = R_{i-1} + (d_i - d_{i-1}) - c_i$;
if ($R_i < 0$) then return ("Not Guaranteed");
}
return ("Guaranteed");
end

3 NEW APPROACH WITH DEADLINE TOLERANCE

In this section we will present some preliminaries concepts and we will describe our contribution after. In (Buttazzo and Stankovic, 1993), Buttazo and Stankovic present the Guarantie 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

ξ denotes a set of active sporadic tasks σ_i ordered by increasing deadline in a linked list, σ_1 being the task with the shortest absolute deadline.
a_i denotes the arrival time of task σ_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 σ_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 σ_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 τ_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 σ_i, i.e., the time interval between the arrival time and the absolute deadline. S_i denotes the first start time of task σ_i, i.e., the time at which task σ_i gains the processor for the first time. s_i denotes the last start time of task σ_i, i.e., the last time, before the current time, at which task σ_i gained the processor.
f_i denotes the estimated finishing time of task σ_i, i.e., the time according to the current schedule at which task σ_i should complete its execution and leave the

system.
L_i denotes the laxity of task σ_i, i.e., the maximum time task σ_i can be delayed before its execution begins.
R_i denotes the residual time of task σ_i, i.e., the length of time between the finishing time of σ_i and its absolute deadline. Baruah et al. (S. Baruah and Shasha, 1991) 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:
$d_i = a_i + D_i$ (1)
$L_i = d_i$ - a_i - C_i (2)
$R_i = d_i$ - f_i (3)
f_1 = t + c_1; $f_i = f_{i-1} + c_i \; \forall$ i > 1 (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 $\xi = \{\sigma_1, \sigma_2, ..., \sigma_n\}$ of active sporadic tasks ordered by increasing deadline in a linked list, the residual time R_i of each task σ_i at time t can be computed by the following recursive formula:

$R_1 = d_1$ - t - c_1 (5)
$R_i = R_{i-1} + (d_i - d_{i-1})$ - c_i. (6) (Buttazzo and Stankovic, 1993)

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

$$R_i = d_i - f_i.$$

By the assumption on set ξ, at time t, the task σ_1 in execution and cannot be preempted by other tasks in the set ξ, hence its estimated finishing time is given by the current time plus its remaining execution time:

$$f_1 = \text{t} + c_1$$

and, by equation (3), we have:

$$R_1 = d_1 - f_1 = d_1 - \text{t} - c_1.$$

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

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

and, by equation (3), we have:

$$R_i = d_i - f_i = d_i - f_{i-1} - c_i =$$
$$d_i - (d_{i-1} - R_{i-1}) - c_i = R_{i-1} + (d_i - d_{i-1}) - c_i$$

and the lemma follows.

Lemma 2. A task σ_i is guaranteed to complete within its deadline if and only if $R_i \geq 0$ (Buttazzo and Stankovic, 1993).

Theorem 3. A set $\xi = \{\sigma_i$, i = 1 to m$\}$ of m active sporadic tasks ordered by increasing deadline is feasibly schedulable if and only if $R_i \geq 0$ for all $\sigma_i \in \xi$, (Buttazzo and Stankovic, 1993).
In our model, we assume that the minimum inter-arrival time P_i of each sporadic task is equal to its relative deadline D_i, thus a sporadic task σ_i can be completely characterized by specifying its worst case execution time C_i and its relative deadline D_i. Hence, a sporadic task set will be denoted as follows: $\xi = \{\sigma_i(C_i, D_i)\}$, i = 1 to m.

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 $HP^{(p,\psi_h)} = [\zeta^{(p,\psi_h)}, 2*\text{LCM} + \zeta^{(p,\psi_h)}]$, where LCM^{p,ψ_h} is the well-known Least Common Multiple of tasks periods and (ζ^{p,ψ_h}) is the largest task offset of all tasks τ_k^{p,ψ_h} for each reconfiguration scenario ψ_h on each processor p.
Let $n + m$ be the number of tasks respectively in ξ_{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:

$\sum_{i=1}^{n+m} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} > K$, where K 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 ψ_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 ψ_h.
We propose, for each reconfiguration scenario ψ_h, to add the tasks of ξ_{old} to a linked list $L_{old}^{\psi_h}$ that we sort on the increasing order of their utilization factor values. Let j^{ψ_h} be the first j tasks of $L_{old}^{\psi_h}$.

