EH-EDF: An On-line Scheduler for Real-Time Energy Harvesting Systems

Hussein EL Ghor*, Maryline Chetto* and Rafic Hage Chehade†
*IRCCyN Lab
University of Nantes, Nantes, France
Email: elghorh@irccyn.ec-nantes.fr
maryline.chetto@univ-nantes.fr
†IUT Saida
Lebanese University, Saida, Lebanon
Email: rhagechehade@ul.edu.lb

***Abstract*— In this paper, we undertake the problem of on-line real-time scheduling in a uniprocessor platform that is powered by a renewable energy storage unit and uses a recharging system such as photovoltaic cells. Since executing tasks require a certain amount of energy, we must take into account the characteristics of the energy source, capacity of the energy storage as well as energy consumption of the tasks, and time. For this sake, we present a scheduling algorithm called $EH-EDF$ (Energy Harvesting - Earliest Deadline First). In such algorithm, scheduling decisions are taken at run-time without having any prior knowledge about the characteristics of the future energy production and task characteristics.**

I. Introduction

A scheduling algorithm is said to be *on-line* if its decisions are taken at run-time without having any prior knowledge about the characteristics of the future tasks [1].

When dealing with real-time systems that take time as the only limitation, we have to differentiate between *underloaded* and *overloaded* real-time systems. A real-time system is said to be underloaded if there exists a feasible schedule for the workload. It is important here to indicate that EDF (Earliest Deadline First) algorithm is optimal in underloaded hard real-time systems. This means that EDF is guaranteed to meet all the task deadlines. On the contrary, overloaded real-time systems does not have a feasible schedule where all tasks meet their deadlines. In this case, the objective will be to maximize the performance of the system. Here, we focus on underloaded real-time systems where energy is the most important limiting factor.

Recently, we presented a scheduling algorithm called *EDeg* (Earliest Deadline with energy guarantee) [6]. In such algorithm, we assume that the set of tasks to be scheduled is well known *off-line*. It is effectively the case when the system is only composed of periodic tasks. $EDeg$ algorithm is called *clairvoyant* algorithm since it must know in advance both the energy source profile and the characteristics of the tasks (arrival time).

The work presented in this paper is the on-line version of the $EDeg$ algorithm. The advantage of this scheduler is that it is completely on-line while $EDeg$ is totally clairvoyant. Now, we are studying an algorithm which ignores the future energy production and only knows the consumption of tasks which are already released on the machine.

The rest of this paper is organized as follows. We first present the related work necessary background in Section II. Section III introduces the models and terminology used in this paper. Fundamental concepts are described in section IV. Section V is the presentation of $EH-EDF$ scheduling algorithm. Finally, we conclude the paper in Section VI.

II. Related Work

Over the past years, a significant amount of researchers have focused their attention on the design of competitive on-line scheduling algorithms as well as on the inherent limitations of these algorithms. Most of works have disregarded an important aspect, namely energy management or have assumed that the energy capacity is sufficient to execute all tasks. In this domain, the Earliest Deadline First is the most notable scheduling algorithm [7], this is because of its optimality. EDF is an on-line algorithm that schedules at each instant of time t, the ready task with a deadline closest to t.

Now, energy constraint is added as an important factor when addressing real-time scheduling problems. This is mainly because of the great advances in both hardware and software technology that have enabled system designers to develop large, complex embedded systems. Such systems consume a large amount of power and rely mainly on a limited energy storage. Under these new conditions, hundreds of research work were done either to minimize the total energy consumption without violating deadlines or to maximize the performance of hard energy constrained systems with a fixed energy budget that is not sufficient to meet al deadlines.

Very recently [4], we presented a scheduling algorithm, $EDeg$ (Earliest Deadline with energy guarantee), that takes into consideration the limits of both time and energy. $EDeg$ relies on two basic concepts: *slack time* and *slack energy*. The main idea behind $EDeg$ 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 execute all future occurring tasks. When this condition is not

978-1-4577-1846-5/11/$26.00 ©2011 IEEE

verified, the processor has to stay idle so that the storage unit recharges as much as possible and as long as all the deadlines can still be met despite execution postponement. Later in [6], same authors proved by performance evaluations the efficiency of this scheduler. However, $EDeg$ is a clairvoyant algorithm since it needs the characteristics of the future tasks and the energy source profile to build an optimal schedule.

