Computers and Electrical Engineering 37 (2011) 498–510

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

A real-time scheduling framework for embedded systems with environmental energy harvesting ☆

Hussein EL Ghor [a], Maryline Chetto [a,*], Rafic Hage Chehade [b]

[a] IRCCyN Laboratory, University of Nantes, 1 Rue de la Noe, F-44321 Nantes, France
[b] Lebanese University, IUT Saida, Lebanon

ARTICLE INFO

Article history:
Received 6 May 2010
Received in revised form 2 May 2011
Accepted 2 May 2011
Available online 31 May 2011

ABSTRACT

Real-time scheduling refers to the problem in which there is a deadline associated with the execution of a task. In this paper, we address the scheduling problem for a uniprocessor platform that is powered by a renewable energy storage unit and uses a recharging system such as photovoltaic cells. First, we describe our model where two constraints need to be studied: energy and deadlines. Since executing tasks require a certain amount of energy, classical task scheduling like earliest deadline is no longer convenient. We present an on-line scheduling scheme, called earliest deadline with energy guarantee (EDeg), that jointly accounts for characteristics of the energy source, capacity of the energy storage as well as energy consumption of the tasks, and time. In order to demonstrate the benefits of our algorithm, we evaluate it by means of simulation. We show that EDeg outperforms energy non-clairvoyant algorithms in terms of both deadline miss rate and size of the energy storage unit.

1. Introduction

The power management is becoming the central issue in embedded systems that must carry their own power source and cannot rely on a power outlet on the wall. Without power, the system is useless. In the consumer space such as mobile communication, the consequence may be minor; however, in the so-called hard real-time embedded systems, energy lack means a failure that can cost millions and even human lives. Traditionally, many embedded systems have been designed to be low-power. But there is a fundamental difference between power-aware and low-power technologies. In a power-aware system, we have to make the best use of the available power and the goal of a scheduler is to assign real-time tasks to time slots such that all timing and power constraints are satisfied [1].

Nowdays, higher energy-density batteries and supercapacitors are being developed but the amount of energy available still severely limits the system's lifespan. On the other hand, in most wireless applications including sensor networks, recharging or replacing batteries is not practical or permitted and consequently alternative power sources which are present in the environment should be employed. Environmental energy harvesting is deemed a promising approach because many sensing environments provide sufficient energy that can be harvested for providing power on an infinite time. Several technologies to extract energy from the environment have been demonstrated including solar, motion-based, biochemical, and vibrational energies and many others are being developed. A key consideration that affects power management in an energy harvesting system is that instead of minimizing the energy consumption and maximizing the lifetime achieved as in classical energy storage operated devices, the system operates in a so-called energy neutral mode by consuming only as much energy as harvested [2].

☆ Reviews processed and approved for publication by Editor-in-Chief Dr. Manu Malek.

* Corresponding author.
E-mail addresses: elghorh@irccyn.ec-nantes.fr (H. EL Ghor), maryline.chetto@univ-nantes.fr (M. Chetto), rafichajj@hotmail.com (R.H. Chehade).

0045-7906/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compeleceng.2011.05.003

Embedded systems are generally real-time ones which have to compute a timely response to external stimuli. Real-time systems can be classified in three categories: hard, soft and weakly-hard. In hard real-time systems, all the application programs called tasks, must be guaranteed to complete within their deadlines. For soft real-time systems, it is acceptable to miss some of the deadlines occasionally with additional value for the system to finish the task, even if it is late. In weakly-hard real-time systems, tasks are allowed to miss some of their deadlines, but there is no associated value if they finish after the deadline. Whatever its category, a real-time embedded system is said to achieve energy neutral operation if its execution requirements can be supported forever despite energy limitations. This supposes to execute the tasks such that the energy used by them is always less than the energy harvested. A simple approach would be to build a harvester whose minimum instantaneous power at any instant is sufficient to supply the maximum power required by any executing task. Unfortunately, this has disadvantages, such as high costs and large space. Most embedded systems constructed to date do not extract power efficiently from the source. As a result, they use a much larger harvester (e.g. solar panel) than necessary to yield the same level of power as a more efficient one, or they rely on a larger, more expensive, higher capacity battery than needed in order to sustain extended operation. In both cases, the low harvesting efficiency limits the achievable performance and will preclude the system from many important applications. This has motivated researchers to design energy harvesting capabilities specifically dedicated to real-time embedded systems from about four years [3]. The crucial issue associated to these systems is to find scheduling mechanisms that can adapt their performance to the available energy profile. Up to now, when designing a real-time embedded system, the first concern has been usually time, leaving energy efficiency as a hopeful consequence of empiric decisions. Now, the primary concern is that power from solar panels or other free sources that cannot be stored (or stored with limited capacity) should be fully consumed greedily, or else this energy will be wasted.

In this paper, we address the problem of scheduling real-time tasks on one processor to meet deadlines and energy constraints. The system we target here consists of a processing unit, an energy harvester and a rechargeable energy storage such as battery or supercapacitor. So, the problem we deal with can be formulated as follows: *How can we schedule the tasks so as to guarantee their timing constraints perpetually by suitably exploiting both the processor and the available ambient energy?*

This work presents experimental results about a scheduling framework called earliest deadline with energy guarantee (EDeg) resulting from the extension of the earliest deadline as late as possible (EDL) server [4]. We modify earliest deadline first (EDF) scheduler so as to account for the properties of the energy source, capacity of the energy storage as well as energy consumption of the tasks. We propose a slack-based method for delaying tasks and making the processor inactive during recharging phases of the energy storage unit. On-line computing by how long the tasks should be delayed is possible thanks to EDL properties.

The remainder of the paper is organized in the following manner. In the next section, we summarize related work. Section 3 describes the model and gives necessary terminology. In Section 4, we present background materials that are essential for the understanding of the paper. Section 5 is the presentation of our scheduling scheme, EDeg, with some indications about practical issues. Results of a simulation study are presented in Section 6. Section 7 concludes the paper and gives some new directions of work.

2. Related work

Energy-aware real-time scheduling has been the subject of intensive research. Most of the works focus on either minimizing the energy consumption or maximizing the system performance such as the lifetime achieved under the energy constraints [5]. In such works, the rechargeability of the energy storage unit is always disregarded. Little work has explored the problem of scheduling real-time tasks in a uniprocessor rechargeable system. The question amounts to find a schedule which is able to execute all the tasks within the deadline and energy storage constraints i.e. without running out of energy. Classical priority driven scheduling has been extended to variable-voltage processors. The idea is to save power by slowing down the processor just enough to meet the deadlines [6]. But solely applying these techniques has limitations in energy harvesting systems because they minimize CPU power, rather than they dynamically manage power according to the profiles of both available energy and processor workload. In what follows, we review the main studies for scheduling tasks in real-time energy harvesting systems.

