Fixed Priority Scheduling Strategies for Ambient Energy-Harvesting Embedded systems

Maryline Chetto, Damien Masson, Serge Midonnet

Fixed Priority Scheduling Strategies for Ambient Energy-Harvesting Embedded systems

Maryline Chetto
Université de Nantes
IRCCyN, UMR CNRS 6597
IUT de Nantes
2 ave du Professeur Jean Rouxel
44475 Carquefou
maryline.chetto@univ-nantes.fr

Damien Masson
Université Paris-Est
LIGM, UMR CNRS 8049
ESIEE Paris
2 bld Blaise Pascal – BP 99
93162 Noisy-le-Grand CEDEX
d.masson@esiee.fr

Serge Midonnet
Université Paris-Est
LIGM, UMR CNRS 8049
Université Paris-Est Marne-la-Vallée
5 bld Descartes
77454 Marne-la-Vallée CEDEX 2
serge.midonnet@univ-paris-est.fr

***Abstract*—The new generation of embedded systems will have the capability to harvest energy from the environment. The electrical energy which is available to power these devices changes over time and is limited by the size of the energy storage unit such as battery or capacitor and the size of the harvester such as a solar panel. In order to cope with this limitation, the system has to dynamically decide when to be active and when to sleep in order to provide the best quality of service without wasting the harvested energy. In this paper, we study this problem for a uniprocessor architecture where periodic tasks have to execute with deadline constraints according to a preemptive fixed priority rule. We evaluate and compare several scheduling approaches by means of simulation.**

I. Introduction

Up to now, Wireless Sensor Networks (WSN) have used batteries as portable energy source. However, systems continue to become smaller. As less energy is available on board, this leads to a short run-time for the devices. Unfortunately, there are emerging embedded applications where sensors are required to operate for very long durations after they are deployed. Extended life of these electronic devices is of particular importance when they have limited accessibility. Ambient energy harvesting is also known as energy scavenging. It is defined as the process where energy is drawn from the environment and then converted and stored for use in electronics applications. We generally use this terminology for low power and small autonomous devices such as wireless sensor networks, and portable electronic equipments. Consequently, ambient energy harvesting has generated a lot of interest for research. It will permit that autonomous embedded systems be powered perpetually. When compared with energy stored in classical storage units such as batteries, the environment represents an infinite source of available energy. Furthermore, an important advantage is eliminating replacement and periodic recharging of batteries that constitute a major part of service and maintenance. Many environmental sources can be exploited, including thermal, optical, mechanical, fluidic, etc. The selection of energy sources must be considered according to the application characteristics. Self powered sensors for medical implants for health monitoring and embedded sensors in structures such as bridges, buildings for remote condition monitoring are typical new generation embedded systems called Energy-Harvesting (EH) embedded systems. A classical EH system is composed of three parts. The harvester scavenges the energy from ambient surroundings and converts it to usable electrical power. The battery or/and capacitor stores the energy. And the computing system further uses this energy to run software. In EH systems, we have to make the best use of the available energy which is highly dependent on the environment. The energy consumption of the system should be adjusted to maximize the performance instead of minimizing the energy consumption as in classical battery powered systems. A new role of the operating system is to manage correctly the activity of the processing unit such that, at every time there is sufficient energy in the storage unit to satisfy all the constraints. Most environmental energy sources do not deliver a constant power over time. The energy generated may arrive in bursts and has to be stored so that the device can still be operated at a later moment. The main challenge for an EH system lies in the optimization of its performance while respecting the time-varying amount of energy without wasting or exhausting the energy stored in the battery or capacitor. We then say that the system operates in an energy neutral mode by consuming only as much energy as harvested [1]. In most applications, the EH system is embedded and has real-time requirements expressed in terms of deadlines attached to the execution of programs, generally called tasks. Then, the performance of such system is measured by the deadline miss rate and heavily depends upon the stored energy and the energy harvested from the environment. Unfortunately, the scavenging power is time-varying and thus very unstable. Therefore, the accurate modeling for energy source plays a key role in designing a good policy to schedule the tasks and reduce the deadline miss rate. So, in that paper, we will describe how to dynamically manage the activity of a single processor system in charge of executing real-time tasks which are periodic and have to be scheduled according to a fixed priority driven policy. We must be able to answer questions like the following: When should the system use energy? When should it be idle and recharge the energy storage? Specifically, we explore different scheduling heuristics and compare their performance. This paper focuses

on the same problem studied in [2] but considers variants of the Fixed Priority (FP) scheduler instead of Earliest Deadline First (EDF) scheduler. Simulations illuminate the merits of our heuristics.