Approach for each $j^{\psi_h} \in [0, n]$. When we add the first $j^{(p,\psi_h)}$ tasks of $L_{old}^{(p,\psi_h)}$ to $\xi_{new}^{(p,\psi_h)}$, there are

three different solutions exist for the feasibility of the system. In such case, if $j^{\psi_h} = 0$, then no tasks to be added to $\xi_{new}^{\psi_h}$. After each reconfiguration scenario ψ_h, the agent suggests the new parameters for new tasks. After that, the agent selects tasks from linked list which are sorted on their utilization factor ($U_i^{p,\psi_h} = \frac{C_i^{(p,\psi_h)}}{T_i^{(p,\psi_h)}}$) with respect to this order to execute on the processor p. Tasks from linked list are moved to be inserted and executed with new tasks. Whenever an old task is moved from linked list to current ready list composed of new tasks, parameters (WCET, periods/deadlines) are recalculated and presented by the agent to the user for each reconfiguration scenario ψ_h. In this case, addition of old task, neither causes new tasks to miss their deadlines nor misses its own deadline.

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

In the current paper, we suppose that on each processor p, each system $\xi^{(p)}$ can be automatically and repeatedly reconfigured at each reconfiguration scenario ψ_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,ψ_h} and VP_2^{p,ψ_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 ψ_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.

Formalization

We assume in this work a system $\xi^{(p)}$ to be composed of a mixture of $n^{(p)}$ periodic and $m^{(p)}$ sporadic tasks on each processor p. An assumed system $\xi^{(p,\psi_{h-1})} = \{\tau_1^{(p)}, \tau_2^{(p)},...,\tau_n^{(p)}\}$ turns after a reconfiguration scenario to $\xi^{(p,\psi_h)} = \{\tau_1^{(p)}, \tau_2^{(p)},...,\tau_n^{(p)}\ \sigma_{n+1}^{(p)}, \sigma_{n+2}^{(p)},...,\sigma_{n+m}^{(p)}\}$ by considering that m^p new sporadic tasks are added to $\xi^{(p,\psi_{h-1})}$. After each addition, the tasks are logically divided into two subsets. One contains the so called new sporadic tasks which are added to the system, and the rest of tasks taken from $\xi^{(p,\psi_{h-1})}$ are considered as old tasks to form the second subset. After any addition scenario, the response time can be increased and/or some old/new tasks miss their deadlines. When a reconfiguration scenario is automatically applied at run-time, the proposed agent logically decomposes the physical processor of $\xi^{(p,\psi_h)}$ into two virtual processors $VP_1^{(p,\psi_h)}$ and $VP_2^{(p,\psi_h)}$ with different utilization factors $UVP_1^{(p,\psi_h)}$ and $UVP_2^{(p,\psi_h)}$ to adapt the system to its environment with a minimum response time. For more explaining, after any reconfiguration scenario and in order to keep only two virtual processors in the system $\xi^{(p)}$, the proposed intelligent agent automatically merges $VP_1^{(p,\psi_{h-1})}$ and $VP_2^{(p,\psi_{h-1})}$ into $VP_1^{(p,\psi_h)}$ and creates also a new $VP_2^{(p)}$ named $VP_2^{(p,\psi_h)}$, to adapt old and new tasks, respectively. The $VP_2^{(p,\psi_h)}$ is assumed to be a located logical pool in idle periods of $VP_1^{(p,\psi_h)}$.

