REAL-TIME TASK RECONFIGURATION IN ENERGY-HARVESTING BASED MULTIPROCESSOR SYSTEMS

Wiem Housseyni, Olfa Mosbahi, Mohamed Khalgui, Maryline Chetto

REAL-TIME TASK RECONFIGURATION IN ENERGY-HARVESTING BASED MULTIPROCESSOR SYSTEMS

Wiem Housseyni,
Olfa Mosbahi,
Mohamed Khalgui
National Institute of Applied Sciences and Technology,
Institute-University of Carthago, Tunisia
Email:
{wiem.housseyni,olfamosbahi,khalgui.mohamed}@gmail.com

Maryline Chetto
University of Nantes IRCCyN UMR CNRS
6597 44321 Nantes, France
Email:maryline.chetto@univ-nantes.fr

KEYWORDS

Real-Time Scheduling, Multiprocessor Architecture, Renewable Energy, Dynamic Reconfiguration, DAG, Degraded Execution.

ABSTRACT

This paper deals with real-time scheduling in a reconfigurable multiprocessor system. Each processor is supplied with a renewable energy source, and uses a rechargeable energy storage. A reconfiguration scenario is any operation that consists in the addition, removal or update of tasks which may result in timing unfeasibility. In our work, each task is represented by a Probabilistic Directed Acyclic Graph (DAGP). Our contribution in this paper concerns the online feasibility problem that is issued from the reconfiguration process. We describe new approaches to solve this problem: the first one changes the timing parameters of the DAGP, the second one decomposes and migrates branches of the DAGP, the third one modifies the scheduling mode, and the fourth one deletes some tasks.

INTRODUCTION

Distributed embedded systems are widely used in real-time domains such as biomedicine, automobile, aircraft and industrial areas. In real-time systems, not only the treatments should be correct but they have to produce outputs in bounded time (Stankovic 1988). For soft real-time systems, missing some deadlines may be tolerable. In contrast, for hard real-time systems, non respect of timing requirements may lead to severe consequences (Liu and Layland 1973, Burns 1991, Manacher 1967).
Batteries are the dominant energy source for embedded real-time systems. However, in addition to their negative impact on the environment, their use may be troublesome due to their limited energy storage capacity and their finite useful live. For some application domains, replacing battery is either costly or impractical. Hence, these embedded systems should be designed to operate incessantly and be autonomous through renewable energy sources. Several technologies are proposed for environmental energy harvesting, in particular solar and vibrational energies. Energy harvesting emerges as a promising technology to surmount energy limitation and enhance system lifetime. In an energy harvesting based system, the main issue is to guarantee that at any moment the system does not consume more energy than harvested and available in the storage.
In this paper, we consider an energy harvesting based real-time system which is composed of a set of identical processors. Preemption and migration of tasks are authorized on every processor. The hardware platform is supplied with renewable energy sources. The harvested energy is stored in rechargeable energy reservoirs with limited capacity (e.g. batteries / supercapacitors).
A reconfigurable computing system undergoes unpredictable events that require adequate online decisions so as to maintain schedulability of the application software. Reconfiguration should be performed whenever a task needs to be added, removed or replaced or a task needs to modify its timing parameters for applicative motivations. The problem of real-time task scheduling on a reconfigurable monoprocessor architecture has received substantial attention (George et al. 2005, Camponogara et al. 2010, Gharsellaoui et al. 2012). In contrast, only few works deal with multiprocessor counterparts. Furthermore, considerable studies consider the problem of minimizing energy consumption so as to maximize system autonomy (Wang et al. 2015, Chniter et al. 2014). But they do not address the energy neutrality problem that characterizes energy harvesting based computing systems.
Prior works on real-time scheduling in energy harvesting systems addressed basic models where tasks are independent from each other. And they mainly focus on monoprocessor architectures even with DVFS facilities. However, these results cannot be simply extended to tasks which are represented by directed acyclic graphs (DAGs) in a distributed context. Consequently, both static task assignment and dynamic task migration in multiprocessor energy harvesting systems are new challenges. We will address these two issues. Firstly, we

will contribute with proposition of a more realistic task model based on DAG. Secondly, we will provide new solutions for task assignment and dynamic reconfiguration. In our model, every task τ_i can be viewed at two distinct levels: i) at the first one, τ_i is a computation box with given period T_i, worst case execution time C_i, relative deadline D_i and worst case energy consumption En_i, ii) at the second level, τ_i is represented by a probabilistic directed acyclic graph called Probabilistic DAG (DAGP). The real-time simulator Cheddar (Singhoff et al. 2004) permits to verify the systems behavior and evaluate the performance of our different approaches. Simulation results bring to light the effectiveness of the proposed DAGP model and reconfiguration strategies measured in terms of deadline miss ratio and energy savings.
The remainder of the paper is as follows. Section II gives a brief state of the art about both energy aware scheduling and directed acyclic graph based models. Section III formalizes the scheduling problem. Section IV presents the new task model, namely Probabilistic DAG (DAGP) and the terminology used throughout this paper. Section V describes our solutions for reconfiguration in real-time energy harvesting based multiprocessor systems. Section VI reports simulation results that bring to light the effectiveness of these solutions measured in terms of deadline miss ratio. And finally we conclude in Section VII with a summary of our contributions.

RELATED WORKS

In this section, we present a state of the art successively about real-time scheduling under energy harvesting constraints, reconfigurable real-time systems and scheduling of DAG tasks.

Real-time Scheduling and Energy Harvesting Considerations

For about ten years, several important works have focused on scheduling in uniprocessor energy harvesting based embedded systems. In (Moser et al. 2006), the Lazy Scheduling Algorithm (LSA), based on the EDF (Earliest Deadline First) rule is proved to be optimal for periodic or aperiodic tasks with deadlines. In (Chetto 2014), another optimal scheduling algorithm, called ED-H is proposed, based on the EDF rule, with less restrictive hypotheses than LSA. Energy harvesting aware scheduling for the multiprocessor case has received much less attention. Among the most interesting studies, Abdallah et Al in (Abdallah et al. 2014) describe and evaluate real-time task assignment heuristics for optimizing the global deadline success ratio. All of these studies consider independent and modular task execution models.

Reconfigurable Real-Time Systems

Several interesting academic and industrial works focused on reconfigurable systems where automatic reconfigurations are applied by intelligent agents (Khalgui 2010). More recently, studies dealt with the same issues with low-power considerations. In (Chniter et al. 2014), two combinatorial optimization approaches based on DVFS processors are described for minimizing energy consumption. A mechanism adjusts deadlines so as to guarantee feasibility conditions and overcome the problem of task rejection. In (Wang et al. 2015), a software-agent-based architecture provides four solutions to reconfigure the system at run-time. In addition, the agent provides three virtual processors in order to reduce the systems power consumption. However, no work deals with energy harvesting assumptions.