The remainder of the paper is organized as follows: We first present the necessary background relative to real-time scheduling with energy harvesting constraints in the remainder of this Section. The model is described in Section II. We give fundamental definitions and introduce scheduling heuristics in Section III. A method for computing the slack time in a fixed priority environment is presented in Section IV. Section V is concerned with an experimental study for comparing the scheduling heuristics with different criteria. Finally, we conclude the paper in Section VI.

Related Work

Only in the past decade, researchers started to address power and scheduling issues with the objective of either minimizing power usage under timing constraints or maximizing the system performance under the energy constraints. Nevertheless, they did not consider the rechargeability of the batteries. For example, the well known Earliest Deadline First scheduler has been extended to variable-voltage processors. The idea is to save power by slowing down the processor just enough to meet the deadlines of the tasks. 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 the work by Allavena et al. in [3], power scavenged by the energy source is constant and all tasks consume energy at a constant rate. Later in [4], Moser at al. propose LSA (Lazy Scheduling Algorithm) to optimally schedule tasks with deadlines, periodic or not. In that work, the total energy consumption of every task is directly connected to its execution time through the constant power of the processing device. But in a real application, instantaneous power consumed by tasks may vary along time depending on circuitry and devices required by the tasks. Very recently, in [2], we relax the restrictive hypothesis that links together energy requirement and execution time of tasks. We present an on-line scheduling scheme, EDeg (Earliest Deadline with energy guarantee) that is a variant of EDF which relies on two fundamental concepts, namely slack time and slack energy. EDeg dynamically applies a slack-based method for making the processor inactive as long as possible without injuring deadlines feasibility. Performance evaluations have been performed and show the efficiency of this scheduler. However, EDeg is a clairvoyant scheduler that needs knowledge of future task arrival times to dynamically build an optimal schedule, which seriously limits its scope. To the best of our knowledge, all works for scheduling energy-harvesting systems so far consider dynamic priority scheduling based on the famous Earliest Deadline First rule. In this paper, our contribution will be to discuss the same question for fixed priority systems which are in the majority among current real-time embedded applications.

II. Models and assumptions

This paper explores an embedded system which is equipped with an energy harvesting device. Our system model then includes the application model, the energy source model and the energy storage model.

a) Application model: We consider here a set of independent periodic tasks that can be denoted as follows: $\Phi = \{\tau_i, i = 1, \ldots, n\}$. 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, $k = 1, 2, ...$ 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 $0 < C_i \leq D_i \leq T_i$ for each $1 \leq i \leq n$. The tasks are scheduled on a single processor system based on a fixed priority driven scheduler.

b) Energy source model: 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 $P_r(t)$). For sensor nodes deployed in a certain environment, a predictor is used to predict the amount of harvested energy in the future [5]. $P_r(t)$ is then the instantaneous charging rate that incorporates all losses caused by power conversion and charging process.

c) Energy storage model: Our system uses an energy storage unit that has a nominal capacity, namely C, corresponding to a maximum energy (expressed in Joules or Watts-hour). The stored energy, denoted as $E_s(t)$ at a given time t, has to remain between two boundaries E_{min} and E_{max}. Consequently, $E = E_{max} - E_{min}$. $E_s(t)$ can be measured with reasonable accuracy, used at any time later with no leak over time. We assume that energy production times can overlap with the consumption times.