2.1. Scheduling frame based systems

In [7], Allavena et al. address the problem of finding the scheduling of frame-based embedded systems which is able to execute all the tasks within the deadlines, starting with an energy storage fully charged, ending at the same energy level as when started. In this model, all task periods are identical and all task deadlines are equal to the common period. Consequently, the order of task execution within a frame is not crucial for whether the task set is schedulable or not. Moreover, the power scavenged by the energy source is assumed to be constant and all tasks consume energy at a constant rate. A solution is presented that schedules tasks in such a way that the wasted recharging energy is minimized and the energy storage level is at all times within two limits. The idea behind this algorithm is to insert as little idle time as necessary for recharging the energy storage and minimizing the length of the schedule. This work is certainly the first one to concentrate on a rechargeable system with hard real-time constraints. However, the solution only deals with frame based systems under

the restrictive hypothesis that each task is characterized by an instantaneous consumption power which is constant along time.

2.2. Scheduling with task rewards

Weakly-hard real-time systems can be designed so as providing services with adjustable quality evaluated in terms of rewards. Reward-based scheduling was explored in the context of imprecise computation where real-time tasks consist of mandatory and optional parts and a reward function is associated with the length of the optional part [8]. Multiple versions allow quality of service tradeoffs by providing different levels of accuracy with different execution times and energy consumptions. When the harvested energy is temporarily low, the service must be lowered or suspended. The main issues are to maximize the overall reward and to determine the minimum battery capacity necessary to optimally exploit a given power source. The research presented in [9] propose off-line solutions in a single speed processor system while a solution with dynamic voltage scaling (DVS) has been proposed recently in [10].

2.3. Scheduling with solar energy harvesting

The first work that really makes adaptive power management for energy harvesting systems has been published in [11]. Their model captures the behaviour of an actual solar energy source through tracing its power profile. In that paper, the authors propose algorithms for tuning system duty cycle based on the parameters of the solar energy source. The system switches between active mode and sleep mode depending on harvesting energy and consequently may operate perpetually. Here, the problem is formulated as a linear program. This linear program has to be solved periodically. Within each period, adaptations of the duty cycle become necessary if the observed energy values differ from the predicted ones. The main drawback of the proposed algorithms is that they do not target at tasks in a real-time pattern. Dealing with real-time tasks under the strong variation of energy source with respect to time remains a central issue up to day [12].

2.4. Scheduling tasks with constant power consumption

Later in [3], Moser et al. focus on scheduling tasks with deadlines, periodic or not, that run on a uniprocessor system that is powered by a rechargeable storage unit. The source power is assumed to be predictable but time-varying. They propose lazy scheduling algorithm (LSA) and prove it to be optimal in terms of deadline miss ratio. LSA is a variation of the famous earliest deadline first scheduler [13]: the system starts executing a task only if the task is ready and has the earliest deadline among all ready tasks and the system is able to keep on running at the maximum power until the deadline of the task. In that work, the consumption power of the computing system is characterized by some maximum value which implies that for every task, its total energy consumption is directly connected to its execution time through the constant power of the processing device. The main disadvantage of this work is that the LSA algorithm executes tasks at full power and therefore, future tasks will violate deadlines because of limited energy. Moreover, in practice, the total energy which can be consumed by a task is not necessarily proportional to its execution time.

2.5. Scheduling with DVFS technology

In [7], Allavena et al. describe an off-line scheduler that uses voltage and frequency selection (DVFS) for a frame based system. While they permit to reduce power consumption by slowing down task execution under deadline constraints, their approach relies on the unrealistic assumption that both the instantaneous consumption power and production power are constant. In 2008, Liu et al. propose an energy aware dynamic voltage and frequency selection algorithm, called EA-DVFS, for periodic tasks [14]. The purpose of EA-DVFS is to efficiently use the task slack and further reduce the deadline miss rate. In this algorithm, whether or not the system slows down the task execution for energy saving depends on the available energy. If the system has sufficient energy, the task is executed at its full speed; otherwise, it is stretched and executed at a lower speed. Unfortunately, this algorithm has limited impact since, in most embedded applications, the energy storage has a non constant recharging rate and every task is characterized by its own profile of power consumption which can vary along time.

2.6. Scheduling tasks with variable power consumption

In practice, the total energy which can be consumed by a task has no correlation with the worst case execution time [15]. For every task, the worst case instantaneous consumption power depends on the circuitry as well as the instrumentation used by the task during and/or after its execution. Let us notice that the biggest energy consumption is not necessarily in the computer. It may come from actuators receiving output data from the tasks such as motors. Clearly, as an embedded system uses a unique energy storage considered as the critical resource of the system, a successful power-aware scheme must consider these non-computation activities and coordinate their power usage as a whole system.

3. Model and terminology

3.1. Task set

A large number of energy-constrained embedded real-time systems operate in a cyclic basis, with a set of tasks that must execute before deadlines. We consider here an embedded system that is composed of periodic tasks as depicted in Fig. 1. The arrival times, energy demands and deadlines of the tasks are known in advance. A periodic task set can be denoted as follows: $\tau = \{\tau_i, i = 1, \ldots, n\}$. A four-tuple (C_i, E_i, D_i, T_i) is associated with each τ_i. In this characterization, task τ_i makes its initial request at time 0 and its subsequent requests at times $kT_i, k = 1, 2, \ldots$ called release times. The least common multiple of $T_1, T_2, \ldots, T_n$ (called the hyperperiod) is denoted by T_{LCM}. Each request of τ_i requires a worst case execution time (WCET) of C_i time units and has a worst case energy consumption (WCEC) of E_i. We assume that the WCEC of a task has no relation with its WCET. A deadline for τ_i occurs D_i units after each request by which task τ_i must have completed its execution. We assume that $0 < C_i \leqslant D_i \leqslant T_i$ for each $1 \leqslant i \leqslant n$. We define the processor utilization as $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$ and the energy utilization as $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$.

U_p is the percentage of processing time if tasks of τ are solely executed on the device. One can interpret U_e, measured in joules/s, as the average power consumed by τ when executing on the device.

A job is any request that a task makes. A four-tuple (r_j, C_j, E_j, d_j) is associated with a job J_j and gives its release time, WCET, WCEC and (absolute) deadline, respectively. A job can be preempted and later resumed at any time and there is no time or energy loss associated with such preemption.