DAG Scheduling

Scheduling tasks modeled by DAGs has received meaningful efforts for parallelization objectives. Saifullah et al. (Saifullah et al. 2014) propose a method that eliminates inter-task dependencies so as a DAG task be transformed into a collection of independent sequential threads. The global earliest deadline first (GEDF) schedulability test is then applied to the resulting set of threads. In (Bonifaci et al. 2013), Bonifaci considers the general parameters DAG tasks synchronization regardless of their internal structures. Only two parameters related to the execution of the task model are defined: the total execution time, and the critical-path length. Recently in (Fonseca et al. 2015), the authors propose a multi-DAG model which facilitates the schedulability analysis of parallel tasks with multiple execution flows on multi-core architectures. All of these works change the nature of a DAG, and transform it into a collection of segments. Nevertheless, no one of these works consider DAGs with probabilities.
To our knowledge, no previous study has addressed the scheduling problem where tasks have to be executed with possible migration in a multiprocessor architecture with regenerative energy. As far as we know, the Probabilistic DAG model is proposed for the first time in this paper.

FORMALIZATION OF RECONFIGURABLE REAL-TIME SYSTEMS

In this section, we describe our system model that consists of a reconfigurable distributed real-time system, a harvesting energy module and an energy storage module.

Real-Time Reconfigurable System

In this paper, we are interested in dynamic task reconfiguration. An external reconfiguration event is defined as an exception or an event that leads to add/remove/update tasks. Consequently, any reconfiguration scenario may increase energy consumption and/or make some tasks to violate their deadlines. We assume that a reconfiguration scenario may affect only one processor at a given time. The system model is depicted in Figure 1.

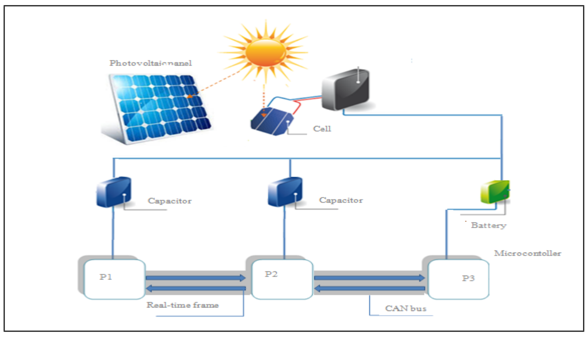

Figure 1: System model

Notation: We consider a real-time embedded system to be formalized as Sys $= \{H_w, S_w\}$ such that H_w is the hardware platform and S_w is the software one. H_w contains a set π of m processors $\pi = \{P_1, ..., P_m\}$. We assume that preemption and migration of tasks are authorized on all processors. The multiprocessor platform is supplied by a renewable energy source and uses a set $\beta = \{B_1, ..., B_m\}$ of m rechargeable energy storage with limited capacities. Each processor P_j in the multiprocessor platform is powered by its own storage unit denoted by B_j. The software platform S_w contains a set ψ of N tasks $\psi = \{\tau_1, ..., \tau_N\}$. Each task is assigned to a given processor according to a technique detailed afterwards. We suppose that $S_w(t)$ is the task set that implements the system *Sys* at a particular time *t*. We denote by P_f a faulty processor (i.e. where tasks have to migrate) and ψ_{P_f} the set of tasks assigned to P_f. Furthermore, in the rest of the paper a subscript "f" represents an item in a faulty processor and we denote by "$|A|$" the cardinality of the set A. In our system, a task τ_i, $i = \{1, .., \text{N}\}$, is characterized by:

- period T_i, worst case execution time (WCET) C_i in conformance with the classical task model of Liu and Layland (Liu and Layland 1973), worst case energy consumption (WCEC) En_i and a degree of criticality dc_i that defines its applicative importance.

- a Probabilistic Directed Acyclic Graph (DAGP) that encodes all its possible execution traces (details are given in section IV.)

The system *Sys* can be reconfigured repeatedly. After any external reconfiguration scenario at a particular time *t* such as addition or removal of tasks, the new implementation is defined by:

$$S_w^+(t) = S_w^-(t) \cup \xi^+/\xi^-,$$

where $\xi^+(t) \subseteq S_w$ is the set of added tasks and $\xi^-(t) \subseteq S_w$ is the set of tasks removed from Sys.

Energy Model

Energy Production Model
Let us assume that the harvesting energy is collected from different energy sources (photovoltaic, piezoelectric, thermal, ...). We suppose that the incoming power received by a given storage unit remains unchanged along time but it may be different from one storage unit to another. Let $P_h(t)$ be the instantaneous charging rate produced by the energy source at time t and *E(t)* be the total energy produced over [0, t] by the power source given by the following formula:

$$E(t) = \int_0^t P_h(t)\,\mathrm{d}t \qquad (1)$$

Energy Storage Model
In this paper, we consider a hybrid energy storage model (battery and super-capacitor). A storage unit is defined by $B_j = (E_{jmax}, E_{jmin})$, $\forall j = \{1, ..., m\}$ where E_{jmax} and E_{jmin} are respectively minimal energy and maximal energy that can be stored. Note that for simplicity and without loss of generality, we assume that the energy storage can be completely depleted to as less as zero. The energy available in the storage B_j at time t is denoted by $E_{Bj}(t)$. We also assume that each energy storage can be charged up to its capacity. Initially, every energy storage is fully-charged: $\forall\ j = \{1..m\}$, $E_{Bj}(0) = E_{jmax}$.
Energy Consumption Model
We assume that the energy consumed by the processor P_j is equal to zero when it does not execute jobs i.e. when it is in the idle state. Furthermore, the energy consumed in the time interval $[t_1, t_2]$ is the cumulative amount of WCEC of tasks which execute on processor P_j between t_1 and t_2. The energy consumed by a job in any unit time-slot is no less than the energy produced in the same unit time-slot.

Initial Task Assignment

The so-called Best Fit Energy Heuristic (BFEH) is proposed for initial task assignment. This partitioning issue amounts to a Bin-Packing one which is known to be NP-hard (Michael and David 1979). Let us define ψ as a set of n periodic real-time tasks and π as a set of m processors. Each task in ψ is assigned to one processor of π as follows:

1. ψ is sorted in decreasing order of the ratios given by worst case energy consumption WCEC, En_i divided by deadline, D_i.
2. π is sorted in decreasing order of the energy storage capacity, $E_{Bj}(t)$.