III. Fixed priority Scheduling

Schedulability analysis of periodic task sets can easily be performed both under fixed and dynamic priority assignments. In particular, a lot of work has been done for the Rate Monotonic (RM) and the Earliest Deadline First (EDF) algorithms [6]. RM scheduling is the optimal fixed-priority (or static-priority) scheduling policy for periodic tasks for the case where task deadlines are coincident with the end of a task's period. That means that if any other fixed-priority scheduling policy can meet deadlines, so can RM. In [6], the authors also derived a sufficient condition for the schedulability of task sets that use a RM priority assignment. Leung and Whitehead in [7] later showed the optimality of a Deadline Monotonic (DM) priority assignment for the case where periodic tasks have deadlines that are at or before the end of their period. However, these well known analysis methods do not take into account energy constraints.

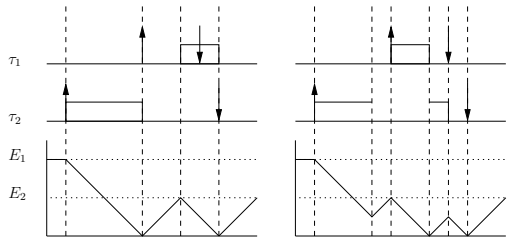


Fig. 1. Non optimality of strategies that do not anticipate energy shortage. In this example, τ_1 has a priority higher than τ_2. We can see that there is no solution for producing a valid schedule τ_1 if τ_2 is not paused before it consumes all the energy. However the task set is feasible if τ_2 is paused before is has consumed all the remaining energy. Note that this result also applies to dynamic priority assignments rules such as EDF.

A. Feasible Systems

In this paper, we focus on preemptive Fixed Priority (FP) scheduling strategies such as RM or DM that are applied at run time in an EH system as one described in the previous section. We are interested with online algorithms i.e. schedulers that must make their decisions at run-time. In what follows, we will use the following definitions: A schedule for a set of periodic tasks Φ is *valid* if the deadlines of all tasks of Φ are met, starting with an energy storage unit fully charged. A periodic task set Φ is *timely-feasible* if there exists a valid schedule for Φ without considering its energy constraints. A periodic task set Φ is *feasible* if there exists a valid schedule for Φ. A scheduling algorithm is *optimal* if it finds a valid schedule whenever one exists.

B. Scheduling Heuristics

We propose here to define and evaluate four scheduling heuristics, all using the same FP rule. The first one is pretty straight forward and natural: it consists to make the system idle for a fixed arbitrary amount of time whenever the energy storage unit becomes empty i.e. at time t where E(t) reaches E_{min}. Our objective is to demonstrate that such a situation may be avoided or at least be delayed. The second heuristic comes from the idea that the current energy level in the battery has to be considered dynamically. In one hand it is not necessary to let the system paused for a long time if a short interval is sufficient to refill a significant fraction of the battery capacity. In the other hand, if the energy level has not sufficiently increased, there is a high probability to quickly reach again the empty state ($E(t) = E_{min}$). This intuition is also enforced by the fact that realistic harvester/battery models do not permit a time linear refill strategy. The third heuristic considers that the longer the battery is recharging, the better it is. The problem then amounts to compute the length of the longest interval during which the system can be paused while guaranteeing no deadline miss. That necessarily implies for the scheduler to dynamically determine the slack time at run time. Explanations relative to that computation will be given in Section IV. However, in order to avoid wasting energy if the storage unit fully replenishes during the paused period, we put the processor in idle state only if there is slack time and the energy level is not maximum. We will show that none of these online scheduling heuristics is optimal. In other words, we can always find at least one feasible task set that is not feasibly scheduled by the heuristic. Indeed, we can construct a scenario where all these heuristics fail to produce a valid schedule while such a schedule exists through adequate idling periods. This scenario is represented in Figure 1. For that reason, we propose an additional heuristic using a threshold value for the energy level: whenever the threshold value is reached, we let the processor idle. However, with an arbitrary value for this threshold, system performances cannot be significantly optimized, even in the average case.