3.2. Energy source

In order to characterize the energy source, we have to concentrate on the harvested energy since it can incorporate all losses caused by power conversion and charging process. Generally, the harvested power is time-varying including solar energy which can be assumed constant on average in a long-term perspective. However, on a short-term perspective, the harvested power is highly unstable. Here, we assume that the energy source is uncontrolled but predictable: we cannot control it but its behaviour may be modeled to predict the expected availability at a given time within some error margin. Solar energy cannot be controlled but models for its dependence on diurnal and seasonal cycles are known and can be used to predict availability. The prediction error may be improved using commonly available weather forecasts for the place where the embedded system is deployed. Consequently, we define the worst case charging rate (WCCR), namely $P_r(t)$, which is a lower bound on the harvested source power output. $P_r(t)$ is then the instantaneous charging rate that incorporates all losses caused by power conversion and charging process. We assume that energy production times can overlap with the consumption times. Clearly, we make no assumption about the nature and dynamics of the energy source, making our approach more easily implemented in real systems where data about the energy source may not be available beforehand. When available, the average power of the fluctuating energy source will be denoted as $\overline{P_r}$.

3.3. Energy storage

Our system uses an ideal energy storage unit (supercapacitor or battery) that has a nominal capacity, namely E, corresponding to a maximum energy (expressed in Joules or Watts-hour). The energy level has to remain between two boundaries E_{min} and E_{max} with $E = E_{max} - E_{min}$. The stored energy may be used at any time later and does not leak any energy over time. If the storage is fully charged, and we continue to charge it, energy is wasted. In contrast, if the storage is fully discharged, no task can be executed.

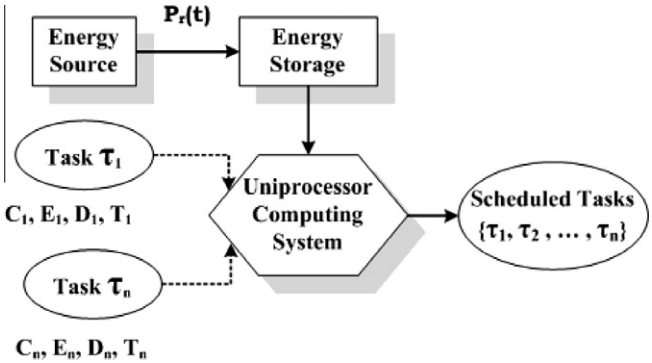


Fig. 1. A typical energy harvesting real-time system.

3.4. Definitions

- A schedule Γ for τ is said to be *valid* if the deadlines of all tasks of τ are met in Γ, starting with a storage fully charged.
- A task set τ is said to be *temporally-feasible* if there exists a valid schedule for τ without considering its energy constraints.
- A task set τ is said to be *feasible* if there exists a valid schedule for τ with considering its energy constraints.
- A scheduling algorithm will be called *optimal* if it finds a valid schedule whenever one exists.

From previous definitions, it clearly appears that every task set τ can be feasible only if $U_p \leqslant 1$ and $\frac{U_e}{P_r} \leqslant 1$.

4. Background material

Real-time task scheduling refers to determine the order in which tasks are to be executed. The problem of scheduling periodic tasks on one processor with no energy constraint has been an active area of research for more than thirty years [16].

There are two popular approaches: fixed-priority algorithms, including rate monotonic [13] and deadline monotonic [17], and dynamic-priority algorithms, including the earliest deadline first (EDF) algorithm [18]. EDF schedules at each instant of time t, the ready task (i.e. the task that may be processed and is not yet completed) whose deadline is closest to t. EDF is an optimal scheduling algorithm in the sense that if a set of tasks can be scheduled by any algorithm, then it can be scheduled by EDF. The EDF algorithm is typically preemptive, in the sense that, a newly arrived task may preempt the running task if its absolute deadline is shorter. This dynamic priority assignment allows EDF to exploit the full processor, reaching up to 100% of the available processing time. EDF can be used to schedule both periodic and aperiodic tasks, this is because the task order is based on the absolute deadline. Liu and Layland [13] proved that a periodic task set with deadlines equal to periods is schedulable by EDF if and only if the total processor utilization U_p is less than or equal to one.

4.1. Static EDS

In general, implementation of EDF consists in ordering tasks according to their urgency and executing them as soon as possible with no inserted idle time. Such implementation is known as earliest deadline as soon as possible (EDS) [4]. For a given periodic task set, the EDS schedule can be pre-computed and memorized in order to reduce scheduling overheads at run time.

Consider a periodic task set τ that is composed of three tasks, $\tau = \{\tau_i, |1 \leqslant i \leqslant 3$ and $\tau_i = (C_i, D_i, T_i)\}$. Let $\tau_1 = (3, 6, 9), \tau_2 = (3, 8, 12)$ and $\tau_3 = (3, 12, 18)$.

The EDS schedule produced on task set τ during the first hyperperiod is depicted in Fig. 2:

4.2. Static EDL

We may imagine an implementation of EDF that leads to execute periodic tasks as late as possible without causing their deadline to be missed. Then, determination of the latest start time for every instance requires preliminary construction of the schedule produced by the so-called earliest deadline as late as possible (EDL) algorithm [4].

The main idea of EDL is to differ the execution of tasks which results in maximizing the length of idle time periods at the beginning of the schedule. Determination of the duration and position of these idle times is done by mapping out the EDL schedule produced from time zero up to the end of the first hyperperiod thanks to recurrent formulae. This is realized by means of the two following vectors [19]:

- *Static deadline vector* $\mathcal{K}$*:* it represents the time instants from 0 to the end of the first hyperperiod, at which idle times occur and is constructed from the distinct deadlines of tasks. $\mathcal{K} = \{k_0, k_1, \ldots, k_i, k_{i+1}, \ldots, k_q\}$. We note that $q \leqslant N + 1$ where N denotes the number of instances within $[0, T_{LCM}]$.

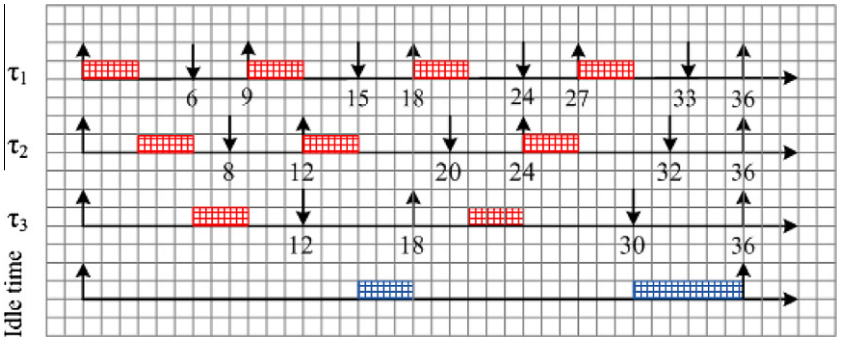


Fig. 2. Static EDS schedule.