Task τ_i is assigned to processor P_j if the EDF schedulability test is satisfied and the storage B_j has the least residual energy. Ideally, the residual energy in the storage should be zero. We consider that all the jobs of a given task should be executed on the same processor.

PROBABILISTIC TASK DAG MODEL

In this section, we introduce a new task model called probabilistic directed acyclic graph (DAGP) attached to each task in the software platform S_w.

Motivation

Typically, the code of any task is constructed from one or more control structures such as the "if-then-else" statements. Two jobs $\tau_{i,h}$ and $\tau_{i,k}$ of task τ_i may execute different parts of the code. Hence, an "execution flow" is defined as the path used by a job throughout its execution. In real world, the various possible execution flows do not have the same chance to be used. A probability is attached to execution flow so as to express the chance or the risk that a job performs this execution flow.

Formalization

A DAGP model is attached to every task τ_i $i = \{1..N\}$ as depicted in Figure 2. This model is a graph $G_i = (V_i, E_i)$ where $V_i = \{\tau_{i,1}, ..., \tau_{i,n_i}\}$ is the set of task nodes that represent the sub-tasks of τ_i. n_i is the number of sub-tasks in G_i, and E_i is the set of directed edges that represent dependencies between nodes in the graph G_i. Each edge is labeled by constant p which designates the inter-arrival period. The inter-arrival time p between $\tau_{i,k}$ and $\tau_{i,h}$ is defined as the amount of time that must elapse after the execution of $\tau_{i,k}$ and before the task $\tau_{i,h}$ can be triggered. We suppose that p is the same among all subtasks. We assume that DAGP G_i attached to task τ_i $i = \{1..N\}$ is characterized by a set $\mathcal{F}_i = \{F_{i,1}, ..., F_{i,m_i}\}$ which denotes the set of all possible execution flows of G_i. m_i is the number of execution flows in G_i. Each execution flow $F_{i,j} = \{Pr_{i,j}, V_{i,j}, E_{i,j}\}$ is characterized by a probability $Pr_{i,j}$, a set of nodes $V_{i,j}$ and a set of edges $E_{i,j}$. We associate a dc_i attribute to each task, that defines its execution emergency. Task τ_i $\forall i = \{1..N\}$ is characterized by sextuplet $(G_i, C_i, D_i, T_i, E_{ni}, dc_i)$, where i) G_i the DAGP associated to task τ_i, ii) C_i the maximum number of CPU clock cycles needed to complete a job instance of the task τ_i called WCET iii) the relative deadline of the

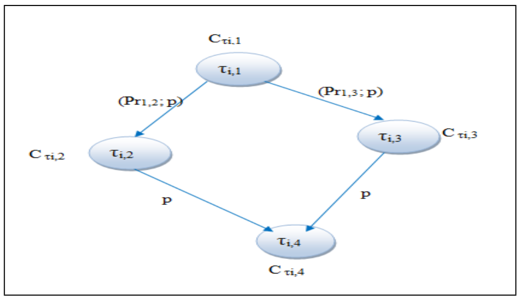

Figure 2: Probabilistic DAG task's model

task, D_i, iv) T_i is the period of the task. We assume that D_i is equal to T_i, v) En_i is the energy requirement of the task τ_i, the amount of energy required for its execution, called WCEC and vi) dc_i the execution emergency level of the task τ_i. We introduce in the following the system model terminology.

Definition 1. The utilization factor of task τ_i is denoted by U_i and defined as follows:

$$U_i = \frac{C_i}{T_i}$$

The CPU utilization of processor P_j, $j = \{1..m\}$ is denoted by:

$$U_{P_j} = \sum_{i=1}^{n} U_i \tag{2}$$

where n is the number of tasks assigned to processor P_j.

Definition 2. The set of n tasks assigned to processor P_j is schedulable under a scheduling policy if the processor utilisation factor U is no greater than the schedulable utilisation U_x as denoted by:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq U_x \tag{3}$$

where:

$$U_x = \begin{Bmatrix} n(2^{\frac{1}{n}} - 1)\ if\ RM\ is\ applied \\ 1\ if\ EDF\ is\ applied \end{Bmatrix}$$

Definition 3. The amount of energy consumed by processor P_j $j = \{1..m\}$ is defined as the sum of WCEC En_i of all n tasks assigned to P_j:

$$E_{p_j} = \sum_{i=1}^{n} En_i \tag{4}$$

Definition 4. The WCET of an execution flow $F_{i,j}$ of task τ_i i = {1 .. N} is defined as the cumulative amount of WCET of all nodes of the set $V_{i,j}$ plus the inter-arrival period p:

$$C_{F_{i,j}} = \sum_{\tau_{i,k} \in V_{i,j}} C_{\tau_{i,k}} + p \times (|V_{i,j}| - 1) \tag{5}$$

The critical execution flow $F^c_{i,j}$ of task τ_i is defined as the execution flow with the longest execution time. **Definition 5.** The WCET of task τ_i is defined as the length of the critical path as follows:

$$C_i = max(C_{F_{i,j}}), j = \{1..m_i\} \qquad (6)$$

In the context of the DAGP model, we introduce the notion of probabilistic utilisation factor $U_{Pr_{\tau_{i,k}}}$ which denotes the utilisation factor of sub-task $\tau_{i,k}$ with probability P_r.

Definition 6. According to Shin and Choi (Shin and Choi 1999) the utilisation factor of sub-task $\tau_{i,k}$ is denoted by:

$$U_{Pr_{\tau_{i,k}}} = Pr \times \frac{C_{\tau_{i,k}}}{T_{\tau_{i,k}}} \qquad (7)$$

Definition 7. The workload $U_{F_{i,j}}$ of execution flow $F_{i,j}$ of task τ_i is defined as the cumulative amount of utilization for all sub-tasks in $V_{i,j}$:

$$U_{F_{i,j}} = \sum U_{Pr_{\tau_{i,j}}}, \tau_{i,j} \in V_{i,j} \qquad (8)$$

Definition 8. The energy, $E_{F_{i,j}}$, consumed during the execution flow $F_{i,j}$ is denoted by:

$$E_{F_{i,j}} = K \times U^2_{F_{i,j}}, K = C \times V^2 \times F \qquad (9)$$

where C is a constant that depends on the processor identity. We denote respectively by F and V the frequency and voltage of the system.

