OEDF: Optimal Earliest Deadline First Preemptively Scheduling for Real-Time Reconfigurable Sporadic Tasks

Hamza Gharsellaoui\textsuperscript{1}, Mohamed Kalgui\textsuperscript{1,2}, Samir Ben Ahmed\textsuperscript{3}
\textsuperscript{1}INSAT Institute - University of Carthago, Tunisia
\textsuperscript{2}ITIA Institute - CNR Research Council, Italy
\textsuperscript{3}FST Faculty - University of Tunis El Manar, Tunisia

ABSTRACT
This paper deals with the problem of scheduling the mixed workload of both uniprocessor 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. 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, or in the worst case by removing some soft tasks according to predefined 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 addressing new criteria such as flexibility and agility [7]. 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) \( C_i \), an offset (starting time) \( a_i \), a period \( T_i \) and a deadline \( D_i \). The general goal of this paper is to be reassured that any reconfiguration scenario 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 [7]. 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 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 \( P_i \) which is assumed to be equal to its relative deadline \( D_i \), and a worst-case execution time (WCET) \( C_i \). 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. The off-line requests support the normal func-
tions 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 and on the dynamic reconfiguration in order to obtain the feasibility of the system at run time, meeting real-time constraints and for the optimization of response time of this system. Indeed, many real-time systems rely on the EDF scheduling algorithm. 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 [3]. This algorithm assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks \( hp = [0, 2^k \text{LCM} + \text{max} (a_1, 1)] \) where LCM is the well-known Least Common Multiple of all task periods and \((a_1, 1)\) is the earliest activation time of each task \( \tau_k \). 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 \( R_i \) before and after each addition scenario and calculates the minimum of those proposed solutions in order to obtain \( Resp_k \) optimal noted \( Resp_k^{opt} \). Where \( Resp_k^{opt} \) is the minimum of the response time of the current system under study given by the following equation: 
\[
Resp_k^{opt} = \min(Resp_k, 1, Resp_k, 2, Resp_k, 3, Resp_k, 4, Resp_k, 5, Resp_k, 6) .
\]
To calculate these previous values \( Resp_k, 1, Resp_k, 2, Resp_k, 3, Resp_k, 4, Resp_k, 5, \) and \( Resp_k, 6 \), we proposed a new theoretical concepts \( R_i, S_i, s_i, f_i \) and \( L_i \) for the case of real-time sporadic operating system (OS) tasks. Where \( R_i \) is the residual time of task \( \sigma_i \), \( S_i \) denotes the first start time of task \( \sigma_i \), \( s_i \) is the last start time of task \( \sigma_i \), \( f_i \) denotes the estimated finishing time of task \( \sigma_i \), and \( L_i \) denotes the laxity of task \( \sigma_i \).

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 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. According to [7], 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 [2], 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 [7], [4]. 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 \( O(n + m) \) [11].

To illustrate the key point of the proposed dynamically approach, we define a new real-time embedded control system in the study \( \xi = Sys_1 \cup Sys_2 \), where \( Sys_1 \) is a set of \( n \) periodic tasks, i.e., \( Sys_1 = \{ \tau_1, \tau_2, \ldots, \tau_n \} \) and \( Sys_2 \) is a set of \( m \) active sporadic tasks \( \sigma_i \) ordered by increasing deadline in a linked list, i.e., \( Sys_2 = \{ \sigma_1, \sigma_2, \ldots, \sigma_m \} \). \( \sigma_1 \) being the task with the shortest absolute deadline.