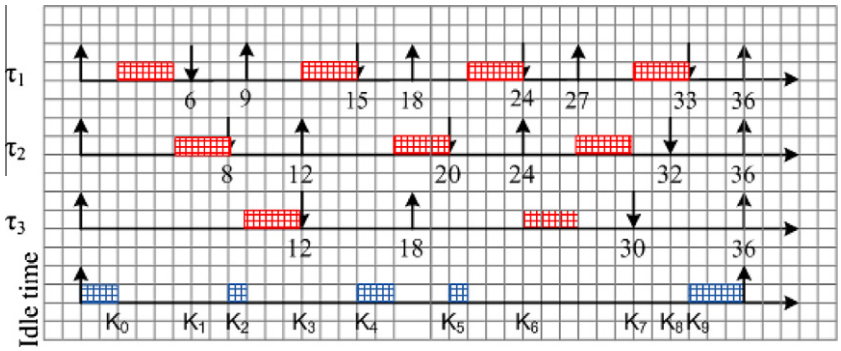


Fig. 3. Static EDL scheduling.

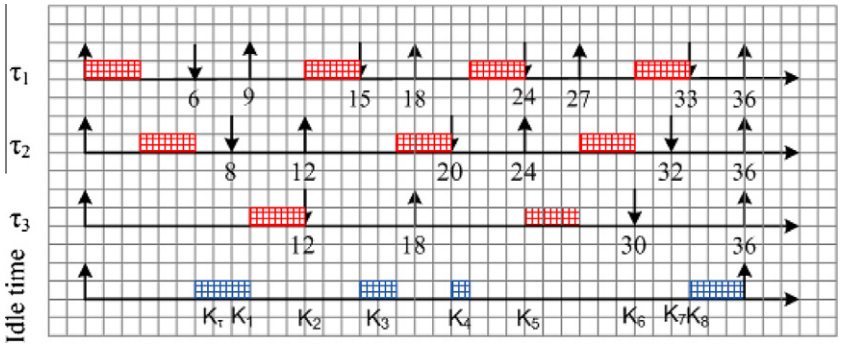


Fig. 4. Dynamic EDL schedule built at time 6.

- *Static idle time vector* $\mathcal{D}$*:* it represents the lengths of the idle times which start at time instants given by $\mathcal{K}$. $\mathcal{D} = \{\Delta_0, \Delta_1, \ldots, \Delta_i, \Delta_{i+1}, \ldots, \Delta_q\}$.

Let us consider the previous task set τ. We note that $K = (0, 6, 8, 12, 15, 20, 24, 30, 32, 33)$ and $D = (2, 0, 1, 0, 2, 1, 0, 0, 0, 3)$ (see Fig. 3).

4.3. Mixed scheduling EDS/EDL

The *slack time* of a hard deadline task set at current time t is the length of the longest interval starting at t during which the processor may be idle continuously while still satisfying all the timing constraints. Slack time analysis has been extensively investigated in real-time server systems in which aperiodic (or sporadic) tasks are jointly scheduled with periodic tasks [19]. In these systems, the purpose of slack time analysis is to improve the response time of aperiodic tasks or to increase their acceptance ratio. Determining slack time at run-time amounts to compute the so-called dynamic EDL schedule precisely defined by the two following vectors:

- *Dynamic deadline vector K(t):* it represents the time instants greater than or equal to t in the current hyperperiod, at which idle times occur.
- *Dynamic idle time vector D(t):* it represents the length of the idle times that starts at time instants given by $K(t)$.

Details of computation are given in [19].

Let us assume that the previous periodic task set has been scheduled according to EDS from time zero until time $t = 6$. To determine the slack time at time 6, we compute the dynamic EDL schedule for the interval [6,36]. Fig. 4 enables us to verify that $K(t) = (6, 8, 12, 15, 20, 24, 30, 32, 33)$, $D(t) = (2, 1, 0, 2, 1, 0, 0, 0, 3)$ and the slack time equals 2. In what follows, we will use the idea of slack stealing to postpone task execution whenever the energy level proves to be insufficient. Then recharging the energy storage unit must be achieved while still guaranteeing all the deadlines constraints.

5. A scheduling algorithm under renewable energy constraints

Under energy constraints, the scheduling algorithm has to make the energy storage level be sufficient to provide energy for all future occurring tasks, considering their timing and energy requirements and the replenishment rate of the storage

unit. For this reason, scheduling tasks according to EDS or EDL may result in unnecessary deadline violations since they are not energy-clairvoyant.

5.1. Presentation of the algorithm

The intuition behind the scheduling algorithm is to run tasks according to the earliest deadline first rule. However, before authorizing a task to execute, we must ensure that the energy storage is sufficient to provide energy for all future occurring tasks. When this condition is not verified, the processor has to sleep so that the storage unit recharges as much as possible and as long as all the deadlines can still be met despite execution postponement.

Following the idea described above, we propose the earliest deadline with energy guarantee (EDeg) algorithm. To formally present the algorithm, we need to introduce two concepts:

- The slack energy of job J_j at time t represents the amount of energy surplus in the storage that can be used from t until the start time of J_j while still guaranteeing its timing and energy requirements. The slack energy of job J_j at time t is given by $E(t) + \int_t^{d_j} P_r(t)dt - A_j$ where A_j is the processor demand within $[t, d_j)$ required by the periodic instances ready to be processed between t and d_j.
- The *slack energy of the periodic task set* at current time t, is the maximum amount of energy that can be consumed from t continuously while still satisfying all the timing constraints. The slack energy at time t is computed only when there is at least one job, say J_j which will be released after t and has a deadline d_j that is less than or equal to that of the highest priority job, ready at t. The slack energy of the system is determined by the minimum slack energy of all the jobs.

The three major components of EDeg algorithm are $E(t)$, *Slack.energy* (t) and it Slack.time(t). In details, t is the current time, $E(t)$ is the residual capacity of the storage unit at time t i.e. the energy that is currently stored. *Slack.energy*(t) and *Slack.time*(t) are, respectively the slack energy and the slack time at time t. PENDING is a Boolean which equals true whenever there is at least one job in the ready list queue. We use the function wait() to put the processor to sleep and function execute() to put the processor to run the ready job with the earliest deadline.

Characteristics of EDeg scheduling are the following:

- We never run out of storage (that is, we never dispatch tasks when there is no energy); this is obvious from the algorithm that does not allow tasks to run after E_{min}.
- We start charging the storage unit when, either it is empty or there is no more sufficient energy to guarantee the feasible execution of all future occurring tasks i.e. the system has no more slack energy.
- The charging process aims to charge at the maximum level, E_{max}, provided there is sufficient slack time. Slack time is computed when entering the wait state and subsequently decremented at each time instant.
- We stop repleneshing the storage unit i.e. the processing device enters the active state as soon as there is no more slack time or the storage unit is fully replenished. Such condition can be easily detected through an interrupt mechanism and adequate circuitry between energy storage unit and processing device.
- We only waste recharging power when there are no pending tasks and the storage unit is full.