Definition 9. The worst case energy consumption En_i of a task τ_i is defined as follows:

$$En_i = Max(E_{F_{i,j}}), j = \{1..m_i\} \qquad (10)$$

CASE STUDY

Let us consider a real-time embedded system Sys such that $H_w = \{P_1, P_2, P_3\}$ where each processor supports only one operating frequency. We assume that task preemption and migration are authorized on all processors. The harvested energy is stored in a set of three batteries / super-capacitors $\beta = \{B_1, B_2, B_3\}$ where P_1 is supplied from $B_1 = 45$, P_2 supplied from $B_2 = 100$ and P_3 supplied from $B_3 = 40$. The software platform initially contains a set of three periodic tasks τ_1, τ_2, τ_3 depicted in Table 1. In this case study, the EDF policy is applied for the three processors. Initially, the task set is assigned to the multiprocessor platform according to the Best Fit energy heuristic (BFEH) presented previously. τ_1, τ_2 and τ_3 are assigned to processor P_1, P_2 and P_3 respectively. Due to the Cheddar implementation, the feasible scheduling result of the system Sys is shown in Figure 9. The system is feasible since the CPU loads of the multiprocessor platform are U_{p_1} = 0,55, $U_{p_2} = 0.56$ and $U_{p_3} = 0.4$. The energy consumptions are $E_{P_1} = 7$, $E_{P_2} = 13$ and $E_{P_3} = 5$ energy units.

Table 1: Initial System Configuration

Task	C_i	T_i	D_i	En_i	d_{c_i}	(m,k)-firm
τ_1	11	20	20	7	A	(1,2)
τ_2	9	16	16	13	B	(1,2)
τ_3	10	25	25	5	C	(1,2)

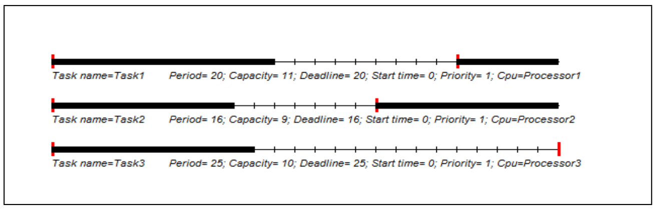

Figure 9: Initial schedule for the system

Reconfiguration scenario1: Suppose that at time t_1, an external reconfiguration scenario is applied to add a new task τ_4 (Table 2) to processor P_1. After addition, as shown in Figure 10, the schedule is infeasible since τ_4 misses its deadline equal to 20. The energy consumed by processor P_1 increases up to 20 energy units.

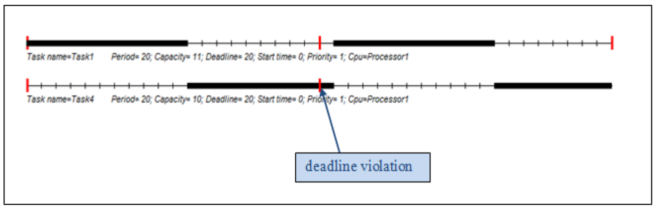

Figure 10: Schedule after addition of task τ_4

Reconfiguration scenario2: Suppose that at time t_2 after t_1 a second external reconfiguration scenario is applied to add task τ_5 (Table 2) to processor P_2. After addition to P_2 as shown in Figure 11, the CPU load is equal to 1.0625. The energy consumption, E_{P_2}, also increases on the hyper-period H (H=[$lcm(T_2, T_5)$] = 48) and becomes equal to 95 units of energy. The schedule is infeasible since task τ_5 misses its deadline at 48 and completes at t = 51.

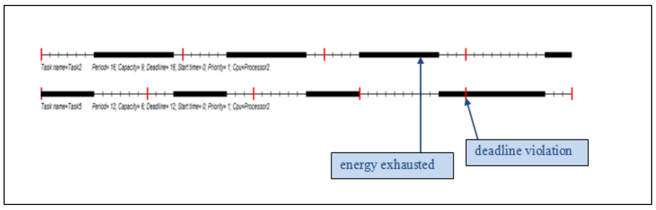

Figure 11: Schedule after addition of task τ_5

Reconfiguration scenario3: Suppose that at time t_3 after t_2 an external reconfiguration scenario is applied to add task τ_6 (Table 2) to processor P_1. After addition, as shown in Figure 12, the CPU load is equal to 1.65.

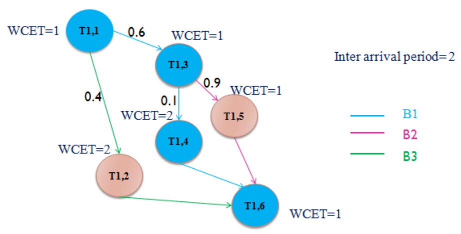

Figure 3: DAGP G_1 associated to task τ_1

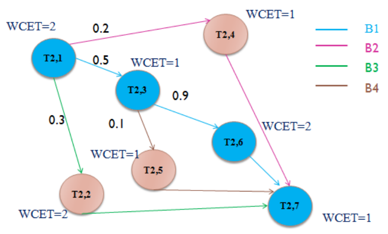

Figure 4: DAGP G_2 associated to task τ_2

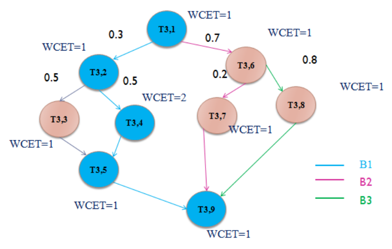

Figure 5: DAGP G_3 associated to task τ_3

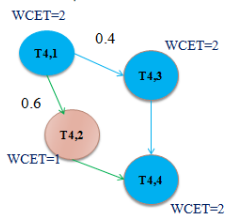

Figure 6: DAGP G_4 associated to task τ_4

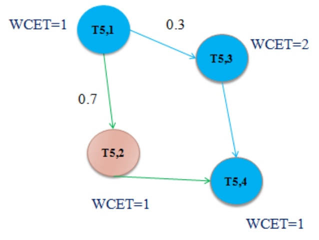

Figure 7: DAGP G_5 associated to task τ_5

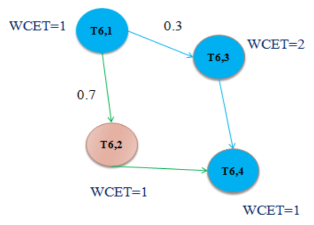

Figure 8: DAGP G_6 associated to task τ_6

The energy consumption E_{P_1} also increases and becomes equal to 32 units of energy on the hyper-period H ($H = [lcm(T_1, T_4, T_6)] = 20$). The schedule on P_1 is infeasible since task τ_4 misses its deadline at 20 and completes at t = 30. Task τ_6 misses its deadline at 20 and completes at t = 33.

