HAL
open science

Toward an Optimal Fixed-Priority Algorithm for Energy-Harvesting Real-Time Systems

Yasmina Abdeddaïm, Younès Chandarli, Damien Masson

To cite this version:

Yasmina Abdeddaïm, Younès Chandarli, Damien Masson. Toward an Optimal Fixed-Priority Algorithm for Energy-Harvesting Real-Time Systems. RTAS 2013 WiP, Apr 2013, United States. pp.45–48. hal-00796646

HAL Id: hal-00796646

https://hal.science/hal-00796646

Submitted on 7 Mar 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Toward an Optimal Fixed-Priority Algorithm for Energy-Harvesting Real-Time Systems

Yasmina Abdeddaïm[1], Younès Chandarli[1,2], and Damien Masson[1]

Université Paris-Est, LIGM UMR CNRS 8049,

[1]ESIEE Paris, 2 bld Blaise Pascal, BP 99,
93162 Noisy-le-Grand CEDEX, France

[2]Université Paris-Est Marne-la-vallée, 5 bld Descartes,
77454 Marne-la-Vallée Cedex 2, France

March 7, 2013

Abstract

This paper addresses the real-time fixed-priority scheduling problem for battery-powered embedded systems whose energy storage unit is replenished by an environmental energy source. In this context, a scheduling policy should account for incoming energy, capacity of the battery and tasks cost of energy. Thus, classical fixed-priority schedulers are no longer suitable for this model. Some algorithms were proposed in precedent works to solve this problem, however, none of them has been proved optimal for concrete tasksets[1]. Based on this motivation, we propose FPC_{ASAP} a scheduling algorithm that handles both energy and timing constraints by predicting future possible failure. Such kind of algorithm is said to be clairvoyant. We expect this algorithm to be optimal.

1 Introduction and Related work

In this paper, we investigate a real-time system model for embedded systems that collect and store energy from their environment. Such systems are composed, in addition to classical embedded system components, by an energy harvester unit (e.g. a solar panel) and an energy storage unit (e.g. a battery). These harvesting embedded systems are more and more present in our lives: sensor networks in structures such as bridges that collects vibration energy, medical implants that collect energy from the human body, mobile or fix devices with

[1]a concrete taskset is a set of real-time tasks whose offsets are known off-line

solar panel or windmill etc. Despite of their energy supply particularity, some of these systems need to satisfy strict timing constraints. Therefore, it is important to consider both the time and the energy costs of tasks for the scheduling operations, since both the energy and the CPU times resources of the system have to be shared by the tasks.

The first work addressing the scheduling problem of energy harvesting systems was presented by Mossé in [1]. The problem was solved under a very restrictive task model: the frame-based model where all the tasks have exactly the same period and the same deadline. Later in [2], Moser et al. proposed an optimal algorithm called LSA (Lazy Scheduling algorithm) for periodic and aperiodic tasks. However, in their hypotheses, the CPU frequency can be changed to adjust the Worst Case Execution Time (WCET) of the tasks depending on their energy consumption. Thus, the results of this work rely on the assumption that tasks energy consumption is directly linked to their WCET. A recent work shows that this hypothesis is not suitable for embedded systems [3].

Later, a clairvoyant algorithm called $EDeg$ has been proposed in [4, 5]. The authors expect that this algorithm may be optimal but as far as we know this property has not been proved. The algorithm $EDeg$ relies on a generalizable meta policy: as long as the system can perform without energy failure, a standard policy such as EDF is used. Then, as soon as *future* energy failures is detected, the system is suspended as long as timing constraints are met or until the energy storage unit is full. To detect such future energy failure, the notion of slack time [6] was extended to the notion of slack energy. Another approach based on model checking for dealing with this problem has been proposed in [7].

Recently, in a paper under review process [8], we proposed PFP_{ASAP}, a fixed-priority algorithm that schedules jobs as soon as possible and we proved its optimality for non concrete tasksets.

The aim of this paper is to study the possibility of extending the optimality of PFP_{ASAP} algorithm to concrete tasks by introducing clairvoyance.

The remaining part of the paper is organized as follow. We present the model in Section 2. In Section 3, we explain why we need a clairvoyant algorithm, then, we propose a new algorithm that we call FPC_{ASAP}. Finally, we discuss future work and we conclude in Section 4.