2.1 STATE OF THE ART

Nowadays, several interesting studies have been published to develop reconfigurable embedded control systems. In [5] Marian et al. propose a static reconfiguration technique for the reuse of tasks that implement a broad range of systems. The work in [6] proposes a methodology based on the human intervention to dynamically reconfigure tasks of considered systems. In [8], an ontology-based agent is proposed by Vyatkin et al. to perform systems reconstructions according to user requirements and also the environments evolution. Window-constrained scheduling is proposed in [9], which is based on an algorithm named dynamic window-constrained scheduling (DWCS). The research work in [10] 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 [10], 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. Old and new tasks are located in different windows to schedule the system with a minimum response time. In [4], 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 [7] 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 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:

To illustrate the key point of the proposed dynamic reconfiguration approach, we consider $\xi = S_{\xi_1} \cup S_{\xi_2}$ a set of 5 characterized tasks, shown in Table 1 as a motivational example. $S_{\xi_1} = \tau_A, \tau_B$, and $S_{\xi_2} = \sigma_C, \sigma_D,$ and $\sigma_E$. $\tau_A$ and $\tau_B$ are periodic tasks and all the rest ($\sigma_C, \sigma_D,$ and $\sigma_E$) are sporadic tasks. Each task can be executed immediately after its arrival and must be finished by its deadline. First, at t time unit, $S_{\xi_1}$ is feasible because the processor utilization factor $U = 0.30 \leq 1$. We suppose after, that a reconfiguration scenario is applied at t1 time units to add 3 new sporadic tasks $\sigma_C, \sigma_D,$ and $\sigma_E$. The new processor utilization becomes $U = 1.21 > 1$ time units. Therefore the system is unfeasible.

<table>
<thead>
<tr>
<th>Task</th>
<th>$a_i$</th>
<th>$D_i$</th>
<th>$T_i = D_i + a_i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>15</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>D</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>11</td>
<td>12</td>
<td>23</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: The characteristics of the 5 tasks used to illustrate the motivation for dynamic reconfiguration approach

* $P_i$ is the inter-arrival time.

Our optimal earliest deadline first (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($\xi; \sigma_a$)

begin $t =$ get current time();
$d_0 = t$;
Insert $\sigma_a$ in the ordered task linked list;
$\xi^* = \xi \cup \sigma_a$;
$k = \text{position of } \sigma_a$ in the task set $\xi^*$;
for each task $\sigma_i$ 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 [2], 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. \( S_1 \) 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. [1] 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's 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:

\[
d_i = a_i + D_i \quad (1)
\]
\[
L_i = d_i - a_i - C_i \quad (2)
\]
\[
R_i = d_i - f_i \quad (3)
\]
\[
f_i = t + c_i; \quad f_i = f_{i-1} + c_i \quad \forall i > 1 \quad (4)
\]

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 system.

**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 \( \sigma_i \) at time \( t \) can be computed by the following recursive formula:

\[
R_1 = d_1 - t - c_1 \quad (5)
\]
\[
R_i = R_{i-1} + (d_i - d_{i-1}) - c_i \quad (6) \quad [2]
\]

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

\[
R_i = d_i - f_i.
\]

By the assumption on set \( \xi \), at time \( t \), the task \( \sigma_i \) 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:

\[
f_i = t + c_i
\]

and, by equation (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 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 \( \sigma_i \) is guaranteed to complete within its deadline if and only if \( R_i \geq 0 \) [2].

**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 \( \sigma_i \in \xi \), [2].

In our model, we assume that the minimum interarrival time \( P_i \) of each sporadic task is equal to its relative deadline \( D_i \), thus a sporadic task \( \sigma_i \) 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: \( \xi = \{\sigma_i(C_i, D_i)\}, i = 1 \text{ to } m \).

### 3.2 CONTRIBUTION: AN ALGORITHM FOR FEASIBILITY TESTING WITH RESPECT TO SPORADIC TASK SYSTEMS

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

*For example, after the first addition scenario, \( \xi^{(0)} \) turns into \( \xi^{(1)} \), \( \xi^{(1)} \) is automatically decomposed into \( VP_1^{(1)} \) and \( VP_2^{(1)} \) for old and new tasks with the processor utilization factors \( UP_1^{(1)} \) and \( UP_2^{(1)} \) respectively.*