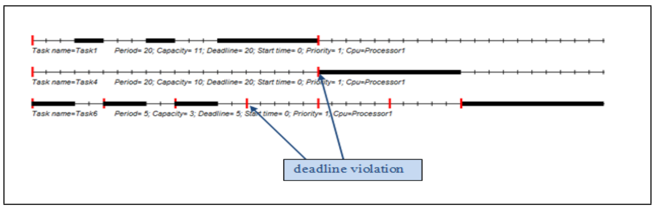

Figure 12: Schedule after addition of task τ_6

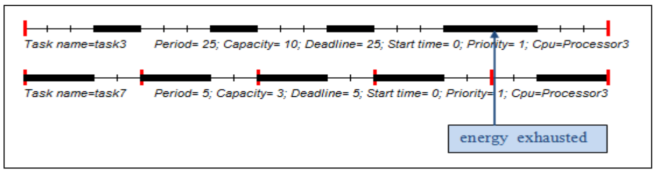

Figure 13: Scheduling of the system after the addition of the task τ_7

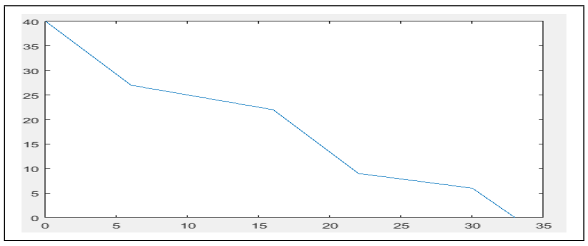

Figure 14: Energy consumption of processor P_3

Reconfiguration scenario4: Suppose that at time t_4 after t_3 an external reconfiguration scenario is applied to add task τ_7 (Table 2) to processor P_3. The CPU utilisation of processor P_3 becomes $U_{p_3} = 0.8$, hence the timing constraint is satisfied but as shown in Figure 13, the schedule is infeasible since the energy constraint is violated. As the energy consumption of E_3 is equal to 80 on the hyper-period H ($H = [lcm(T_3, T_7)] = 75$), τ_3 stops at t=20 before completing execution since there is no sufficient energy in the storage unit. The energy consumption profile of processor P_3 is presented in Figure 14.

Table 2: System Reconfiguration Scenarios

Task	C_i	T_i	D_i	En_i	d_{c_i}	(m,k)-firm
Reconfiguration scenario 1						
τ_4	10	20	20	13	D	(1,2)
Reconfiguration scenario 2						
τ_5	6	12	12	14	D	(1,2)
Reconfiguration scenario 3						
τ_6	3	5	5	3	A	(1,2)
Reconfiguration scenario 4						
τ_7	6	15	15	13	E	(1,2)

The reconfiguration scenario increases the energy consumption and/or pushes some tasks to violate the corresponding deadline. In order to re-establish the system feasibility, we propose four solutions which are detailed in section VI.

NEW SOLUTIONS FOR FEASIBLE LOW-POWER REAL-TTME RECONFIGURABLE SYSTEMS

The problem of scheduling probabilistic directed acyclic graphs (DAGP) on a reconfigurable multiprocessor platform under hard real-time and energy constraints is known to be NP-complete (Michael and David 1979). We assume that some events such as hardware failures may occur and impose a dynamic software reconfiguration for maintaining feasibility. This paper addresses the even more difficult problem of scheduling on reconfigurable systems that entirely rely on energy harvesting with limited capacity storage. We provide four approaches to address these challenges.

Motivation

A run-time external reconfiguration scenario is a dynamic operation allowing the addition/removal of the assumed DAGP tasks. Thereafter, some tasks may miss their hard deadlines and the energy constraint may be violated. Hence, checking the system feasibility after any reconfiguration scenario is of utmost importance. Further, each processor in the multiprocessor platform should satisfy the hard real-time constraint in which the CPU processor utilization is no greater than the schedulable utilisation U_x, in addition to the energy constraint where the processor's energy consumption is at most equal to the energy harvested in the storage associated to the processor. If the system feasibility is satisfied then the system operates normally. Besides, if at least one of the feasibility conditions is violated then the dynamic software reconfiguration solutions are applied one by one in order to reconfigure the system and re-establish system feasibility. The four solutions are performed in a hierarchical order:

- Modify the inter-arrival period of DAGP tasks so as to decrease the global load,
- Decompose each DAGP task of the faulty processor to a set of hipsters and let them migrate to other non-faulty processors,
- Degrade the quality of service on each faulty processor. Tasks may be executed according to (m,k)-firm constraints,
- Delete hipsters or DAGP tasks so as to minimize the global deadline miss ratio.

Formalization

In this section, we formalize the solutions.
Solution 1:Inter-Arrival Period Modification (IAPM)
It consists in modifying the inter-arrival period for two consecutive sub-tasks so as to decrease the global load.
Proposition 1: To re-obtain system feasibility on the faulty processor, the new inter-arrival period p should be equal to:

$$\frac{U_x - \sum_{i=1}^{n}\sum_{j=1}^{|V_{i,c}|}\frac{C_{i,j}}{T_i}}{\sum_{i=1}^{n}\sum_{j=1}^{|V_{i,c}|}\frac{(|V_{i,c}|-1)}{T_i}}. \tag{11}$$

Proof: According to formula (3), a given task set is assumed to be schedulable under a scheduling policy if the utilisation processor factor U is no greater than the schedulable utilisation U_x:

$$\sum_{i=1}^{n}\frac{C_i}{T_i} \leq U_x$$

According to **definition 4** the WCET C_i of task τ_i, $i = \{1..N\}$, is equal to $C_{i,j} + (|V_{i,c}| - 1) * p$ by replacing C_i by its value in formula (3) the feasibility condition can be calculated as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{|V_{i,c}|}\frac{C_{i,j} + (|V_{i,c}| - 1) * p}{T_i} \leq U_x$$

Then, the inter-arrival period p

$$p \leq \frac{U_x - \sum_{i=1}^{n}\sum_{j=1}^{|V_{i,c}|}\frac{C_{i,j}}{T_i}}{\sum_{i=1}^{n}\sum_{j=1}^{|V_{i,c}|}\frac{(|V_{i,c}|-1)}{T_i}}$$

Running-example 1: Let us consider the reconfiguration scenario1, the solution1 "Inter-arrival period modification" is applied to processor P_1. According to formula (11), the inter-arrival period p is decreased up to 1. Hence the WCET of tasks τ_4 and τ_1 are decreased to 8 and 9. Therefore, the processor utilization factor is modified to be U_{p_1} = 0,85 from U_{p_1} = 1.05. As shown in Figure 15, the schedule on P_1 is feasible since U_{p_1} = 0,85.