2 Model

2.1 Taskset Model

We consider a concrete real-time taskset in a renewable energy environment defined by a set of n periodic and independent tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by its priority P_i, its worst case execution time C_i, its period T_i, its deadline D_i, its offset O_i and its worst case energy consumption E_i. The execution time C_i and the energy consumption E_i of a task are fully independent. A task τ_i releases an infinite number of jobs which are separated by T_i time units, its jth job is denoted $J_{i,j}$ and must execute during C_i time

units and consume E_i energy units. All the tasks consume energy linearly, i.e. a constant amount of energy for each execution time unit. Deadlines are constrained or implicit. The taskset is priority-ordered, task τ_n being the task with the lowest priority.

2.2 Target Application Description

We consider an embedded system connected to an energy harvesting device. An energy harvesting device is a system collecting energy from its environment (e.g. with a solar panel). The collected energy is stored in an energy storage unit with fixed capacity (e.g. chemical battery or capacitor). We suppose that the quantity of energy that arrives in the storage unit is a function of time which is either known or bounded.

The replenishment of the storage unit is performed continuously even during the execution of tasks, and the energy level of the battery fluctuates between two thresholds E_{min} and E_{max} where E_{max} is the maximum capacity of the storage unit and E_{min} is the minimum energy level that keeps the system running. The difference between these two thresholds is the part of the battery capacity dedicated to tasks execution, denoted C. We suppose that C is as large as needed to never reach E_{max} during replenishment periods. For the sake of clarity, we can consider without loss of generality that $E_{min} = 0$ and that $C = E_{max}$. The battery level at time t is denoted $E(t)$. We note $P_r(t)$ the replenishment function of the battery. Then, in order to simplify the problem, we assume $P_r(t)$ to be a linear function, i.e. $P_r(t) = P_r \times t$, P_r being a positive integer.

Below, we use P_r instead of $P_r(t)$ to denote the replenishment function. The replenishment process in energy harvesting systems is usually slower than the dissipation, for this reason we suppose that tasks consume more energy than the one which is replenished during executions, i.e. $\forall i$, $E_i \geq C_i \times P_r$.

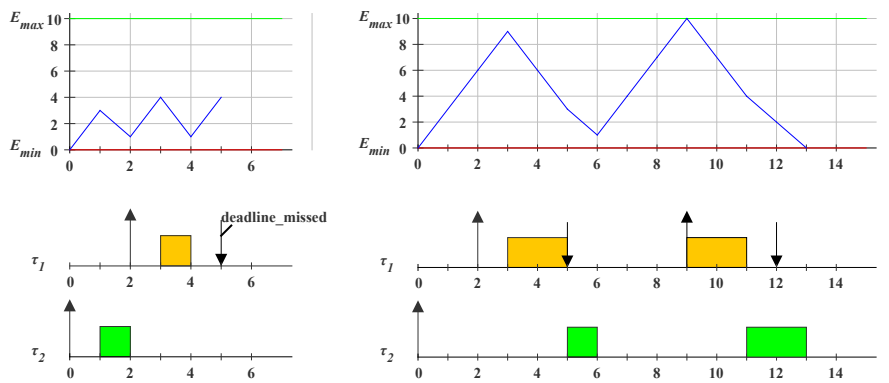
3 The FPC_{ASAP} Algorithm

3.1 The PFP_{ASAP} algorithm

The PFP_{ASAP} algorithm schedules tasks as soon as possible when there is energy available in the battery, and suspends execution to replenish the energy needed to execute one time unit. A proof of the optimality of this algorithm for non concrete tasksets is proposed in [8] and experiments showed that is has a low overhead. However, the algorithm is no longer optimal when we deal with concrete taskset. Figure 1 shows an example where PFP_{ASAP} fails to schedule a taskset while a valid schedule exists. We can see in Figure 1(b) that PFP_{ASAP} schedules task τ_2 before τ_1 which has no time to replenish enough energy. The PFP_{ASAP} algorithm decides to execute τ_2 without having any information about what will happen later. When τ_1 is activated, the energy available in the storage unit is not sufficient because τ_2 consumed it and the

-	O_i	C_i	E_i	T_i	D_i	P_i
τ_1	2	2	12	10	3	1
τ_2	0	3	15	15	15	2

(a) $E_{max} = 10$, $E_{min} = 0$ and $P_r = 3$



(b) PFP_{ASAP} schedule (c) A feasible schedule

Figure 1: PFP_{ASAP} is not optimal

available slack-time is not sufficient to replenish enough energy. However, if PFP_{ASAP} had checked whether the future jobs meet their constraints or not, deadline misses could be avoided, Figure 1(c) illustrates such a decision. This leads us to say that PFP_{ASAP} fails to schedule some feasible tasksets because it does not measure the impact of its decisions on future jobs.