IV. Slack time in fixed priority Systems

We define the slack-time of a task set Φ at time t as the maximum amount of time the system can be suspended from time t without missing any deadline. It is denoted $S(t)$. The slack-time of a task τ_i at time t, denoted $S_i(t)$, corresponds to the maximum amount of time τ_i can be delayed from time t without missing its deadline. This value represents the unused CPU time units at priorities higher than or equal to i. We have $S(t) = \min_{\forall i} S_i(t)$. These notions were introduced in [8] in order to address the issue of scheduling mixed traffic composed of hard real-time periodic tasks and soft real-time aperiodic ones. The slack-time computation then officiates as an on-line admission control test for aperiodic traffic. Algorithms to dynamically compute the exact value and upper bounds in fixed priority driven systems are given in [9], [10], [11].

A. Processing exact slack time $S_i(t)$

To calculate $S_i(t)$, we consider the system between t and the next deadline of the i-level priority task (τ_i). During this interval the system can be viewed as a succession of i-level busy periods (periods where the processor is servicing priorities higher or equal to i) and i-level idle periods (periods where the processor is idle or periods where processor is servicing priorities lower than i). The slack time value $S_i(t)$ is then the sum of the i-level idle period lengths. To process the exact slack value we use the two equations to compute the end of a busy period starting at instant t and to compute the length of an idle period starting at time t. The process steps can be summarized as follows: 1) initializing the slack value $S_i(t)$ at time 0, 2) processing $w_i(t)$, the length of the busy period starting at time t, 3) processing $vi(t, w_i(t))$, the length of the idle period starting at time $t + w_i(t)$, 4) incrementing the slack value by the length of the idle period, 5) restarting the process at step 2 or stopping it if the deadline of the task τ_i is reached. This algorithm could not be processed at any time instant because of its computational complexity. So, in order to simplify its dynamic computation, the slack-time value is obtained at time t' from the slack-time value computed at time t. Two general cases have to be studied: none of the hard periodic tasks ends its execution in $[t, t')$ or one periodic hard real-time task (τ_i) ends its execution at time $t'' \in [t, t')$. In the first cas, then there are two possibilities

for the processor activity between t and t': the processor is idle or it is executing hard periodic tasks. In the first case, the slack is reduced by $(t'-t)$ for all priorities. However, if the processor is executing the hard real-time task τ_i, then the system is idle for higher priorities ($k < i$), and the slack is reduced by $(t'-t)$ only for these priorities. In the second case, then all *i-level* idle times present in $[t, d_i(t))$ will be present in $[t, d_i(t) + T_i) = [t, d_i(t''))$, which is the new interval to consider for the *i-level* slack times computation. Therefore, the τ_i termination can only increase $S_i(t)$ but never decreases it. Consequently, $S_i(t)$ has to be recomputed each time τ_i ends a periodic activation. So, assuming that there is a time t where the $S_i(t)$ was up to date for all tasks, the algorithm to compute $S(t'')$ is :

1) if none of the periodic hard real-time tasks ends in $[t, t')$
   a) if the processor is idle or executing soft aperiodic requests
   $$\forall j : S_j(t') = S_j(t) - (t' - t) \quad (1)$$
   b) if the processor is executing hard periodic task τ_i
   $$\forall j < i : S_j(t') = S_j(t) - (t' - t) \quad (2)$$
2) if hard real-time task τ_i ends at time $t'' \in [t, t')$, $S_i(t'')$ has to be computed using the recursive analysis described at the beginning of this section.

V. Experimental results

We developed a discrete event simulator to evaluate the effectiveness of the scheduling heuristics previously introduced, all using a simple and easy to implement fixed priority rule. For all schedulers, we consider the system as failed as soon as one deadline is missed. The simulator is available under the GNU Public License at http://dajam.fr/Softwares/, with sufficient material for reproducing the experiments.

A. Simulation restrictions

In our simulations, we assume that $P_r(t)$ is constant along time. Moreover, we assume that all tasks linearly consume their energy budget over time, and their consumption rate, given by E_i/C_i, is always greater than $P_r(t)$ (every task execution leads to discharge the battery).