The framework of EDeg scheduling is as follows:

Algorithm 1. Earliest deadline with energy guarantee algorithm (EDeg)

```
while (1) do
  while PENDING=true do
    while (E(t) > E_min and Slack.energy(t) > 0) do
      execute()
    end while
    while (E(t) < E_max and Slack.time(t) > 0) do
      wait()
      end while
    end while
    while PENDING=false do
      wait()
    end while
end while
```

5.2. Efficiency

The computations of Slack.energy(t) and Slack.time(t) are thus the major keys to the operation of the EDeg algorithm. As shown in [19], the slack time of a periodic task set at a given time instant can be obtained on-line by computing the dynamic

EDL schedule, with complexity $O(K.n)$. n is the number of periodic tasks, and K is equal to $\lfloor R/p \rfloor$, where R and p are, respectively the longest deadline and the shortest period of current ready tasks.

The complexity for computing the slack energy is $O(K.n)$ too. As EDeg has low and constant space requirements, this makes it easily implementable on many low-power, unsophisticated hardware platforms including micro-controllers.

A suggestion to improve the efficiency of the scheduler in terms of overhead is to compute statically a lower bound on the slack time and a lower bound on the slack energy. And we use them at run-time instead of exact values which would require on-line computations. The effect will be only, first to stop charging earlier and second to stop executing tasks earlier. As a consequence, decreasing the processor overheads due to computations will cause increasing the number of tasks preemptions.

5.3. Measurement support

As any power management algorithm, EDeg typically needs information about available energy resources. Many battery operated devices, ranging from hand-helds to laptops, provide the facility to monitor the residual capacity of the battery. A second related measurement is the variability in this energy supply. A method to estimate the energy input then is to measure the current flowing out of the harvesting source and its voltage. This immediately yields the instantaneous power input at any time point. Data about when and how much environmental energy is available is directly provided by these measurements. Also, these measurements can be tracked at the desired resolution in time to estimate the variability of the energy source.

5.4. Illustrative example

Let us consider the previous periodic task set where each task is now characterized by its worst case energy consumption. $\tau = \{\tau_i | 1 \leqslant i \leqslant 3, \tau_i = (C_i, D_i, T_i, E_i)\}$. Let $\tau_1 = (3, 6, 9, 8), \tau_2 = (3, 8, 12, 8)$ and $\tau_3 = (3, 12, 18, 8)$. We assume that the energy storage capacity is $E = 6$. For simplicity, the rechargeable power, P_r, is constant along time and equals 2. We note that $U_p = 0.75, U_e = 2$ and consequently the necessary feasibility condition related to energy constraints, $U_e \leqslant \overline{P_r}$ is satisfied. When every task executes, it consumes energy with average power given by $\frac{E_i}{C_i}$ i.e. 8/3, roughly equal to 2.66. Whenever the processor is active, the storage unit discharges and the decreasing rate is 0.66 in average.

By scheduling the task set τ according to EDS within the first hyperperiod i.e. from 0 to 36, disregarding energy limitations, we verify that τ is temporally-feasible (see Fig. 2). However, when considering the energy constraints, the task set reveals to be not feasible since the storage unit becomes empty at $t = 9$. As the system immediately stops, we conclude that the deadline miss ratio during the hyperperiod is about 33% as shown by Fig. 5.

Under static EDL scheduling, the system stops at $t = 27$ due to energy shortage. The deadline miss ratio is consequently 78% (see Fig. 6). Even if deadline missing occurs later compared with EDS, EDL does not provide acceptable performance. While energy is wasted in the early part of the schedule, afterwards, it reveals to be insufficient to cope with surplus of energy demand due to execution postponement.

Applying a classical greedy scheduler such as EDS or EDL reveals impossible because the system fails as soon as energy shortage occurs. In contrast, singularity of EDeg lies in inserting processor idle times for recharging the storage unit only whenever necessary. And the recharging time is computed from the current slack time of the task set in order to still guarantee all the deadlines while avoiding energy overflow. Then, the processor is let inactive as long as the energy storage has not filled completely ($E = E_{max}$) and the latest start time of the next periodic task has not been attained. Clearly, this amounts to make slack stealing controlled by the residual capacity of the energy storage. The EDeg schedule is described in Fig. 7.

Let us explain how EDeg dynamically constructs the schedule. At time 0, the residual capacity i.e. remaining energy is maximum since the storage is full. τ_1 is the highest priority task, executes until time 3 and consumes 8 energy units. At time 3, the residual capacity is given by $E_{max} - E_2 + Pr * C_2 = 4$. τ_2 has now the highest priority and the slack energy is undefined

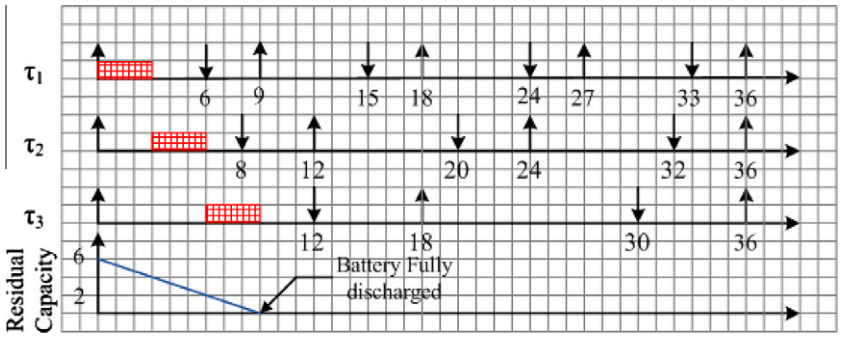


Fig. 5. EDS scheduling.

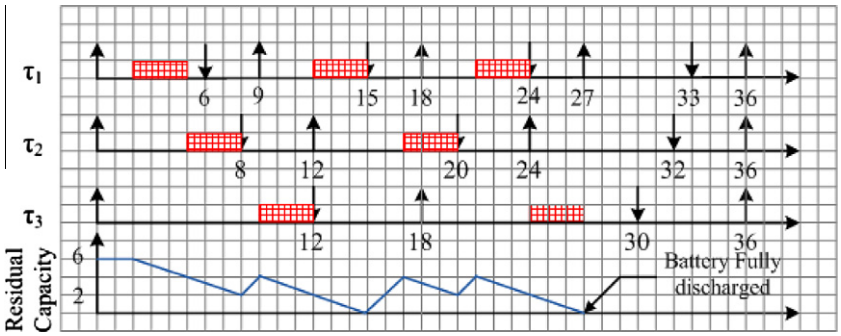


Fig. 6. EDL scheduling.