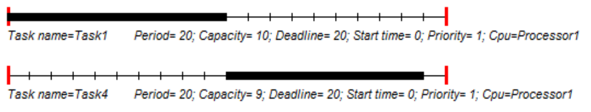

Figure 15: Schedule after Inter-arrival period modification

Solution 2: Decomposition and Migration of Branches
This solution is divided into two steps:

- First step: selection of a branch or group of branches and make them migrating to other non-faulty processors in order to re-establish the system feasibility,
- Second step: selection of processors into which the migrant branches will be assigned.

Step1: Hipster Selection Heuristic

Proposition 2: The task with the lowest degree of criticality will be decomposed into a set of hipsters. Then the current critical execution flow will be removed from the DAGP.

Step2: Processor Selection Heuristic

Proposition 3: We calculate the set of processors for which the candidate hipsters can be affected. Then, sort this set in increasing order of energy availability in storage unit.

Running-example 2: When the second reconfiguration scenario is applied, Solution 2 "Decomposition-Migration" is as follows. ***Step1:*** *Sort the task set* $\{\tau_2 = (9,16,16,13), \tau_5 = (6,12,12,14)\}$ *in a non decreasing order of critical level then the task* τ_2 *is selected to be decomposed. Let* $G_2\{\mathcal{F}_2, V_2, E_2\}$ *be the DAGP associated to task* τ_2 *as depicted in Figure 4. Let,* $\mathcal{F}_2 = (F_{2,1}, F_{2,2}, F_{2,3}, F_{2,4})$ *be the set of all execution flows of* τ_2 *such that* $F_{2,1}$ *is the critical execution flow. The new DAGP* G_2 *associated to task* τ_2 *after elimination of the critical execution flow* $F_{2,1}$ *is shown in Figure 16 and new parameters of* τ_2 *become equal to (7,16,16,12).* ***Step2:*** *By migrating the hipster* $F_{2,1}$ *to processor* P_3*, the system feasibility is re-established as presented in Figure 17. The schedule is feasible since* U_{p_3} *=0.96 and* U_{p_2} *= 0.93. The total energy consumptions of processor* P_2 *and* P_3 *are* $E_{p_2} = 92, E_{p_3} = 25$.

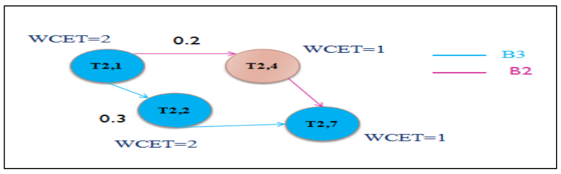

Figure 16: DAGP G_2 associated to task τ_2 after elimination of the critical execution flow $F_{2,1}$

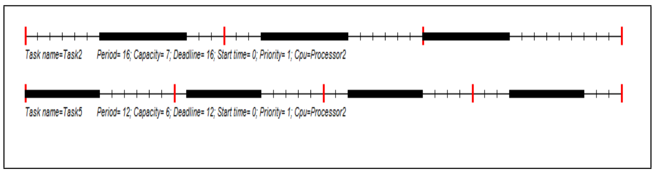

Figure 17: Schedule after decomposition and migration of branches

Solution 3: Degradation

This solution is based on the (m,k) firm constraint proposed by Hamdaoui (Hamdaoui and Ramanathan Decembre 1995) in order to provide a flexible real-time system with graceful degradation of the quality of service (QoS). We tolerate some deadline violations while maintaining a certain global quality of service in case of system overload. Therefore, we accept that in a faulty processor P_f, tasks may be executed with (m,k)-firm constraints.

Proposition 4: The task set ψ is schedulable under (m,k) firm constraints if $\frac{m}{k}$ is equal to $\lfloor \frac{U_x}{\sum_{i=1}^{n} \frac{C_i}{T_i}} \rfloor$

Proof: A given task set ψ is assumed to be schedulable with (m,k) firm constraints under a given scheduling policy if the utilisation processing factor U with (m,k) firm requirements, defined by $U_{m,k} = \sum_{i=1}^{n} \frac{C_i * m_i}{T_i * k_i}$, is no greater than the schedulable utilisation U_x defined as follows:

$$\sum_{i=1}^{n} \frac{C_i * m_i}{T_i * k_i} \leq U_x$$

Suppose, that $\frac{m_i}{k_i}$ is a constant θ then, $\theta \leq \frac{U_x}{\sum_{i=1}^{n} \frac{C_i}{T_i}}$

$$\theta = \lfloor \frac{U_x}{\sum_{i=1}^{n} \frac{C_i}{T_i}} \rfloor \tag{12}$$

Running-example 3: When the third reconfiguration scenario is applied, Solution 3 "Degradation" is as follows.
$\sum_{i=1}^{n} \frac{C_i * m_i}{T_i * k_i} \leq 1$, $\theta = \lfloor \frac{1}{\sum_{i=1}^{n} \frac{C_i}{T_i}} \rfloor$, $\theta = \frac{1}{2}$, $m = \frac{1}{2} * k$
Therefore, we accept that in the faulty processor P_1 *the tasks be executed under (1,2)-firm constraints which indicate that the deadlines of at least 1 instance among 2 consecutive ones must be met for tasks* τ_1, τ_4 *and* τ_6.

Solution 4: Removal Hipster (RHH)

Delete hipsters or DAGP tasks so as to minimize the global deadline miss ratio. For any faulty processor P_f, we associate to each task in ψ_{P_f} a density denoted by:

$$\gamma = \frac{En_i}{T_i} \tag{13}$$

Proposition 5: We sort all tasks in increasing order of densities so that we can reject one by one those with highest densities until the remaining utilization factor of the faulty processor is lower than U_x and the total energy consumption is below $E_{Bf(t)}$.

Running-example 4: When the reconfiguration scenario 4 is applied, Solution 4 "Removal Hipster" is performed to processor P_3*. According to formula (13)* $\gamma_3 = \frac{1}{5}$ *and* $\gamma_7 = \frac{13}{15}$*. Then, task* τ_7 *has the highest density. By removing the critical execution flow* $F_{7,1}$*, parameters of* τ_7 *become (5,15,15,4). Therefore, the processor utilization factor is modified to be* U_{p_3} *= 0,73 instead of* U_{p_3} *= 0.8 and the total energy consumption* $E_{p_3} = 35 \langle E_{B3}(t) = 40$*. In this case study, the solution " Removal Hipster" reduces the total energy consumption of processor* P_3 *with up to 56.25%.*

EXPERIMENTAL STUDY

According to the solutions formulated above, Algorithm1 evaluates a set of unpredictable reconfiguration scenarios applied repeatedly during the running execution of the system and provides four solutions in order to re-establish the system feasibility.