B. Formal definition of Scheduling strategies

Five *Energy Harvesting in Fixed Priority* (EHFP) scheduling policies have been evaluated:

1) $EHFP_1$*:* All tasks are processed as soon as possible according to the Fixed Priority (FP) rule until the storage unit is empty or there are no more task ready to be executed. If the storage unit is empty and there is at least one ready task, the processor is put into sleep mode during x units of time where x is an input of the scheduler. During that period, the energy storage unit replenishes. Default value for x is 1.

2) $EHFP_2$*:* All tasks execute as soon as possible according to FP. When the storage unit becomes empty, the processor is put into sleep mode until the energy level reaches a threshold value, namely E_{th}, given as an input of the scheduler.

3) $EHFP_3$*:* All tasks execute as soon as possible according to FP. When the storage unit becomes empty, the processor is put into sleep mode as long as the slack time remains positive.

4) $EHFP_4$*:* All tasks execute as soon as possible according to FP. When the storage unit becomes empty, the processor is put into sleep mode as long as the slack time remains positive and the energy level remains less than E_{max}.

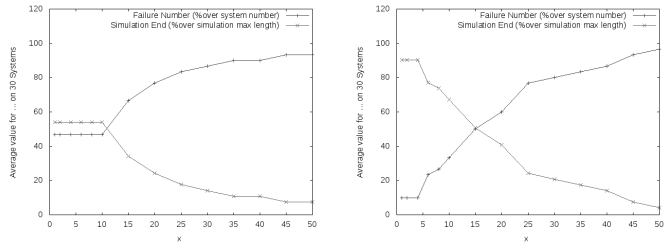
5) $EHFP_5$*:* There are two threshold parameters, namely E_{thmin} and E_{thmax}. All tasks execute as soon as possible according to FP. When the energy level reaches E_{thmin}, the system is put into sleep mode as long as the slack time remains positive and the energy level remains less than E_{thmax}.

C. Simulation methodology

1) Task sets generation: Groups of periodic task sets are generated with utilization factors respectively equal to 30, 50, 70 and 90%. Results presented in this section are averages over groups of ten task sets. Each task set contains six real-time periodic tasks. Periods are randomly generated with an exponential distribution in the range [40-2560] time units. Costs are randomly generated with an uniform distribution in the range [1-period]. In order to test task sets with deadlines less than periods, we also randomly generate deadlines with an exponential distribution in the range [cost-period]. Nevertheless, these results are not presented here since they do not bring any additional conclusions. Priorities are assigned to tasks according the Deadline Monotonic policy and solely timely-feasible task sets are evaluated.

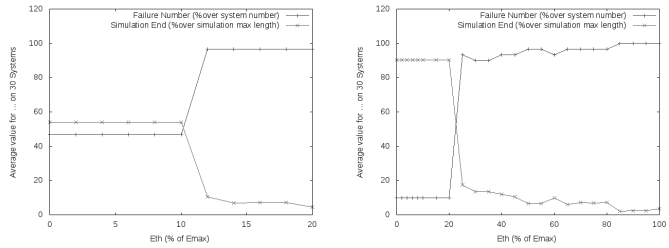
2) Energy Parameters Generation: Energy consumption of tasks (WCEC) are randomly generated, but constrained by E_{max} and P_r values. Simulations have been performed with different values for E_{max} and P_r. Energy consumption per time unit of each task, E_i/C_i, is generated with an uniform distribution in the range $(P_r, E_{max}/3]$. E_i is then obtained by multiplying this value by C_i.