For example, we assume that k = 1 and we restreint in this case to a uni-processor system, and we have 2 initial tasks τ_1 and τ_2 in an assumed system $sys_1^{p=1}$ with $\xi^{(1,0)} = \xi^{(0)} = \{\tau_1, \tau_2\}$. First, we add $\{\sigma_3, \sigma_4$ and $\sigma_5\}$ to $\xi^{(0)}$ that automatically turns into $\xi^{(\psi_1)} = \{\tau_1, \tau_2, \sigma_3, \sigma_4$ and $\sigma_5\}$. In $\xi^{(1)}$ (h =1), subset $\{\tau_1, \tau_2\}$ is considered as old tasks to be executed by $VP_1^{(\psi_1)}$, whereas subset $\{\sigma_3, \sigma_4$ and $\sigma_5\}$ is considered as new sporadic tasks to be executed by $VP_2^{(\psi_1)}$. $VP_2^{(\psi_1)}$ is located in idle periods of $VP_1^{(\psi_1)}$. We propose thereafter, the arrival of new sporadic tasks σ_6 and σ_7 to be added to $\xi^{(\psi_1)}$ that evolves into $\xi^{(\psi_2)} = \{\tau_1, \tau_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6$ and $\sigma_7\}$. $VP_1^{(\psi_1)}$ and $VP_2^{(\psi_1)}$ are automaticlly merged into $VP_1^{(\psi_2)}$ where subset $\{\tau_1, \tau_2, \sigma_3, \sigma_4$ and $\sigma_5\}$ is considered as old tasks to be executed by this virtual processor. In this case, subset $\{\sigma_6, \sigma_7\}$ is executed by the second newly created virtual processor $VP_2^{(\psi_2)}$ which is located in idle periods of $VP_1^{(\psi_2)}$.

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 ψ_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,ψ_h} for each reconfiguration scenario ψ_h and on each processor p (S2)
By applying equation (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 ψ_h, the following formula must be enforced:
$R_i^{p,\psi_h} \geq 0$ 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 ψ_h and on each processor p (S3)
By applying equation (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: $a_i^{p,\psi_h} \rightarrow a_{inew}^{p,\psi_h} \rightarrow (a_{inew}^{p,\psi_h} = a_i^{p,\psi_h} + \Delta^{p,\psi_h} t)$ for each reconfiguration scenario ψ_h and on each processor p (S4)

By applying equation (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:

$a_{inew}^{p,\psi_h} + D_i^{p,\psi_h} - t - C_i^{p,\psi_h} \geq 0$, where $a_{inew}^{p,\psi_h} = a_i^{p,\psi_h} + \Delta^{p,\psi_h} t$.
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 ψ_h (S5)
$\xi^p = \{\tau_i^p(C_i^p, D_i^p, m_i^p, I_i^p), i = 1 to n^p\}$.
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 ψ_h (S6)
The agent proceeds now as a sixth solution to migrate some tasks of ξ_{new}^{p,ψ_h} and ξ_{old}^p on the processor p for each reconfiguration scenario ψ_h. Indeed, the agent is responsible for allocating the tasks to the K computing processors in an optimal way.

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 ψ_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. 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 figure 1. In order to relocate a task, the intelligent agent notifies the task by means of a migration request signal(1). Whenever

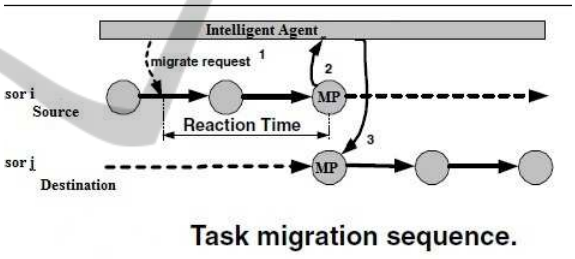

Figure 1: The Task Migration Sequence.

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 ψ_h. In such case of these two reasons, all the relevant state information of that migration point is transferred to the intelligent agent(2). 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(3). Finally, the task resumes execution at the corresponding migration point (MP).

One of the main issues in homogeneous (we suppose before that all the processors are identical) task migration is the overhead incurred by checking for a pending migration request during normal execution (i.e. when there is no pending migration request). Especially since a task requires frequent migration points in order to reduce the reaction time. The reaction time (Figure 1) is the time elapsed between selecting a task for migration and the selected task reaching the next migration point. In order to minimize the checking overhead during normal execution, further denoted as migration initiation, we propose a novel technique for the new generation of embedded systems, that uses the inclusion and exclusion group informations of each task for each reconfiguration scenario ψ_h in order to reduce the area search feasiblity of such systems and to minimize the reaction time and consequently the response time will be minimized too.