Algorithm 1 Main algotithm

```
while 1 do
  Reconfig();
  if $feasible(P_j) = false$ then
    Inter-arrival period Modification($P_f$, $\psi_{P_f}$)
    if $feasible(P_j) = false$ then
      Decomposition Migration($P_f, \psi_{P_f}, B_f(t), \pi, \psi$)
      if $feasible(P_j) = false$ then
        Degradation($P_f, \psi_{P_f}$)
        if $feasible(P_j) = false$ then
          Removal Hipster($\psi_{P_f}, B_f(t)$)
        end if
      end if
    end if
  end if
end while
```

Decomposition-Migration DAGP Solution

Hipster Selection Heuristic
Algorithm2 depicts the pseudo-code of hipster selection heuristic (HSH). For a faulty processor P_f, Branch Selection Heuristic sorts tasks in increasing order based on criticality level dc, then, the task τ_{acc} with the lowest dc is selected to be decomposed. We calculate the WCET and WCEC of each execution flow in τ_{acc}. In order to get a feasible execution in the faulty processor P_f, the workload should be no greater than U_x and the total energy consumption should be no greater than the amount of energy available in the storage B_f. Therefore, the scheme deletes the current critical execution flow F^c_{acc} and recalculates the new one until the feasibility conditions are satisfied. The overall run-time computational complexity of Hipster Selection Heuristic is $O(N^2)$.
Processor Selection and Assignment Heuristic
Algorithm3 depicts the pseudo-code of processor selection and assignment heuristic (HSAH). The overall runtime computational complexity of Processor Selection and Assignment Heuristic is O(m).

Removal Hipster Heuristic

The pseudo-code of the Removal Hipster Heuristic (RHH) is given in algorithm 4. For a faulty processor P_f we associate to each assigned task a density defined as the ratio of WCEC over the period T_i. We sort all tasks in an increasing order of densities so as to reject critical execution flows with highest densities, one by

Algorithm 2 Hipster selection heuristic

```
Input:$\psi_{pf} = \{\tau_1, ..., \tau_n\}$ tasks' set assigned to the faulty processor $P_f$ ; $\mathcal{F}_{acc} = \{F_{acc,1}, ..., F_{acc,N}\}$ denotes the set $F_{acc}$ of all possible execution flow of $G_{acc}$; The energy available in the storage of processor $P_f$ at a time t $B_f(t)$.
Output:Candidate branches.
Sort tasks' set $\psi_{pf}$ in increasing order of criticality level
$\tau_{acc} \leftarrow min(\psi_p)$
for $k = 1 to N$ do
  Calculate $WCET(F_{acc,k})$, Calculate $E_{F_{acc,k}}$
end for
repeat
  $F \leftarrow Max(\mathcal{F}_{acc})$
  Delete(F,$G_{acc}$)
  $C_{\tau_{acc}} \leftarrow Max(C_{F_{acc,i}})$, $E_{\tau_{acc}} \leftarrow Max(P_{F_{acc,i}})$
  Calculate $U_{P_f}$
  Calculate $E_{P_f}$
until $U_{p_f} < U_x$ and $E_{p_f} < B_f(t)$
```

Algorithm 3 Processor Selection and Assignement Heuristic

```
Input:Processors' set $\pi = \{P_1, ..., P_m\}$;
Output:Branch assigned to non faulty processor.
$P \leftarrow \pi/\{P_f\}$
for i=1 to m-1 do
  if $U_i \leftarrow U_i + \frac{C_{acc}}{T_{acc}} < U_x$ then
    insert($P_i, \pi_{acc}$)
  end if
end for
Affect $F_{acc}$ to the candidate processor which has the highest energy level.
```

one, until the remaining utilization of the faulty processor is lower than U_x, and the total energy consumption is below $E_{Bf(t)}$. The overall run-time computational

Algorithm 4 Removal Hipster

```
Input:$\psi_{P_f} = \{\tau_1, ..., \tau_n\}$ task set assigned to faulty processor $P_f$; The energy available in the storage of processor $P_f$ at time t $B_f(t)$.
Output: feasible tasks set
for i=1 to n do
    $\gamma_i = \frac{E_i}{T_i}$
end for
sort task set $\psi_{pf}$ in decreasing order of task densities
$T_{acc} \leftarrow T$
for i=1 to n do
    if $U_{p_f} > U_x$ and $E_{p_f} > E_{B_f}(t)$ then
        reject critical execution flow $F_n^c$ of nth task
        done with task rejection, break
    end if
end for
```

complexity of Removal Hipsters Heuristic is O(n).

Simulation And Analysis

We explore in this section the performance of the four solutions, RHH, DMH, DH, IAPM. In order to evaluate them, we consider a set of 50 tasks to be schedulable on 8 identical processors. Parameters of tasks are randomly generated as follows:

- The period T_i (i = 1, .., 50) is randomly chosen in the range [500, 900]
- The WCET C_i (i = 1, .., 50) is randomly chosen in the range [6, 20]
- The emergency execution level dc_i (i= 1, .., 50) is randomly chosen in the range [A, D]

A DAGP generator is used to generate graphs of tasks $G_i = (V_i, E_i)$ (i= 1, .., 50) according to C_i, T_i. The generated graphs are similar to those taken in the case study. We assume a set of 6 unpredictable reconfiguration scenarios applied repeatedly. The periodicity of reconfiguration is equal to 100 units of time. Each scenario adds a set of n tasks such that n is randomly chosen in the range [10,50]. Moreover, we assume that a reconfiguration scenario may affect randomly one of the 8 processors. Figure 18 presents the percentage of satisfied deadlines for the solutions IAPM, DH and RHH. It clearly shows that the performance of all algorithms degrades when the processor utilization factor increases due to a set of reconfiguration scenarios applied repeatedly at run-time. IAPM has the strongest degradation whereas RHH and DH provide similar results when U is equal to 1.76. Indeed, when U exceeds 1.34, IAPM loses performance. In general, RHH outperforms all other

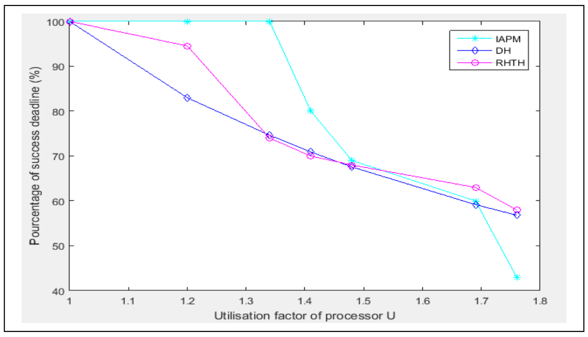

Figure 18: Percentage of satisfied deadlines, when a set of reconfiguration scenario is applied repeatedly during the execution time

solutions. Figure 19 presents the percentage of energy gain. It can be derived from this figure that the solution RHH achieves energy savings of up to 67% when compared to the energy consumption of the initial task schedule. However, the solution DMH provides energy savings up to 5%.