3) Simulations Description: Every simulation starts with a fully charged energy unit i.e. $E(0) = E_{max}$, in the worst case timing scenario that corresponds to the synchronous activation of all periodic tasks at time $t = 0$. Every simulation terminates when either a deadline is missed or time $t = 100000$ is reached. The different measures are: 1) the number of feasible task sets (on the interval [0-100000]), 2) the average finishing date of the schedule corresponding to the first deadline violation whenever one exists, 3) the average time spent with maximum energy level in the storage i.e. $E(t) = E_{max}$, 4) the average time spent with minimum energy level in the storage i.e. $E(t) = E_{min}$, 5) the average duration of busy-periods (corresponding to discharging phases of the energy storage) and 6) the average duration of idle-periods (corresponding to recharging phases of the energy storage or energy storage fully charged) Finally, we compare these measures for different scenarios. We evaluate the impact of increasing/decreasing E_{max} i.e. the capacity of the energy storage unit and P_r. For policies $EHFP_1$ and $EHFP_2$, we also evaluate the impact of varying the input parameter of theses heuristics.



(a) for $E_{max} = 100$ and $P_r = 10$ (b) for $E_{max} = 200$ and $P_r = 40$

Fig. 2. Effects of parameter x on $EHFP_1$



(a) $E_{max} = 100$ and $P_r = 10$ (b) $E_{max} = 200$ and $P_r = 40$

Fig. 3. Effects of parameter E_{th} on $EHFP_2$

4) Effects of x parameter on $EHFP_1$: Simulations are performed first with $E_{max} = 100$ and $P_r = 10$, and second with $E_{max} = 200$ and $P_r = 40$. Results are respectively presented on Figure 2(a) and Figure 2(b). In both cases, we observe a correlation between the number of failures (in other terms the ratio of non-feasibly scheduled task sets) and the length of the simulation. Moreover, we observe that the performances are constant until x reaches a value (10 for 2(a) and up to 5 for 2(b)). There after this threshold, the pause time is too big and a higher number of deadlines are missed.

Conclusion: This set of simulations enables us to conclude that the length of the pause intervals has an impact on system performance. A value in a reasonable small range permits to increase the mean system life time, but there is no optimal value.

5) Effects of parameter E_{th} on $EHFP_2$: Simulations are performed first with $E_{max} = 100$ and $P_r = 10$, and second with $E_{max} = 200$ and $P_r = 40$. Results are respectively presented on Figure 3(a) and Figure 3(b). First, we observe that in the worst and best cases, $EHFP_2$ behaves similarly to $EHFP_1$. As for the x parameter of $EHFP_1$, the threshold parameter attached to $EHFP_2$ gives the same ratio of performances until a critical value (10% for 3(a) and 20% for 3(b)). Identically to $EHFP_1$ we cannot draw an optimal value for that parameter.

Conclusion: We demonstrate through these simulations that basing the paused time duration on a criteria that depends on the battery size permits to enlarge the range of values that offer the best average performances. However, there is no optimal value for this threshold in the general case, and $EHFP_2$ does not provide a better performance than $EHFP_1$, in terms of deadline missings.