3.2 As Soon As Possible Clairvoyant Fixed-Priority Algorithm

We aim to find a scheduling algorithm that can be optimal for concrete tasksets. PFP_{ASAP} is not optimal but it can be enhanced to extend its optimality. The algorithm has to be clairvoyant in order to avoid potential future failure or deadline misses.

We propose to study a new clairvoyant algorithm that we call *As Soon As Possible Clairvoyant Fixed-Priority Algorithm* (FPC_{ASAP}). An algorithm is said to be clairvoyant when it takes scheduling decisions according to the future state of the system, i.e. future processor and energy load, future battery level, and the future amount of incoming energy to the storage unit. The clairvoyance must be limited to a finite interval of time that we call *Clairvoyance window*.

The FPC_{ASAP} algorithm inherits the behavior of PFP_{ASAP} and add clairvoyance capabilities. It schedules jobs as soon as possible whenever the two following conditions are met:

- there is enough energy available in the storage unit to execute at least one

Algorithm 1 FPC_{ASAP} Algorithm

1: $t \leftarrow 0$
2: **loop**
3: $A \leftarrow$ set of active jobs at time t
4: **if** $A \neq \emptyset$ **then**
5: $J_{i,j} \leftarrow$ the highest priority job of A
6: $d_i(t) \leftarrow$ the next absolute deadline of $J_{i,j}$
7: **if** $ResponseTime_{PFP_{ASAP}}(t+1, J_{i,j}, E(t+1)) > d_i(t)$ **then**
8: execute $J_{i,j}$ for one time unit at time t
9: **else**
10: **if** $Clairvoyance_{PFP_{ASAP}}(t, J_{i,j}, d_i(t), E(t))$ **then**
11: execute $J_{i,j}$ for one time unit at time t
12: **else**
13: suspend the system for one time unit
14: **end if**
15: **end if**
16: **end if**
17: $t \leftarrow t+1$
18: **end loop**

time unit,

- the execution of the current job does not lead to a deadline miss of jobs of higher priority levels which are requested during clairvoyance window (see below).

If these conditions are not satisfied, the algorithm suspends all executions for one time unit.

Algorithm 1 shows how FPC_{ASAP} takes decisions at time t when a job of priority level i is ready to run. The FPC_{ASAP} algorithm checks first if the execution of jobs of priority level higher or equal to t according to PFP_{ASAP} rules meet theirs deadlines. Algorithm 1 first checks if it is possible to delay the current job by comparing its response time at time $t+1$ with its deadline (line 7), then, it repeats the process for higher priority jobs. This prevent from delaying the current job uselessly, because in the case where it is impossible to delay, if a deadline miss occurs in a higher priority level in the clairvoyance window, the failure cannot be avoided and the system is not schedulable with FPC_{ASAP}.

The first question is: which jobs should be considered in the clairvoyance computation ? In other words, which job can suffer from the energy consumed by the current job of priority level i at time t ?

In fixed-priority context, when a higher priority task is requested, it cannot be delayed by lower priory ones. However, since the energy is a common resource, lower priority tasks can consume the energy needed for higher priority ones and can so delay them. The aim of clairvoyance is to delay lower priority

tasks when such a problem is detected. Knowing that a job cannot be delayed more than its deadline, the jobs which are affected by the execution or the delay of $J_{i,j}$ at time t, are the ones of priority levels higher or equal to i which are requested between time t and the absolute deadline of $J_{i,j}$. This defines our clairvoyance window. If a deadline miss occurs after the deadline of $J_{i,j}$, the system is not feasible and clairvoyance cannot avoid that because whatever happens, $J_{i,j}$ must finish executing before its deadline.

The second question is: how can we check if future jobs meet their deadlines ?