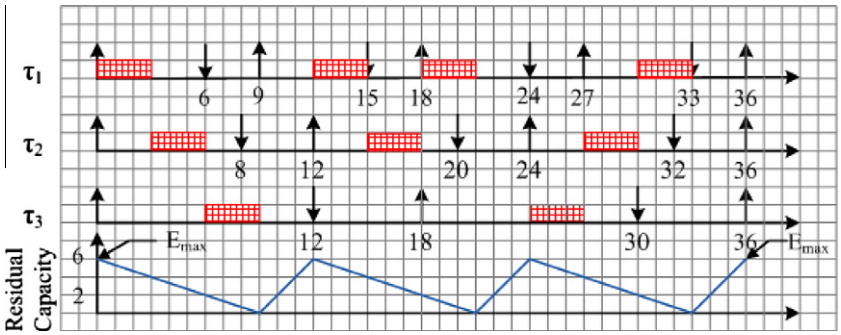


Fig. 7. EDeg scheduling.

since no future instance has a deadline less than the deadline of τ_2. τ_2 executes completely until time 6 and consumes 8 energy units. The residual capacity then equals 4 energy units. τ_3 has now the highest priority and completely executes until time 9 where the residual capacity falls at 0. As the storage is empty, the processor has to remain idle as long as the storage has not fulfilled (predicted at time 12) and the latest start time has not been attained (at time 12 which is computed using dynamic EDL). τ_1 is then the highest priority task, executes till time instant 15 where the residual capacity is 4. τ_2 is then the highest priority task, executes till time 18 where the residual capacity equals 4. τ_1 executes from time 18 until time 21 where the residual capacity falls to 0. The processor then remains idle as long as the storage has not fulfilled (predicted at time 24) and the latest start time has not been attained (at time 24 which is computed using dynamic EDL).

At time 24, the highest priority task, say τ_3 executes and completes exactly at time 27 where the residual capacity equals 4. τ_2 executes till time 30. τ_1 then executes until time 33 where the energy storage is empty. The processor is let idle for recharging until the end of the hyperperiod at $t = 36$. We note that neutral operation is guaranteed since the storage is fully charged at the end of the hyperperiod.

In contrast to EDS and EDL, EDeg feasibly schedules the task set, with the same characteristics of the storage unit and the power source profile.

6. Performance evaluation

6.1. Simulation details

Two main issues need to be discussed in order to evaluate *EDeg*: energy storage size and operational performance level in terms of deadline missing. To evaluate the effectiveness of the proposed EDeg algorithm, we develop a discrete-event simulation in C/C++. In the simulator, we implement *EDeg*. For sake of comparison, we also implement *EDS* and *EDL* and two heuristics, respectively called *EDd_1* and *EDd_A*. *EDd_A* is the EDS scheduler that discards all the ready instances whenever the storage unit is empty and consequently lets the processor idle until the next release time. *EDd_1* is the EDS scheduler that discards only one instance (the highest priority one) whenever the storage unit is empty and then lets the processor idle until the next release time.

We report here part of a performance analysis which consists of two simulation experiments. In the first experiment, we measure the percentage of feasible instances during one hyperperiod. In the second one, we measure the remaining energy in the storage unit (also called residual capacity) along time up to the first deadline violation or the end of the first hyperperiod.

Since the performance of the above algorithms is severely affected by the properties of the arriving tasks, we use a simulator that generates 30 tasks with least common multiple of the periods equal to 3360. The worst-case computation times are set according to the processor utilization, U_p. Deadlines are less than or equal to periods and greater than or equal to the computation times ($C_i \leqslant D_i \leqslant T_i$). The rechargeable power is variable with time. A random number generator enables us to produce, for every quantum of time, a power energy profile with minimum value 1 and a maximum value, here 9, as an input of the simulator. The energy consumptions are set according to the energy utilization, U_e, which necessarily verifies $U_e \leqslant \overline{P_r}$. We assume that the energy storage is fully charged at the beginning of the simulation. After a deadline violation is detected, the simulation terminates for *EDL* and *EDS*. Under *EDd_1* and *EDd_A*, the simulation continues until the end of the hyperperiod. The simulation is repeated for 1000 task sets under a given processor utilization. For the fair comparison of *EDeg*, *EDS*, *EDL*, *EDd_1* and *EDd_A*, all simulations are performed under the same condition.

6.2. Experiment 1: size of the energy storage and deadline miss rate

First, we take interest in the average ratio of instances that meet their deadlines with the five scheduling algorithms by varying the energy storage capacity. This metrics enables us to deduce two measures. The first one gives us an indication about the ratio of time during which there is no deadline violation. The second one gives, for each scheduler and for a given processor utilization ratio, the minimum size of the storage that ensures time and energy feasibility. We report here the results of three simulation studies where the processor utilization U_p is set to 0.3, 0.6 and 0.9, respectively. Fig. 8 depicts the percentage of instances that meet their deadlines over the energy storage capacity E. For each task set, we compute E_{feas} as the minimum storage capacity which permits to achieve neutral operation i.e. time validity of the schedule with battery fully recharged at the end of the hyperperiod. Consequently, we make vary E so as all task sets turn out to be feasible under EDeg.

For $U_p = 0.3$, the percentage of feasible instances with *EDeg* algorithm is 100% when the energy storage capacity is at least 1700 energy units. Consequently, $E_{feas} = 1700$. Under *EDS*, maximum idle time is made available at the end of the hyperperiod. As the storage is initially full, the energy which is available will be used to execute task instances. EDS idle times in the early part of the schedule do not permit the storage to fully recharge because the processor is quasi-continuously busy. In the EDL schedule, tasks are delayed as much as possible and the storage unit recharges sufficiently during the idle times to execute task instances in the busy intervals without making the storage empty and the system failed. As a result, *EDL* will give better results than *EDS* in terms of number of feasible instances before first deadline violation.

In summary, both *EDS* and *EDL* are greedy schedulers which perform badly in energy constraint environment and do not permit to build a valid schedule. Consequently, we are interested in examining the performances of two best-effort heuristics that provide respective solutions in order to cope with energy lack situations and continue to operate despite a deadline violation. When the energy storage unit is empty, *EDd_1* discards the highest priority instance that is ready. Then, we let the processor idle until the next release time. *EDd_1* performs significantly better than *EDd_A* that systematically discards all ready instances when the energy storage unit is empty. Even if no time is wasted in attempting to execute tasks which will not meet their deadlines, the total number of discarded instances is higher with *EDd_A* compared with *EDd_1*. However, when the size of the energy storage increases, the number of discarded instances will be close to each other and consequently the difference between *EDd_1* and *EDd_A* will not be significant. As observed in Fig. 8a, 100% task instances meet their deadlines i.e. task sets are feasible when the energy storage capacity is 5800 for *EDS*, *EDd_1* and *EDd_A*. That means that the storage unit must be more than 3 times bigger with *EDS*, *EDd_1* and *EDd_A* to maintain zero deadline miss, compared with *EDeg*.