**Formalization**

We assume in this work a system \( \xi \) to be composed of a mixture of \( n \) periodic and \( m \) sporadic tasks. An assumed system \( \xi^{(i-1)} = \{\tau_1, \tau_2, ..., \tau_n\} \) turns after a
reconfiguration scenario to \( \xi^{(i)} = \{\tau_1, \tau_2, \ldots, \tau_n, \sigma_{n+1}, \sigma_{n+2}, \ldots, \sigma_m\} \) by considering that \( m-n \) new sporadic tasks are added to \( \xi^{(i-1)} \). 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 \( \xi^{(i-1)} \) 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 \( \xi^{(i)} \) into two virtual processors \( V P_1^{(i)} \) and \( V P_2^{(i)} \) with different utilization factors \( U V P_1^{(i)} \) and \( U V P_2^{(i)} \) 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 \( \xi \), the proposed intelligent agent automatically merges \( V P_1^{(i-1)} \) and \( V P_2^{(i-1)} \) into \( V P_1^{(i)} \) and creates also a new \( V P_2 \) named \( V P_2^{(i)} \), to adapt old and new tasks, respectively. The \( V P_2^{(i)} \) is assumed to be a located logical pool in idle periods of \( V P_1^{(i)} \).

For example, we have 2 initial tasks \( \tau_1 \) and \( \tau_2 \) in an assumed system \( S_{sys} \) with \( \xi^{(0)} = \{\tau_1, \tau_2\} \). First, we add \( \{\sigma_1, \sigma_2 \text{ and } \sigma_3\} \) to \( \xi^{(0)} \) that automatically turns into \( \xi^{(1)} = \{\tau_1, \tau_2, \sigma_3, \sigma_4 \text{ and } \sigma_5\} \). In \( \xi^{(1)} \), subset \( \{\tau_1, \tau_2\} \) is considered as old tasks to be executed by \( V P_1^{(1)} \), whereas subset \( \{\sigma_3, \sigma_4 \text{ and } \sigma_5\} \) is considered as new sporadic tasks to be executed by \( V P_2^{(1)} \). \( V P_2^{(1)} \) is located in idle periods of \( V P_1^{(1)} \). We propose thereafter, the arrival of new sporadic tasks \( \sigma_6 \) and \( \sigma_7 \) to be added to \( \xi^{(1)} \) that evolves into \( \xi^{(2)} = \{\tau_1, \tau_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6 \text{ and } \sigma_7\} \). \( V P_1^{(1)} \) and \( V P_2^{(1)} \) are automatically merged into \( V P_1^{(2)} \) where subset \( \{\tau_1, \tau_2, \sigma_3, \sigma_4 \text{ and } \sigma_5\} \) 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 \( V P_2^{(2)} \) which is located in idle periods of \( V P_1^{(2)} \).

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 or to remove some soft tasks as following:

- **Solution 1**: Moving some arrival tasks to be scheduled in idle times. (Idle times are caused when some tasks complete before its worst case execution time) (S1)

- **Solution 2**: maximize the \( d_i \) (S2)

By applying equation (3) that notices: \( R_i = d_i - f_i \), we have: \( R_i = d_i - t - C_i \).

Or, to obtain a feasible system after a reconfiguration scenario, the following formula must be enforced: \( R_i \geq 0 \).

By this result we can write: \( d_{\text{new}} - t - C_i \geq 0 \), where \( d_{\text{new}} = d_i + \theta_i \).

So, \( d_i + \theta_i - t - C_i \geq 0 \Rightarrow \theta_i \geq t + C_i - d_i \).

- **Solution 3**: minimize the \( c_i \) (S3)

By applying equation (3) that notices: \( R_i = d_i - f_i \), we have: \( R_i = d_i - t - C_i \).

By this result we can write: \( d_i - t - C_{\text{new}} \geq 0 \), where \( C_{\text{new}} = C_i + \beta_i \).