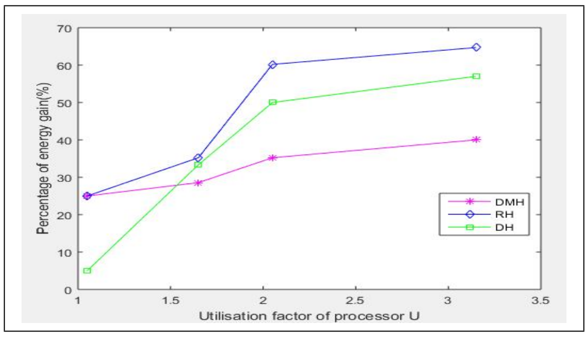

Figure 19: Percentage of energy gain when a set of reconfiguration scenario is applied repeatedly during the execution time

CONCLUSION AND DISCUSSION

We firmly believe that nowadays it has become crucial to investigate new models and techniques to schedule contemporary applications subject to energy and real-time requirements, especially when unpredictable reconfiguration scenarios occur at run time. To the best of our knowledge, the work reported in this paper is the first one that focuses on scheduling of real-time periodic tasks in a reconfigurable multiprocessor energy harvesting real-time system. In this paper, we have presented a partitioning heuristic in the energy harvesting context, namely BF-EH, in order to build the initial system configuration. However, when an external reconfiguration scenario is applied, the system may evolve towards an infeasible state. We have proposed a new task model where a Probabilistic Directed Acyclic Graph DAGP is attached to each task. Indeed, the probability of execution flows may reduce the percentage of missed deadlines and increases the energy availability in the storage unit. We have proposed four approaches to re-establish the system feasibility. Extensive simulation experiments show that the proposed four solutions achieve energy savings up to 67%, and reduce the deadline miss ratio up to 10%.

REFERENCES

Abdallah N.; Queudet A.; and Chetto M., 2014. *Task Partitioning Strategies for Multi-core Real-Time Energy Harvesting Systems.* In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on.* IEEE, 125–132.

Bonifaci V.; Marchetti-Spaccamela A.; Stiller S.; and Wiese A., 2013. *Feasibility analysis in the sporadic DAG task model.* In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on.* IEEE, 225–233.

Burns A., 1991. *Scheduling hard real-time systems: a review. Software Engineering Journal*, 6, no. 3, 116–128.

Camponogara E.; de Oliveira A.B.; and Lima G., 2010. *Optimization-based dynamic reconfiguration of real-time schedulers with support for stochastic processor consumption. Industrial Informatics, IEEE Transactions on, 6(4), 594-609.*

Chetto M., 2014. *Optimal scheduling for real-time jobs in energy harvesting computing systems. Emerging Topics in Computing, IEEE Transactions on*, 2, no. 2, 122–133.

Chniter H.; Khalgui M.; and Jarray F., 2014. *Adaptive embedded systems: New composed technical solutions for feasible low-power and real-time flexible OS tasks.* In *Informatics in Control, Automation and Robotics (ICINCO), 2014 11th International Conference on.* IEEE, vol. 1, 92–101.

Fonseca J.; Nlis V.; Raravi G.; and Pinho L.M., 2015. *A Multi-DAG Model for Real-Time Parallel Applications with Conditional Execution. In Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on (pp 225-233) IEEE.*

George S.E.; Bocko M.; and Nickerson G., 2005. *Evaluation of a vibration-powered, wireless temperature sensor for health monitoring.* In *Aerospace Conference, Big Sky, MT, USA.* IEEE, 3775–3781.

Gharsellaoui H.; Khalgui M.; Ahmed S.B.; et al., 2012. *New Optimal Solutions for Real-Time Reconfigurable Periodic Asynchronous Operating System Tasks with Minimizations of Response Time. International Journal of System Dynamics Applications (IJSDA)*, 1, no. 4, 88–131.

Hamdaoui M. and Ramanathan P., Decembre 1995. *A dynamic priority assignment technique for streams with (m, k)-firm deadlines. IEEE Transactions on Computers, vol 44, no 12, pp 1443-1451.*

Khalgui M., 2010. *NCES-based modeling and CTL-based verification of reconfigurable embedded control systems. Computers in Industry Journal, vol 61, no 3, pp 198-212.*

Liu C.L. and Layland J.W., 1973. *Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the Association for Computing Machinery*, 20, no. 1, 46–61.

Manacher G.K., 1967. *Production and stabilization of real-time task schedules. Journal of the Association for Computing Machinery*, 14, no. 3, 439–465.

Michael R.G. and David S.J., 1979. *Computers and intractability: a guide to the theory of NP-completeness. WH Freeman and Co, San Francisco.*

Moser C.; Brunelli D.; Thiele L.; and Benini L., 2006. *Lazy scheduling for energy harvesting sensor nodes.* In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, Springer. 125–134.

Saifullah A.; Ferry D.; Li J.; Agrawal K.; Lu C.; and Gill C.D., 2014. *Parallel real-time scheduling of DAGs. Parallel and Distributed Systems, IEEE Transactions on*, 25, no. 12, 3242–3252.

Shin Y. and Choi K., 1999. *Power conscious fixed priority scheduling for hard real-time systems.* In *Design Automation Conference, 1999. Proceedings. 36th.* IEEE, New Orleans, LA, USA, 134–139.

Singhoff F.; Legrand J.; Nana L.; and Marcé L., 2004. *Cheddar: a flexible real time scheduling framework.* In *ACM SIGAda Ada Letters.* ACM, New York, vol. 24, 1–8.

Stankovic J.A., 1988. *Real-time computing systems: The next generation, Tech. Report COINS TR 88-06.* Department of Computer and Information Science, University of Massachusetts Amherst.

Wang X.; Khemaissia I.; Khalgui M.; Li Z.; Mosbahi O.; and Zhou M., 2015. *Dynamic low-power reconfiguration of real-time systems with periodic and probabilistic tasks. Automation Science and Engineering, IEEE Transactions on*, 12, no. 1, 258–271.