Scheduling mixed task sets in energy harvesting embedded systems

Conference Paper · July 2020

CITATIONS
0

READS
12

3 authors:

Rola el Osta
University of Nantes
12 PUBLICATIONS **12** CITATIONS
SEE PROFILE

Maryline Chetto
University of Nantes
122 PUBLICATIONS **890** CITATIONS
SEE PROFILE

Hussein El Ghor
Lebanese University
34 PUBLICATIONS **120** CITATIONS
SEE PROFILE

Scheduling mixed task sets in energy harvesting embedded systems

Rola El Osta
LS2N Laboratory - UMR CNRS 6004
University of Nantes
Nantes, France
rolaosta@hotmail.com

Maryline Chetto
LS2N Laboratory - UMR CNRS 6004
University of Nantes
Nantes, France
maryline.chetto@univ-nantes.fr

Hussein El Ghor
LENS Laboratory
Lebanese University
Saida, Lebanon
hussain@ul.edu.lb

***Abstract*—A real-time applicative software consists of both aperiodic and periodic tasks. The periodic tasks have regular arrival times and strict deadlines. The aperiodic tasks have irregular arrival times and no deadline. The objective of an optimal aperiodic task server is to guarantee minimal response times for the aperiodic tasks and no violation of hard deadlines for periodic tasks. We consider a real-time energy harvesting system composed of energy harvester, energy storage unit, uniprocessing unit and the real-time tasks. We propose a novel aperiodic task scheduler which is an extension of the slack stealing server so as to cope with fluctuations in energy availability. Experimental results show that the proposed server improves the system performance. We compare them to Background servicing in deadline miss rate and the minimum storage capacity requirement for zero deadline miss rate. The new server decreases the deadline miss rate by at least 20%, and the minimum storage capacity by at least 20% under various processor and energy utilizations.**

***Index Terms*—Earliest Deadline First, energy harvesting, aperiodic servicing, preemptive scheduling, energy management.**

I. Introduction

One of the key features for the next generation of portable wireless devices used for sensing, environmental/infrastructure monitoring, etc. is for them to become self-powered. Harvesting ambient energy from the environment to provide power for these devices has been recognized as one of the most promising technologies [1] [2]. Energy harvesting (EH) permits to deploy sensors wherever it is not possible or practical to maintain a wired network infrastructure. Techniques exist to harvest energy via photovoltaic cells, thermoelectric converters, etc. Nevertheless, new problems arise because perpetual operation of an autonomous system implies *energy neutrality*: The system should never consume more energy than available. In contrast to classical battery-operated embedded systems, batteries are solely used as energy buffers and not as primary energy sources. As a positive consequence, the cost and size of batteries are reduced significantly.

In this paper, we consider an application software with real-time requirements where tasks are given by computation times as well as amount of energy required by their execution. Specifically, periodic tasks are given by periods and deadlines whereas aperiodic tasks have unpredictable arrival times and have non deadline. Scheduling periodic tasks with no energy limitation has been extensively studied for a long time [3]. The most famous result is certainly the proof of optimality of the Earliest Deadline First (EDF) scheduler reported in [4]. We recently proved that EDF which is greedy and non clairvoyant is no more suitable if the processor is supplied thanks to fluctuating regenerative energy [5]. Nonetheless, a clairvoyant and idling variant of EDF called ED-H has been proved optimal for uniprocessor energy harvesting systems where all the tasks, periodic or not, have hard deadlines to meet [6]. In this paper, we will focus on the scheduling issue where the application software is a mixed task set composed of both periodic hard deadline tasks and soft aperiodic tasks. The paper will address the following question: how to reduce the response time of aperiodic tasks while guaranteeing no deadline miss for the periodic tasks when all the tasks execute on a monoprocessor self-powered device? The contributions described in this paper are partly resulting from the PhD work reported in [7] and [8].

The paper is organized as follows: First, we describe a novel aperiodic task server which is an extension of the Slack Stealing server [9]. Second, by means of simulations, we demonstrate significant system performance improvement in aperiodic responsiveness comparing to Background servicing. Third, to provide insights for system designers, we report simulation results that show the impact of the aperiodic server on the response time for the soft aperiodic tasks.

II. System Model and Assumptions

The energy harvesting system consists of four parts: energy harvester, energy storage, the processor and the real-time tasks (see figure 1). The energy harvester draws the energy from an environmental source and feeds into the energy storage unit with instantaneous charging rate $P_p(t)$ at time t. We assume that energy production and energy consumption can occur at the same time. The instantaneous power consumed by any task is not less than the instantaneous power drawn from the source. The energy storage is a reservoir to store energy. $E(t)$ gives the energy level of the storage at the time instant t. The energy in the storage may be used at any time later. We assume an ideal storage unit with no leakage. Energy is wasted whenever the storage unit become fully charged (i.e. $E(t) = E$ at time t). When the energy storage is empty, the processor cannot continue functioning and stops. We assume

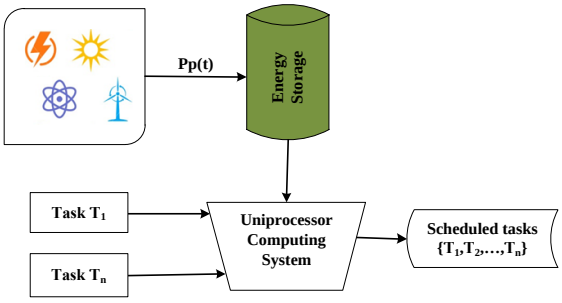

Fig. 1. Framework of an energy harvesting device

that energy production and energy consumption can occur at the same time.