A simple way is to compute their response times and compare their termination dates with their absolute deadlines. Since FPC_{ASAP} uses PFP_{ASAP} algorithm for clairvoyance, we have to compute response times according to PFP_{ASAP} rules. Algorithm 2 shows how to compute the response time of a job $J_{i,j}$ at time t with the following notations:

- $x_i(t)$: next activation of task τ_i,
- $we_i(t)$: energy demand of priority level i at time t,
- $wp_i(t)$: processor demand of priority level i at time t,
- $w_i(t)$: time demand of priority level i at time t,
- $c_i(t)$: is the remaining time cost of job $J_{i,j}$ at time t,
- $e_i(t)$: is the remaining energy cost of job $J_{i,j}$ at time t,
- $d_i(t)$: is the absolute deadline of job $J_{i,j}$ at time t.

Algorithm 2 $ResponseTime_{PFP_{ASAP}}$

INPUT : t, $J_{i,j}$, $E(t)$
$w'_i(t) \leftarrow t + \epsilon$
repeat
  $w_i(t) \leftarrow w'_i(t)$
  $we_i(t) = \sum_{l \leq i} \left(e_l(t) + \left\lceil \frac{w_i(t) - x_l(t)}{T_l} \right\rceil \times E_l \right) - E(t)$
  $wp_i(t) = \sum_{l \leq i} \left(c_l(t) + \left\lceil \frac{w_i(t) - x_l(t)}{T_l} \right\rceil \times C_l \right)$
  $w'_i(t) = t + \max \left(\left\lceil \frac{we_i(t)}{P_r} \right\rceil, wp_i(t) \right)$
  if $w'_i(t) > d_i(t)$ **then**
    return $w'_i(t)$
  end if
until $w_i(t) = w'_i(t)$
return $w_i(t)$

Algorithm 2 behaves like the classical fixed-priority response time algorithm and include tasks energy cost in calculations. It computes analytically the response time of a job $J_{i,j}$ of priority level i by computing progressively the time demand $w'_i(t)$ which is the maximum time needed to satisfy both of the processor demand $wp_i(t)$ and the energy demand $we_i(t)$. The energy demand $we_i(t)$ is derived from the notion of processor demand. It represents the sum of the energy cost of all the jobs of priority equal or higher than i requested during the time interval $[t, w_i(t)[$. Then, the initial battery level $E(t)$ is removed to fit the exact amount of energy to be replenished (Equation is given in line 5 of Algorithm 2).

We deal with the maximum of $we_i(t)$ and $wp_i(t)$ to get the real response time which includes all higher priority interferences and the replenishment periods necessary for a PFP_{ASAP} schedule. Algorithm 2 supposes that the current battery level $E(t)$ and the remaining value of both energy cost $e_i(t)$ and processor cost $c_i(t)$ are known at time t.

3.3 Clairvoyance computation

At time t, when the FPC_{ASAP} algorithm has to decide whether to execute or not the current job $J_{i,j}$, it uses clairvoyance to predict potential future failure. The clairvoyance consists of computing the response time of all the jobs which are requested during the clairvoyance window, i.e. $[t, d_i(t)[$ and checking if they meet their deadlines. Algorithm 3 shows how to do that.

Algorithm 3 $Clairvoyance_{PFP_{ASAP}}$ Algorithm

1: **INPUT** : $t, J_{i,j}, d_i(t), E(t)$
2: $R \leftarrow$ set of jobs $J_{k,j}$ of priority level higher than i which are requested during interval $[t, d_i(t)[$
3: **for all** $J_{k,j} \in R$ **do**
4: $\quad r_{k,j} \leftarrow$ the activation date of $J_{k,j}$
5: $\quad$ **if** $ResponseTime(J_{k,j}, r_{k,j}, E(r_{k,j})) > d_k(r_{k,j})$ **then**
6: $\quad\quad$ **return** $False$
7: $\quad$ **end if**
8: **end for**
9: **return** $True$

Algorithm 3 supposes that the remaining cost of tasks and the battery level at each job request are known. This assumption simplifies the clairvoyance computation, however, computing these values need heavy calculations and may complexify the clairvoyance algorithm and consequently the overhead of the FPC_{ASAP} Algorithm. This concern remains an open problem.