So, \( d_i - t - C_i - \beta_i \geq 0 \Rightarrow d_i - t - C_i \geq \beta_i \)

\[ \Rightarrow \beta_i \leq d_i - t - C_i \]

- **Solution 4**: Enforcing the starting time to come back: \( a_i \rightarrow a_{\text{new}} \rightarrow (a_{\text{new}} = a_i + \Delta t) \) (S4)

By applying equation (1) that notices: \( d_i = a_i + D_i \), we have:

\[ R_i = a_i + D_i - t - C_i \]

Or, to obtain a feasible system after a reconfiguration scenario, the following formula must be enforced: \( R_i \geq 0 \).

By this result we can write:

\[ d_i - t - C_i \geq 0, \text{ where } a_{\text{new}} = a_i + \Delta t. \]

So, we obtain: \( a_i + \Delta t + D_i - t - C_i \geq 0 \).

\[ \Rightarrow \Delta t \geq t + C_i - a_i - D_i. \]

- **Solution 5**: Tolerate some non-critical Tasks in among \( n (m,n) \) firm (for a reasonable cost) (S5)

\[ \xi = \{\tau_i(C_i, D_i, m_i, I_i), i = 1, m,n\}. \]

\( m_i = 1 \), it tolerates missing deadline, \( m_i = 0 \), it doesn’t tolerate missing deadline, \( I_i = H \), Hard task, \( I_i = S \), Soft task,

\[ \Rightarrow \Delta t \geq t + C_i - a_i - D_i. \]

- **Solution 6**: Removal of some non-critical tasks (to be rejected) (S6)

\[ \xi = \{\tau_i(C_i, D_i, m_i, I_i), i = 1, m,n\}. \]

\( m_i = 1 \), it tolerates missing deadline, \( m_i = 0 \), it doesn’t tolerate missing deadline, \( I_i = H \), Hard task, \( I_i = S \), Soft task,

For every solution the corresponding response time is: \( Resp_{k,1} \) = the response time calculated by the first
solution,
Resp_k,2 = the response time calculated by the second solution,
Resp_k,3 = the response time calculated by the third solution,
Resp_k,4 = the response time calculated by the fourth solution,
Resp_k,5 = the response time calculated by the fifth solution,
Resp_k,6 = the response time calculated by the sixth solution.

We define now, Resp_k optimal noted Resp^opt_k according to the previous three solutions calculated by the intelligent Agent (Solution 1, Solution 2, Solution 3, Solution 4, Solution 5 and Solution 6) by the following expression:
Resp^opt_k = \min(Resp_k,1, Resp_k,2, Resp_k,3, Resp_k,4, Resp_k,5, and Resp_k,6) (the minimum of the six values). So, the calculation of Resp^opt_k allows us to obtain and to calculate the minimizations of response times values and to get the optimum of these values.

3.3 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)
begin
   t = get current time();
   R_0 = 0;
   d_0 = t;
   Insert \sigma_a in the ordered task list;
   \xi' = \xi \cup \sigma_a;
   k = position of \sigma_a in the task set \xi';
   for each task \sigma_i such that i \geq k do {
      \begin{align*}
      R_i &= R_{i-1} + (d_i - d_{i-1}) - c_i; \\
      \text{if} (R_i \geq 0) \text{ then } &\text{return ("Guaranteed");}
      \end{align*}
   }
   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 !!");
   \end{align*}
end

This algorithm assumes that sporadic tasks span no more than one hyperperiod of the periodic tasks hp = [0, 2*LCM+max_k(a_k,1)], where LCM is the well-known Least Common Multiple of all task periods and (a_k,1) is the earliest activation time of each task \tau_k [7]. We use their technique for acceptance test. The extension of the proposed algorithm should be straightforward, when this assumption does not hold and its running time is O(n + m) [11]. 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 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 the both four techniques, you can see [12].
The OEDF scheduling result is shown in figure 1.

4.2 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 those 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 CONCLUSION AND FUTURE WORKS

This paper deals with reconfigurable systems to be implemented by a hybrid system 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 uniprocessor systems and proven it correct. Finally, our important future work is the generalization of our contributions for the Reconfigurable real-time embedded systems.

References