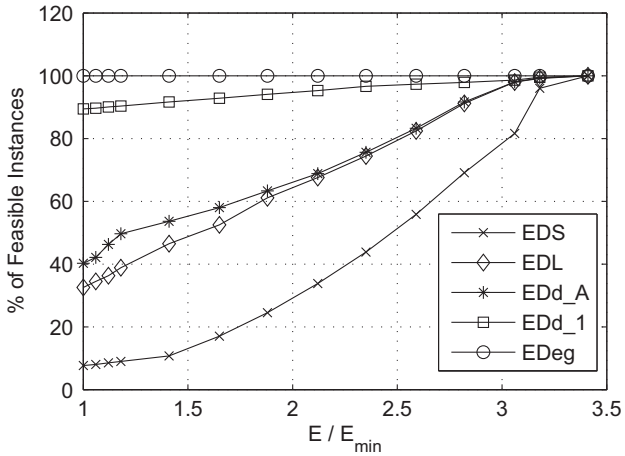
For $U_p = 0.6$ (see Fig. 8b), the percentage of feasible instances under *EDeg* algorithm is 100% when the energy storage capacity is at least 2400 energy units. Furthermore, the energy storage capacity that is required for *EDS*, *EDd_1* and *EDd_A* to ensure 100% feasible instances is 5900. That means that *EDeg* can provide the same level of performance with a storage unit which is about 2.5 times less. We observe that the relative performance gain of *EDeg* in terms of capacity savings is decreasing when the processor utilization rate is increasing. For higher values of processor utilization, savings are decreasing, yet they are still significant.

For $U_p = 0.9$ (see Fig. 8c), *EDeg* obtains capacity savings of about 50% compared with *EDL*, *EDS*, *EDd_1* and *EDd_A*. Let us notice that all schedulers require exactly the same storage size when $U_p = 1$ since the processor is continuously busy.

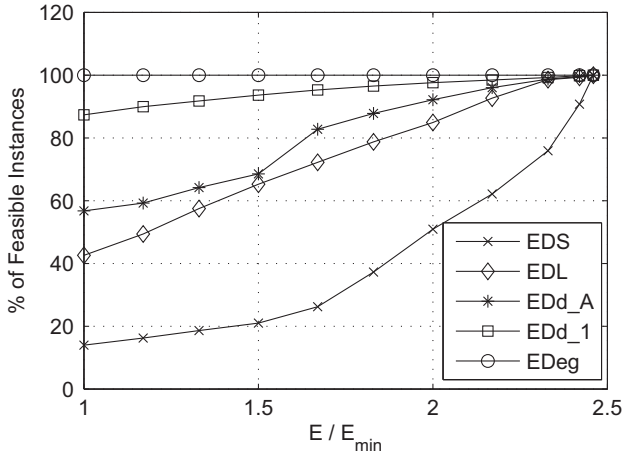
In summary, experiment 1 points out that the proposed EDeg scheduler is very effective in reducing deadline miss rate and storage size for a real-time system with energy harvesting facilities. And lower is the processor utilization rate, higher is the capacity saving.

6.3. Experiment 2: remaining energy in the storage unit

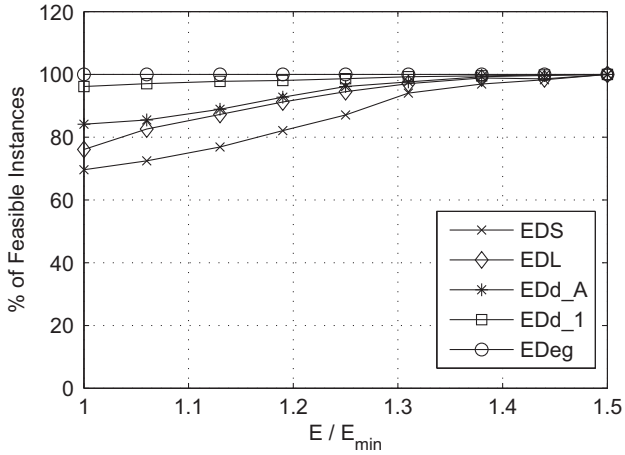
Finally, we are interested in the remaining energy stored in the system. Fig. 9 aims to illustrate for $U_p = 0.3$, how the energy level in the storage unit changes along time. We only report this information for the three following schedulers, *EDeg*, *EDS* and *EDL*. Under *EDS*, the remaining energy is decreasing until the storage unit is empty or not sufficient to execute the highest priority instance. On the contrary, *EDL* will benefit from the idle times which are present in the early part of the hyperperiod in order to recharge the storage unit. This is why *EDL* stores significantly more energy than *EDS*. In our simulation, *EDS* and *EDL*, respectively stop at about 10% and 20% of the hyperperiod (Fig. 9). *EDeg* runs as *EDS* except that, whenever there is no sufficient energy to execute the highest priority task, the processor starts sleeping but does not stop definitively. Recall that in our model, the instantaneous power that is consumed by a task may be arbitrarily high. Conse-



(a) Low processor utilization (U_p=0.3)



(b) Medium processor utilization (U_p=0.6)



(c) High processor utilization (U_p=0.9)

Fig. 8. Percentage of feasible instances. (a) Low processor utilization ($U_p = 0.3$). (b) Medium processor utilization ($U_p = 0.6$). (c) High processor utilization ($U_p = 0.9$).

quently, whenever a task requires to be run, its energy consumption requirement is compared with the amount of energy that will be available during the next unit of time. This implies that the energy level will decrease systematically when exe-

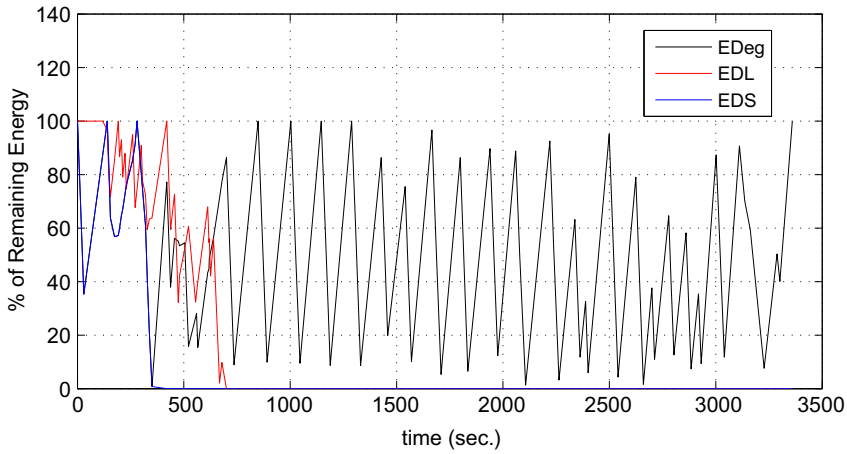


Fig. 9. Variation of the remaining energy level.

cuting a task without necessarily attaining the minimum level i.e. 0. Then, the storage unit will recharge until being fulfilled and as long as the system will be able to meet all the deadlines. As seen in Fig. 9, the energy level is decreasing until $t = 350$, where the remaining energy is not sufficient to execute the next task instance. Therefore the processor idles until $t = 420$, where the energy level is around 77% of the total capacity. This phenomena is repeating along the hyperperiod. The storage unit is fulfilled again at the end of the hyperperiod, which means that the application runs in an energy neutral operation mode.