• **Solution 7:** Removal of some non critical tasks (to be rejected) for each reconfiguration scenario ψ_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 to $n^p\}$.

m_i^{p,ψ_h} = 1, it tolerates missing deadline,

m_i^{p,ψ_h} = 0, it doesn't tolerate missing deadline,

I_i^{p,ψ_h} = H, Hard task,

m_i^{p,ψ_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($Resp_{k,1}^{p,\psi_h}$, $Resp_{k,2}^{p,\psi_h}$, $Resp_{k,3}^{p,\psi_h}$, $Resp_{k,4}^{p,\psi_h}$, $Resp_{k,5}^{p,\psi_h}$, $Resp_{k,6}^{p,\psi_h}$ 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 based on (A.K.MOK, 1983) is satisfied for each reconfiguration scenario ψ_h:

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

Then, after the reconfiguration scenario ψ_h was applied at run-time to the whole system by the inteligent 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 ψ_h:

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

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

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

Now by adding the following formulas, we have:

$\sum_{i=1}^{(n+m)^{(1,\psi_h)}} \frac{C_i^{(1,\psi_h)}}{T_i^{(1,\psi_h)}} < 1$,

$\sum_{i=1}^{(n+m)^{(2,\psi_h)}} \frac{C_i^{(2,\psi_h)}}{T_i^{(2,\psi_h)}} < 1$,

... < ...

... < ...

$\sum_{i=1}^{(n+m)^{(j,\psi_h)}} \frac{C_i^{(j,\psi_h)}}{T_i^{(j,\psi_h)}} < 1,$

$... \quad < ...$

$... \quad < ...$

$\sum_{i=1}^{(n+m)^{(K,\psi_h)}} \frac{C_i^{(K,\psi_h)}}{T_i^{(K,\psi_h)}} < 1,$

$\Rightarrow \sum_{i=1}^{(n+m)^{\psi_h}} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} < \underbrace{1+1+1+...+1}_{K times}.$

$\Rightarrow \sum_{i=1}^{(n+m)^{\psi_h}} \frac{C_i^{\psi_h}}{T_i^{\psi_h}} < K.$

This work, concentrates on the context of systems containing a set of tasks which is not feasible. The reconfiguration was applied in order not only to obtain the system's feasibility, but also to get the performance of the system by reducing the response time of the processes to be tolerated in interactive environment in order to minimize the response time of the studied reconfigurable embedded system at run-time for each reconfiguration scenario ψ_h and in each processor p.

We can observe that our proposed approach provides an optimal global scheduling algorithm which schedules tasks according to EDF in each processor p for each reconfiguration scenario ψ_h. All tasks meet their deadlines after a reconfiguration scenario ψ_h was applied at run-time. We can also observe, that our proposed algorithm selects tasks to migrate from one processor source i to another processor destination j in an optimal way such that overall utilization of task set is minimum. Parameters of tasks i.e., period, deadline and worst case execution time, are generated randomly. We have illustrated 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(ξ; σ_a)
For each h in [1..M] **Do**
begin t = get current time();
R_0^{p,ψ_h} = 0;
d_0^{p,ψ_h} = t;
Insert σ_a in the ordered task list;
ξ^{p,ψ_h}` = $\xi^{p,\psi_h} \sqcup \sigma_a$;
k = position of σ_a in the task set ξ^{p,ψ_h}`;
for each task σ_i^{p,ψ_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

The extension of the proposed algorithm should be straightforward, when this assumption does not hold and its running time is $O(n+m)^2$. So, Intuitively, we expect that our algorithm performs better than the Buttazo and Stankovic one. 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.

4.1 Experiments

Running Example 1:
We apply our contribution to this first running example on the particular case of uni-processor systems (k = 1) and we could observe that the recalculation points of the utilization factor, when parameters of

new tasks are modified, decreases and becomes less than or equal to 1 and we can deduce that the system is now feasible.

Running Example 2:
We apply our contribution to this second running example and we could observe that the recalculation points of the utilization factor, when parameters of new tasks are modified, decreases and becomes less than or equal to k on the case of multiprocessor systems and we can deduce that the system is now feasible.

4.2 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 (Mani B. Srivastava, 1998). The OEDF scheduling result is shown in figure 2.

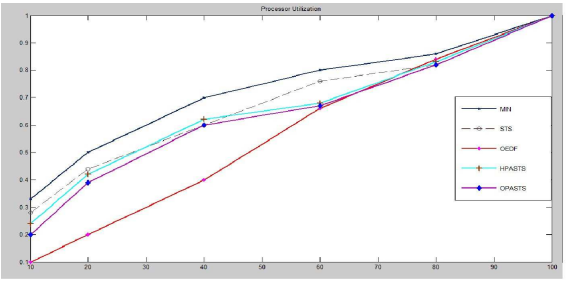

Figure 2: Processor Utilization.