III. Model and Terminology

We consider a uniprocessor system that is composed of *aperiodic* tasks that are known by the system at the time of their arrival. An aperiodic task set can be denoted as follows: $\Psi = \{T_i, i = 1, \cdots, n\}$. Every task T_i is characterized by (r_i, C_i, E_i, D_i) where r_i is the arrival time of task T_i. Each request of T_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 T_i occurs D_i units after each request by which task must have completed its execution. We assume that $0 < C_i < D_i$ for each $1 \leq i \leq n$. We define the processor load as $L_p = \sum_{i=1}^{n} \frac{C_i}{D_{max}}$ and the energy load as $L_e = \sum_{i=1}^{n} \frac{E_i}{D_{max}}$ where D_{max} is the longest deadline in the task set Ψ.

L_p is the percentage of processing time if tasks of Ψ are solely executed on the device. One can interpret the energy load L_e, measured in joules/s, as the average power consumed by Ψ when executing on the device.

A. Energy Source

We assume that ambient energy is harvested and converted into electrical power. We cannot control the energy source but we can predict the expected availability with a lower bound on the harvested source power output, namely $P_s(t)$. 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. This power is then the instantaneous charging rate that incorporates all losses caused by power conversion and charging process. 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.

B. Energy Storage

We consider a uniprocessor system that uses an ideal energy storage unit (supercapacitor or battery) of nominal capacity 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.

C. Definitions

To be more clear, we include several definitions and terms that are used in this paper.

Definition *1:* An algorithm is said to be *on-line* if its scheduling decisions are taken at run-time without having any prior knowledge about the characteristics of the future tasks.

Definition *2:* An algorithm is said to be *clairvoyant* if it knows in advance the arrival time of the tasks and the energy profile produced by the source. Otherwise, it is non-clairvoyant.

IV. Fundamental Concepts

A. Slack Time

By definition, 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 [3]. In these systems, the purpose of slack time analysis is to improve the response time of aperiodic tasks or to increase their acceptance ratio. A means of determining the maximum amount of slack which may be stolen, without jeopardizing the hard timing constraints, is thus key to the operation of the so-called slack-stealing algorithms. Determining slack time is realized at run-time by computing the so-called dynamic EDL (Earliest Deadline as Late as possible) schedule [3].

In this work, we consider that tasks are aperiodic and known by the system at the time of their arrival. So, we are obliged to schedule tasks according to an on-line scheduling algorithm. In this case, we compute the slack time based on the tasks that are ready at current time t.

Let us consider a task set Ψ of *aperiodic* tasks. $\Psi = \{T_i, i = 1, \cdots, n\}$. Every task T_i is characterized by (r_i, C_i, E_i, D_i). Denote Ψ' as the set of ready tasks at current time t. Then the slack time at current time t is computed as:

$$Slack.time(t) = \min_{1 \leq j \leq n} (D_j - sum(C_i \mid D_i < D_j)) \quad (1)$$

B. Illustrative Example

Consider a task set $\Psi = \{T_i, i = 1, \cdots, 4 \; and \; T_i = (r_i, C_i, D_i)\}$. Let $T_1 = (0, 3, 18)$, $T_2 = (4, 2, 12)$,$T_3 = (5, 3, 24)$,$T_4 = (0, 4, 16)$ and $T_5 = (8, 3, 20)$. We assume that we have to calculate the slack time at time $t = 6$. We note that $L_p = \sum_{i=1}^{n} \frac{C_i}{D_{max}} = \frac{5}{8} < 1$.

By scheduling the task set Ψ according to EDF, we first execute T_4 from time $t = 0$ *to* 4 and then T_2 from time $t = 4$ *to* 6. Now we have to compute the slack time at time $t = 6$. It is obvious that tasks T_1 and T_3 are ready at this time. According to equation 1, we have to compute the slack time for the ready tasks and then the minimum of these slack times will be the slack time at time $t = 6$.

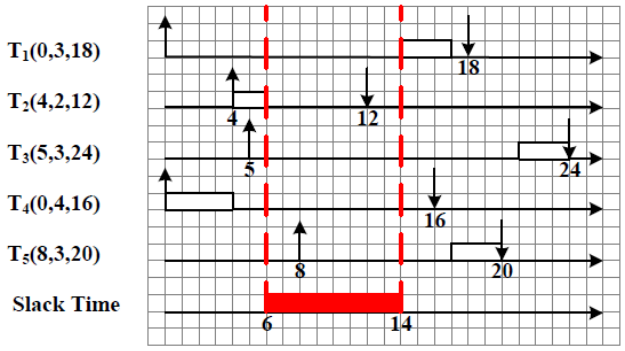

Fig. 1. Slack time at time 6

V. Energy Harvesting-Earliest Deadline First (EH-EDF) Scheduling Algorithm

In this section, we consider that we are obliged to schedule tasks according to an on-line algorithm that ignores the future energy production and the arrival time of tasks. It knows only the energy consumption of tasks that are released on the processor.