7. Conclusion

Energy-aware design becomes a more important issue in embedded systems that require best use of available power sources and deliver high performance at the same time. The performance of a practical energy harvesting real-time system, measured by the deadline miss rate, heavily depends upon the stored energy and the energy harvested from the environment. Unfortunately, the scavenging power is time-varying and thus very unstable. We target a scheduling framework for embedded systems with variable power constraints which need to operate perennially thanks to the environmental energy. In this paper, our focus was on scheduling periodic tasks with deadlines on a uniprocessor and fixed-speed system which is powered by a renewable energy storage with limited capacity such as a battery or a capacitor.

This work presents the following contributions to research: our scheduler is model-free with respect to the energy source. It can be implemented in any energy harvesting system without the need for a priori information about the source which may be uncontrollable and time-varying. We propose an efficient way of scheduling tasks, based on the on-line computation of the slack time and the slack energy which is a new concept dedicated to hard deadline tasks with regenerative energy constraints.

We started by describing the preemptive scheduling algorithm, namely EDeg, that is a variation of EDF. EDeg has been designed for any set of time critical tasks, periodic or not, given any energy source profile with a variable power production and an energy storage unit with limited capacity. The simulation study reports the performance of EDeg, measured by the deadline miss rate compared with classical EDF. It shows that EDeg outperforms EDS and EDL for all processor utilization factors. Several interesting issues need further attention. To expand the applicability of our scheduling framework, we would like first to incorporate additional power management techniques including voltage/frequency scaling and dynamic power management to support more effective power-aware designs.

References

[1] Schmitz MT, Al-Hashimi BM, Eles P. System-level design techniques for energy-efficient embedded systems. Kluwer Academic Publishers; 2004.
[2] Kansal A, Hsu J, Zahedi S, Srivastava MB. Power management in energy harvesting sensor networks. ACM Trans Embedded Comput Syst 2007;6(4).
[3] Moser C, Brunelli D, Thiele L, Benini L. Real-time scheduling for energy harvesting sensor nodes. Real-Time Syst 2007;37(3):233–60.
[4] Chetto H, Chetto M. Some results of the earliest deadline scheduling algorithm. IEEE Trans Softw Eng 1989;15(10):1261–9.
[5] Sinha A, Chandrasan A. Dynamic power management in wireless sensor networks. IEEE Design Test Comput 2001;18(2):62–74.
[6] Raghunathan V, Kansal A, Hsu Jason, Friedman Jonathan, Srivastava Mani. Design considerations for solar energy harvesting wireless embedded systems. In: Proceedings of the fourth international symposium on information processing in sensor networks, 2005. p. 457–62.
[7] Allavena A, Mosse D. Scheduling of frame-based embedded systems with rechargeable batteries. Workshop Power Manage Real-time Embedded Syst 2001.
[8] Shih WK, Liu JWS. On-line scheduling of imprecise computations to minimize error. SIAM J Comput 1996;25(5):1105–21.
[9] Rusu C, Melhem RG, Mosse D. Multi-version scheduling in rechargeable energyaware real-time systems. In: Proceedings of 15th Euromicro conference on real-time systems, 2003. p. 95–104

[10] Moser C, Chen J-J, Thiele L. Reward maximization for embedded systems with renewable energies. In: Proceedings of 14th IEEE international conference on embedded and real-time computing systems and applications, 2008. p. 247–256.
[11] Kansal A, Hsu J. Harvesting aware power management for sensor networks. In: IEEE Proceedings of design automation conference, 2006.
[12] Hsu J, Zahedi S, Kansal A, Srivastava M, Raghunathan V. Adaptive duty cycling for energy harvesting systems. In: Proceedings of the international symposium on low power electronics and design, 2006. p. 180–5
[13] Liu C-L, Layland J-W. Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 1973;20(1):46–61.
[14] Liu S, Qiu Q, Wu Q. Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting. In: Proceedings of the conference on design, automation and test in Europe, 2008. p. 236–41.
[15] Jayaseelan R, Mitra T, Li X. Estimating the worst-case energy consumption of embedded software. In: Proceedings of 12th IEEE real-time and embedded technology and applications symposium, 2006. p. 81–90.
[16] Liu J-W-S. Real-time systems. Prentice-Hall; 2000.
[17] Leung J-Y-T, Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks. Perform Evaluat 1982;2(4):237–50.
[18] Dertouzos ML. Control robotics: the procedural control of physical processes. Proc Int Federat Inform Process Cong 1974:807–13.
[19] Silly-Chetto M. The EDL server for scheduling periodic and soft aperiodic tasks with resource constraints. Real-Time Syst 1999;17(1):1–25.

Hussein EL Ghor is currently pursuing the Ph.D. degree at the University of Nantes, France. He has been a member of the Real Time System group at IRCCyN Laboratory from 2009. He received the engineering degree from the Lebanese University, Lebanon, in 2002. His research interests concern real-time scheduling and partitioning with particular emphasis on energy harvesting systems.

Maryline Chetto received the degree of Docteur in control engineering and the degree of Habilitée à Diriger des Recherches in Computer Science from the University of Nantes, France, in 1984 and 1993, respectively. She is currently a full Professor with the Institute of Technology of Nantes and is conducting her research at IRCCyN Laboratory. Her main research interests are scheduling and power management for real-time systems. She has published more than 70 journal articles and conference papers in the area of real-time systems.

Rafic Hage Chehade received the Ph.D. degree in electrical engineering from the University of Lille 2, France, in 1989. His doctoral thesis was on medical instrumentation. From 1990 to 1997, he was teaching in the University of Lille 1 and Ecole d'ingenieurs HEI. He joined the Lebanese University in 1997 and is currently an Associate Professor in the Institute of Technology of Saida. His research interests include digital communication and embedded technology.