4.3 Discussion

In experiments, if the resulting U(t) $>$ 1, we set U(t) to be 1. We varied the average processor utilization from the light workload (10 tasks) to heavy workload (100 tasks) generated randomly. 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.
Moreover, if the number of solutions presented by the intelligent agent to the user increases, then chances of executing more new added tasks increase and the performance of the real time scheduling is more efficient. This is due to the fact that the reconfiguration issues are increased, the user selects the best solution which gives the minimum utilization factor of the system and ameliorates the response time and hence the chances of executing more new tasks are increased as well.
The agent should define the different solutions for the user. In this case, the user can choose the best solution that satisfies functional requirements. These results were suggested by the tool RT-Reconfiguration and give a feasible system which is proven also by Cheddar.

5 CONCLUSIONS

This paper 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 paper, 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. Finally, our important future work is the generalization of our contributions for the Reconfigurable real-time embedded systems.

REFERENCES

A.K.MOK (1983). *Fundamental Design Problems of Distributed Systems for The HardReal-Time Environment*. Technical Report No. MIT/LCS/TR-297, PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, Massachusetts, USA.

Al-Safi, Y. and Vyatkin, V. (2007). *An ontology-based reconfiguration agent for intelligent mechatronic systems*. Int. Conf. Hol. Multi-Agent Syst. Manuf., vol. 4659, pp. 114-126, Regensburg, Germany, 4th edition.

Brocal V., BalbastreP., B. R. R. L. (2011). *Task period selection to minimize hyperperiod, Emerging Technolo-*

gies & Factory Automation (ETFA), IEEE conference on, pp.1-4, 2011. doi: 10.1109/ETFA.2011.6059178, Toulouse, France, 16th edition.

Buttazzo, G. and Stankovic, J. (1993). *RED: Robust Earliest Deadline Scheduling*. Int. Workshop On Responsive Computing Systems, Austin, 3rd edition.

C. Angelov, K. S. and Marian, N. (2005). *Design models for reusable and reconfigurable state machines*. L.T. Yang et al., Eds., Proc. of Embedded Ubiquitous Comput.

Dertouzos, M. (1974). *Control Robotics: The Procedural Control of Physical Processes*. Proceedings of the IFIP Congress.

H. Gharsellaoui, M. K. and BenAhmed, S. (2012). *Feasible Automatic Reconfigurations of Real-Time OS Tasks*. IGI-Global Knowledge, USA, isbn13: 9781466602946 edition.

J. Legrand, L. (2004). *Cheddar : a Flexible Real Time Scheduling Framework*. ACM SIGAda Ada Letters, volume 24, number 4, pages 1-8. Edited by ACM Press, ISSN:1094-3641.

M. N. Rooker, C. Subder, T. S. A. Z. O. H. and Ebenhofer, G. (2007). *Zero downtime reconfiguration of distributed automation systems: The CEDAC approach*. Int. Conf. Indust. Appl. Holonic Multi-Agent Syst., Regensburg, 3rd edition.

Mani B. Srivastava, Miodrag Potkonjak, I. H. (1998). *On-Line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor*. International Conference on Computer-Aided Design (ICCAD '98), San Jose, California, USA, 8th edition.

P. Balbastre, I. R. and Crespo, A. (2002). *Schedulability analysis of window-constrained execution time tasks for real-time control*. 14th Euromicro Conf. Real-Time Syst., 14th edition.

S. Baruah, G. Koren, B. M. A. R. L. R. and Shasha, D. (1991). *On-line Scheduling in the Presence of Overload*. IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico.

T. Tia, J. W.-S. Liu, J. S. a. R. H. (1994). *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.

West, R. and Schwan, K. (1999). *Dynamic window-constrained scheduling for multimedia applications*. IEEE 6th Int. Conf. Multi. Comput. Syst., 6th edition.

X. Wang, M. K. and L, Z. W. (2011). *dynamic low power reconfigurations of real-time embedded systems*. in: Proc. Pervas. Embedded Comput. Commu. Syst., Algarve, Portugal, 1st edition.