A. Presentation of the Algorithm

The intuition behind $EH-EDF$ algorithm is to schedule *aperiodic* tasks as soon as possible according to EDF.
We take a hypothesis that a task can consume energy with any power. In this case, and before attempting to execute a task, we must ensure that there is a sufficient energy to execute it. When the battery becomes empty or unable to execute a task, we have to compute the *slack time* ($Slack.time(t)$) based on the tasks presented in the list. The scheduler will then let the processor idle until the battery replenishes or the slack time becomes zero. We must note that during recharging, when a new task arrives, we must insert it in the list, update slack time and continue to let the processor idle.
The advantage of this scheduler is that it is completely on-line while $EDeg$ is totaly clairvoyant.
Following the idea described above, we propose the Energy Harvesting - Earliest Deadline First ($EH-EDF$) algorithm.

The major components of the $EH-EDF$ algorithm are: $E(t)$ and $Slack.time(t)$. 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. We define $Slack.time(t)$ as the slack time at time t. We use the function $wait()$ to put the processor to sleep and function $execute()$ to put the processor to run the ready job with EDF.

The framework of the $EH-EDF$ scheduling algorithm is as follows:

$EH-EDF$ charges the energy storage to the maximum level provided there is sufficient slack time. Slack time is computed when entering the *wait* state and decremented at each time instant. We stop charging the storage unit 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 storage unit and processing device. Therefore, we waste recharging power only when there are no waiting tasks in the ready list and the storage unit is full.

Algorithm 1 [Energy Harvesting-Earliest Deadline First ($EH-EDF$)]

Input: A Set of aperiodic Tasks $\Psi = \{T_i | T_i = (r_i, C_i, D_i, E_i) \ \ i = 1, \cdots, n\}$ According to EDF, current time t, battery with capacity ranging from E_{min} to E_{max}, energy level of the battery $E(t)$, source power $P_s(t)$.
Output: $EH-EDF$ Schedule.

```
1: t = 0
2: while (1) do
3:    while E(t) > 0 do
4:       /*Execute Tasks according to EDF*/
5:       execute()
6:       t = t + 1
7:       if t=arrival time of T_i then
8:          insert T_i in the list of ready tasks
9:       end if
10:   end while
11:   /*If E(t) ≤ 0, then we have to compute the slack time*/
12:   Compute Slack.time(t)
13:   /*Measure the energy storage after recharging it for
      time (Slack.time(t))*/
14:   while E(t) < E_max and Slack.time(t) > 0 do
15:      wait()
16:      t = t + 1
17:      if t=arrival time of T_i then
18:         insert T_i in the list of ready tasks
19:         update Slack.time(t)
20:      end if
21:   end while
22: end while
```

B. Illustrative Example

Consider a task set Ψ with five aperiodic tasks as in the above example such that $\Psi = \{T_i | 1 \leq i \leq 5\}$ where $T_i = (r_i, C_i, D_i, E_i)$. Let $T_1(0,3,18,9)$, $T_2(4,2,12,12)$, $T_3(5,3,24,7)$, $T_4(0,4,16,10)$ and $T_5(8,3,20,9)$. The energy storage capacity is assumed to be equal to 10 energy units. For sake of simplicity, the rechargeable power, P_s is constant along time and equals 2. The processor load $L_p = \sum_{i=1}^{n} \frac{C_i}{D_{max}} = \frac{15}{24} < 1$ and the energy load $L_e = \sum_{i=1}^{n} \frac{E_i}{D_{max}} = \frac{47}{24} < P_s = 2$.
By scheduling the task set Ψ according to EDF without energy constraints, it is clear that Ψ is temporally feasible. On the contrary, when energy constraints are taken into consideration, Ψ reveals to be not feasible since the storage becomes empty at $t = 6$. When applying $EH-EDF$ (figure 2), the task set Ψ is scheduled according to EDF from $t = 0$ till $t = 6$ where the energy storage capacity is empty. Then, we have to insert a processor idle time for recharging the storage.