3.4 Complexity

The main goal of this work is to provide an optimal algorithm. The complexity of this algorithm is crucial but in this stage of research we first focus on optimality,

then, we will study the possibility of reducing the complexity. The computations of clairvoyance and response time algorithms are thus the major keys to the operations performed by FPC_{ASAP} algorithm. The response time of a job at a given instant is given by Algorithm 2 with complexity $O(K.n)$ where n is the number of tasks, and K the number of iterations. In the worst case, K corresponds to the number of interfering jobs and thus is bounded by R/p, where R and p are respectively the longest deadline and the shortest period of tasks. The clairvoyance computation of a periodic taskset at a given time instant can be obtained on-line by computing the PFP_{ASAP} schedule. This is performed with complexity $O(K^2.n)$ by computing the response time of each job activated during the clairvoyance window. The number of these jobs coincides with the number of interfering jobs in the worst case, i.e. K. Therefore, the complexity of FPC_{ASAP} in the worst case is $O(K^2.n)$.

4 Conclusion and Future work

The aim of this work was to find an optimal algorithm to schedule fixed-priority tasksets in energy-harvesting environment. The PFP_{ASAP} algorithm is optimal but only for non concrete taskset. In this paper we presented FPC_{ASAP} a new algorithm that inherits the behavior of PFP_{ASAP} and add failure predictability capabilities. We explained why we need clairvoyance and we gave a sufficient clairvoyance window length that allow us to verify whether or not there will be future failures caused by early executions.

To continue this work, we plan to explore the following points.

- Optimality: the most important future work is to prove that FPC_{ASAP} is optimal. This algorithm is expected to keep the optimality of PFP_{ASAP} for non concrete

tasksets because it behaves the same when there is no future deadline misses and extends it to concrete ones by delaying lower priority jobs as long as needed when a future problem is detected.

- Battery size: in this paper we supposed that the battery is as large as needed to never reach E_{max}. First, we have to find the minimal capacity satisfying this condition. Then, we will study the impact of E_{max} on our algorithm when the battery size is set by the designer.
- Clairvoyance: we will try to find a clairvoyance algorithm that computes the exact response times of jobs with a reasonable complexity.
- Preemptions: we will study the possibility of optimizing the number of preemptions and we will evaluate the performance of a non preemptive version of FPC_{ASAP}.
- Priority assignment: we will study the effect of priority assignment policies on FPC_{ASAP} behavior and clairvoyance complexity, for example reversed Rate Monotonic policy (RM^{-1}) can reduce the number of higher priority interferences which can enhance sensitively the complexity. We will try also to provide a sufficient and necessary feasibility condition that can be used to build an optimal priority assignment (OPA) based on Audsley's algorithm [9] and Davis' criteria [10].
- Assumptions: finally, we will be interested in measuring the effect of the assumptions we set on both replenishment and task consumption functions, indeed, we will try to find the worst case of both consumption and replenishment models.

References

[1] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS)*, 2001.

[2] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling with regenerative energy," in *Proceedings of 18th Euromicro Conference on Real-Time Systems*, 2006.

[3] R. Jayaseelan and T. Mitra, "Estimating the Worst-Case Energy Consumption of Embedded Software," in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2006.

[4] H. EL Ghor, M. Chetto, and R. H. Chehade, "A real-time scheduling framework for embedded systems with environmental energy harvesting," *Computers and Electrical Engineering*, vol. 37, pp. 498–510, July 2011.

[5] M. Chetto, D. Masson, and S. Midonnet, "Fixed Priority Scheduling Strategies for Ambient Energy-Harvesting Embedded systems," in *Proceedings of IEEE/ACM International Conference on Green Computing and Communications*, 2011.

[6] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems," in *Proceedings of the 13th IEEE Real-Time Systems Symposium*, 1992.

[7] Y. Abdeddaïm and D. Masson, "Real-time scheduling of energy harvesting embedded systems with timed automata," in *RTCSA*, 2012.

[8] Y. Abdeddaïm, Y. Chandarli, and D. Masson, "The Optimality of PFPasap Algorithm for Fixed-Priority Energy-Harvesting Real-Time Systems," Report, Feb. 2013.

[9] N. Audsley, *Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times*, ser. Technical report. University of York, Department of Computer Science, 1991.

[10] R. I. Davis and A. Burns, "Priority assignment for global fixed priority preemptive scheduling in multiprocessor real-time systems," in *Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*, 2009.