D. Comparison with fixed E_{max} and P_r

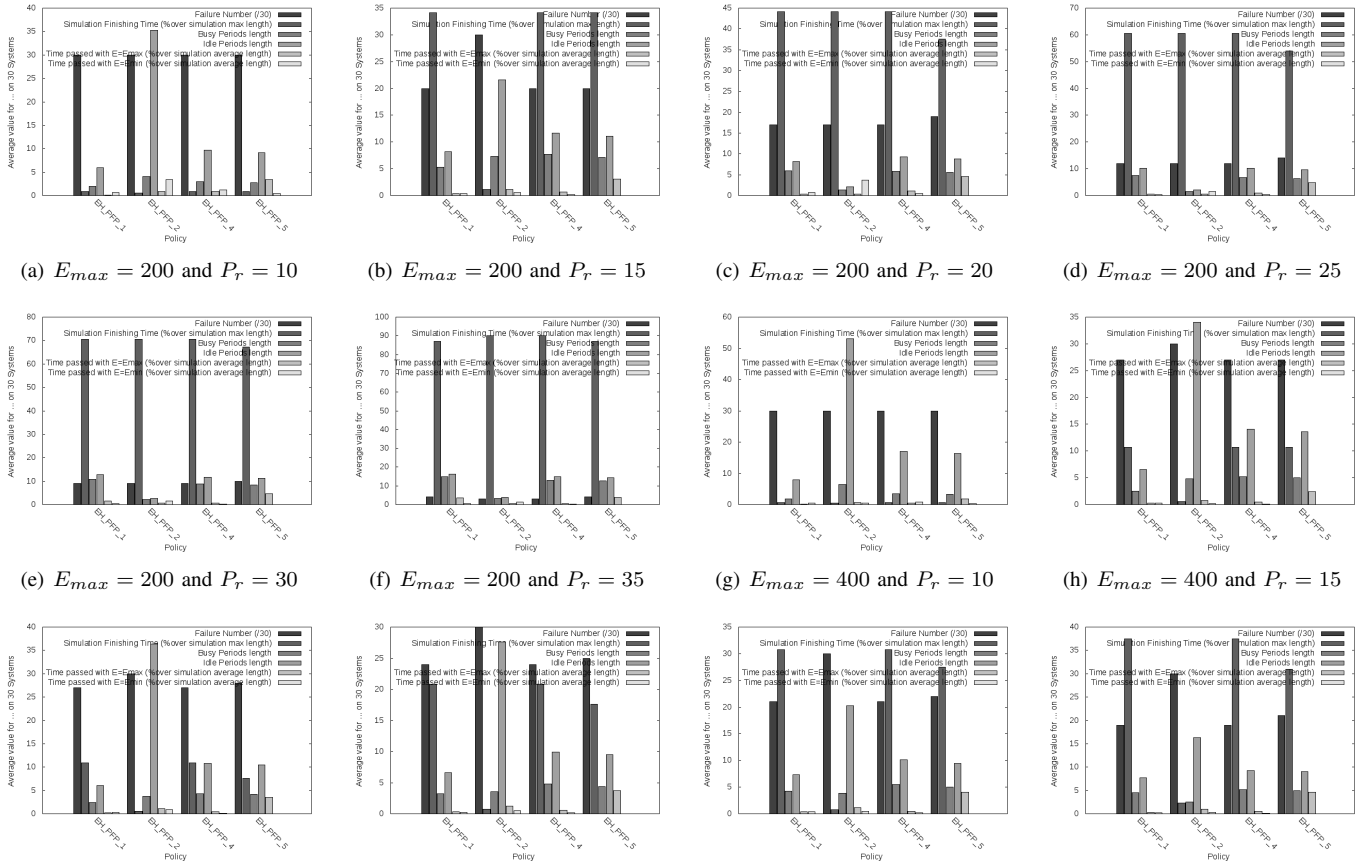
Simulations are now performed by making vary E_{max} and P_r. We pay attention to compare the different heuristics when the energy constraints vary. We set $x = 6$ for $EHFP_1$ and $E_{th} = 10\%$ for $EHFP_2$ and $EHFP_5$. Simulation results are presented on Figure 4. Complete and detailed results can be consulted on-line at http://dajam.fr/greenCom2011. Due to its very bad performance, policy $EHFP_3$ is eliminated from this section. That demonstrates that a policy, for providing acceptable performance, has to take into account the maximal upper bound for the battery charge. In consequence $EHFP_3$ is not competitive.

1) Performance evaluation in terms of feasibility: One crucial criteria for measuring the performance of any real-time scheduling strategy is the ratio of feasible task sets. Experiments empirically demonstrate that, using this criteria, $EHFP_4$ outperforms the other policies in most cases. However, even if this ratio is never less than for other policies, task sets which fail may not be the same. Moreover the gain is generally quite small. It is highly probable that no online scheduler can be optimal. In other terms, only a clairvoyant algorithm (i.e. an algorithm which knows in advance all the future) can produce a valid schedule whenever one exists.