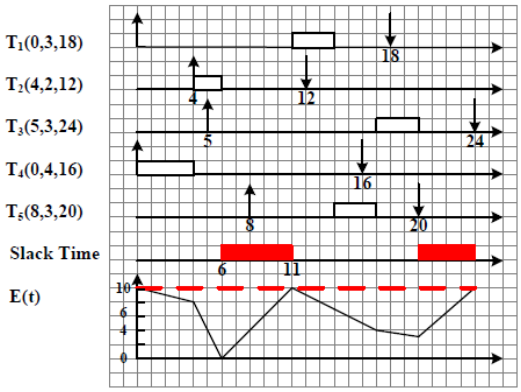

Fig. 2. Illustrative example on $EH - EDF$

The time of recharging is computed from the current *slack time* of the task set in order to still guarantee all the deadlines while avoiding energy overflow. In details, the energy storage is full at time $t = 0$, T_4 is the highest priority task and it is executed according to EDF until $t = 4$ where the energy storage capacity $E(t) = E_{max} - E_1 + P_s C_1 = 8$ energy units. At time $t = 4$, T_2 is ready and has the highest priority, it is executed until $t = 6$, where the energy storage is empty. As the storage is empty, the processor has to remain idle as long as the storage has not fulfilled and the slack time is not zero. According to equation 1, we have to compute the slack time of all released tasks and then the slack time of the system will be the minimum of the computed slack times. At $t = 6$, T_1 and T_3 are released. Thanks to equation 1, The slack time of T_1 and T_3 is equal to 9 and 15 respectively. Therefore, the $Slack.time(t = 6) = 9$. Consequently, the processor has to stay idle until $t = 15$ and the energy storage is recharged. But it is important to note that at $t = 8$, task T_5 is released and therefore, we have to update the slack time. The slack time for T_5 is 6. As a result, the $Slack.time(t = 8) = 6$, that is the minimum of the slack times. The battery is recharged until $t = 11$ where it is full. Thus we stop recharging for not losing any wasted energy.

At $t = 11$, the energy storage is equal to 10 energy units and T_1 has the highest priority. It is executed until $t = 14$ and the remaining energy $E(t) = 7$ energy units. T_5 is then the highest priority task and is executed until $t = 17$ where $E(t) = 4$ energy units. At $t = 17$, T_3 is the highest priority task, and it is executed until $t = 20$ where the remaining energy $E(t)$ is equal to 3 energy units. The processor now has no tasks to be executed and the processor has to remain idle until $t = 24$ where the battery capacity is full again.

In contrast to EDF, $EH - EDF$ feasibly schedules the task set Ψ, given the characteristics of the storage unit and the power source profile.

VI. Conclusion

We presented in this article the framework of an on-line monoprocessor scheduling algorithm, namely $EH - EDF$. The advantage of this scheduler is that it is completely on-line and not clairvoyant such as EDF. The future work will focus on the performance of $EH - EDF$ relative to non-idling schedulers such as EDF and to an optimal clairvoyant scheduler.

Acknowledgment

The work presented in this paper is sponsored by CEDRE project which corresponds to a bilateral collaboration between University of Nantes and the Lebanese University.

References

[1]Allan Borodin and Ran El-Yaniv, *Online computation and competitive analysis.* Cambridge University Press, Cambridge, 1998.

[2]Buttazzo G. *Hard real-time computing systems: predictable scheduling algorithms and applications*, 2^{nd} edition. Springer, Berlin, 2005.

[3]M. Silly-Chetto, *The EDL Server for scheduling periodic and soft aperiodic tasks with resource constraints*. The Journal of Real-Time Systems, 17: 1-25, 1999.

[4]M. Chetto and H. Ghor, R *eal-time scheduling of periodic tasks in a mono-processor system with rechargeable energy storage*. WIP Proceedings of the 30th IEEE Real-Time Systems Symposium, December 2009.

[5]V. Devadas, F. Li, H. Aydin, *Competitive analysis of online scheduling algorithms under hard energy constraint*, Real-Time Systems, Volume 46, pp. 88-120, 2010.

[6]H. El Ghor, M. Chetto and R. Hajj Chehade, *A Real-Time Scheduling Framework for Embedded Systems with Environmental energy Harvesting*, In Computers & Electrical Engineering, 2011.

[7]C.-L. Liu, J.-W. Layland, *Scheduling algorithms for multiprogramming in a hard real-time environment.* Journal of the Association for Computing Machinery, Volume 20, Issue 1, pp. 46-61, 1973.