2) Performance evaluation in terms of busy/idle period duration: Every schedule can be characterized by the average number of idle time periods and so the average duration of idle time periods. This criteria is of particular importance in systems using a DPM (Dynamic Power Management) mechanism. Many architectures provide the equivalent of a halt instruction that reduces CPU power during idle periods. However, there are usually significant latencies and overheads for entering and exiting these states. Consequently, the number of state switchings will influence the resulting efficiency of the system. Higher will be the average duration of the idle time periods, lower will be the energy and time overheads incurred by implementation of a DPM mechanism. Moreover, this number will have an impact on the lifetime of the energy storage unit, in general a battery or capacitor. Charging a battery is not a linear process and longer it is paused, the more energy it harvests. As for the previous performance criteria, we cannot conclude on the existence of an optimal strategy regarding the average number of state switching. No policy appears as the best one amongst the heuristics under evaluation. Nevertheless, we observe that, in comparison with other heuristics and in most cases, $EHFP_4$ maximizes the average idle/busy period duration. Only in very few cases, $EHFP_4$ is outperformed by an other heuristic with a non significant gap.

VI. Conclusion

Ambient energy harvesting can provide an extended lifespan and support to conventional electronics systems including embedded systems with real-time constraints. This paper has explored how to optimize the performance of these systems



(a) $E_{max} = 200$ and $P_r = 10$ (b) $E_{max} = 200$ and $P_r = 15$ (c) $E_{max} = 200$ and $P_r = 20$ (d) $E_{max} = 200$ and $P_r = 25$

(e) $E_{max} = 200$ and $P_r = 30$ (f) $E_{max} = 200$ and $P_r = 35$ (g) $E_{max} = 400$ and $P_r = 10$ (h) $E_{max} = 400$ and $P_r = 15$

(i) $E_{max} = 400$ and $P_r = 20$ (j) $E_{max} = 400$ and $P_r = 25$ (k) $E_{max} = 400$ and $P_r = 30$ (l) $E_{max} = 400$ and $P_r = 35$

Fig. 4. Performance evaluations

which are provided with a preemptive fixed priority scheduler. In our model, there is no restrictive assumption on the profile of the energy source as well as tasks can consume energy with variable power. We show that the conventional greedy algorithm called FP do not work very well. We propose several variants of FP to derive more efficient solutions. The main conclusion that emerges from our experiments is the highly probability of the non existence for an optimal algorithm if the recharging rate is not taken into account both to decide when to stop the system and for how-long. However we proposed an heuristic, $EHFP_4$, simply named *slack-time based* heuristic which offers pretty good average performance in a very large range of situations. Nevertheless, in the context of energy and time constrained systems, constructing a valid schedule for every feasible task set requires clairvoyance and cannot be performed with an online scheduling algorithm. Future work will then consist in proposing an optimal clairvoyant scheduling algorithm. Our objective is to adapt to fixed priority systems, the concept of slack energy initially proposed in [2] for a dynamic priority driven system based on EDF scheduling. How to dynamically compute the slack energy at run time will be the key point of the scheduler as well as proving its optimality.

REFERENCES

[1] A. Kansal, J. Hsu, M. Srivastava, and V. Raghunathan, "Harvesting aware power management for sensor networks," in *DAC'06*, pp. 651–656.

[2] M. Chetto and H. Ghor, "Real-time scheduling of periodic tasks in a monoprocessor system with rechargeable energy storage," in *WIP RTSS'09*.

[3] A. Allavena and D. Moss, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Workshop on Power Management for Real-Time and Embedded Systems (with RTAS'01)*.

[4] C. Moser, D. Brunelli, L. Thiele, and L. Benini, "Real-time scheduling for energy harvesting sensor nodes," *Real-Time Syst.*, vol. 37, no. 3, pp. 233–260, 2007.

[5] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. B. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *IPSN'05*, pp. 457–462.

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

[7] J. Y. T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[8] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks fixed priority preemptive systems," in *RTSS'92*, pp. 110–123.

[9] R. I. Davis, K. Tindell, and A. Burns, "Scheduling slack time in fixed priority pre-emptive systems," in *RTSS'93*, pp. 222–231.

[10] R. I. Davis, "On exploiting spare capacity in hard real-time systems," Ph.D. dissertation, University of York, 1995.

[11] D. Masson and S. Midonnet, "Userland Approximate Slack Stealer with Low Time Complexity," in *RTNS'08*, pp. 29–38.