A four-tuple (C_i, E_i, T_i) is associated with a periodic task τ_i and gives its Worst Case Execution Time (WCET), Worst Case Energy Consumption (WCEC) and period respectively. We assume that E_i may not be proportional to C_i [10]. A job represents a request made by a task. The first job of τ_i is released at time 0 and the subsequent ones at times $kT_i, k = 1, 2, ...$ called release times. H is the least common multiple of the request periods T_i, called the hyper-period. The processor utilization of the set of periodic tasks τ is $U_{pp} = \sum_{\tau_i \epsilon \tau} \frac{C_i}{T_i}$ which is lower than 1.

In addition, we consider Ap the stream of m soft aperiodic requests, defined as $Ap = \{Ap_i | 1 \leq i \leq m\}$ and $Ap_i = (r_i, c_i, e_i)$. r_i is the arrival time of the soft aperiodic task Ap_i. c_i and e_i are respectively the worst case execution time and the worst case energy requirement of Ap_i.

III. Background and Related Work

A. Aperiodic task servicing

The so-called TBS (Total Bandwidth Servicing) algorithm permits to execute aperiodic tasks with efficient responsiveness [11]. Once any aperiodic task arrives, it is assigned a virtual deadline. Then, it is jointly scheduled by EDF with the periodic tasks. The virtual deadline depends on processor utilization available for the aperiodic tasks called capacity of the server. An improved version of TBS called TB^* was proved to be optimal [11]. An iterative process is applied so as to shorten the virtual deadline and consequently improve the aperiodic response time. Such approach was extended so as to adapt to energy harvesting settings [12]. However, no theoretical analysis permits to state the relative performance of the so-called TB-H server.

Using the available slack of periodic tasks for advancing the execution of aperiodic requests is the basic principle adopted by the Earliest Deadline Late Server (EDL) [9]. The idea is to postpone the execution of periodic tasks as long as possible. The processor idle times of the so-called dynamic EDL schedule serve to execute the aperiodic tasks as soon as possible, profiting from the immediate available surplus of processing time called slack time. The EDL Server was proved optimal, that is, the response times of aperiodic tasks are the best achievable. In summary, whenever at least one aperiodic task arrives in the system, the dynamic EDL schedule is updated so as to determine the future time intervals where to execute the periodic tasks and the future time intervals where to execute the aperiodic ones for optimal responsiveness.

The outline of the EDL server is described by the pseudo-code of Algorithm 1.

Algorithm 1 The Slack Stealing server EDL

Require:
t: current time
$\Gamma(t)$: list of periodic tasks
$Ap(t)$: list of aperiodic tasks
while True **do**
  if $Ap(t)$ is not empty **then**
    Update the EDL schedule in order to execute the periodic tasks of $\Gamma(t)$ in the EDL busy periods
    if SlackTime(t)>0 **then**
      Schedule the tasks in $Ap(t)$ according to First Come First Serve
    else
      Schedule the tasks in $\Gamma(t)$ according to EDF
    end if
  else
    Schedule the tasks in $\Gamma(t)$ according to EDF
  end if
  $t := t + 1$
end while

B. Hard deadline scheduling with energy consideration

As EDF, the energy harvesting aware ED-H algorithm selects for execution the ready task with the closest deadline [6]. Nevertheless, ED-H may deliberately postpone the execution of this task so that no energy starvation may occur in the future. As a consequence, ED-H includes Dynamic Power Management so as to decide when the processor should be in the busy mode and when the processor should be in the idle mode. At every time instant, the decision depends on two dynamic variables respectively called *slack time* and *preemption slack energy*. The slack time represents the maximum length of the time interval where the processor could be let idle while guaranteeing no deadline violation. The preemption slack energy represents the maximum energy which could be consumed by the active task while no task can suffer from energy starvation.

Optimality of the ED-H scheduler has been established in [6]. If a hard real-time task set is schedulable by any algorithm on a platform composed of given processor, energy harvester and energy reservoir, then it is schedulable using the ED-H algorithm on the same platform.

IV. Energy harvesting aware aperiodic task servicing

In this section we will introduce a novel scheduling algorithm which is drawn from EDL and is adapted to mixed task sets in real-time systems with energy harvesting constraints. We assume that the hard deadline periodic tasks are scheduled according to the optimal scheduler ED-H.

A. SSP: Slack Stealing servicing

The Slack Stealing server, namely SSP (Slack Stealing with energy Preserving), works in the same way as the EDL server but taking into account variations on energy availability. We will see here that the concept of slack concerns both time and energy. The optimality of the SSP server is established in the sense that SSP provides the shortest aperiodic response time among all possible aperiodic servers for real-time energy harvesting systems. The main principle of the slack stealer SSP for aperiodic servicing with ED-H is to authorize aperiodic job executions as long as it does not involve a deadline violation for all the jobs generated by the periodic task set τ. Let us recall that a deadline violation occurs either because of processing time starvation (lack of time to complete a task before deadline) or energy starvation (lack of energy to complete a task before deadline).

This leads us to consider the system slack at current time t as a pair of values respectively called slack time and slack energy. The slack time of τ at time t is defined as the maximum processing time which is available at t after executing timely the tasks of τ. Slack time is a dynamic value that expresses variation of processing surplus. Its computation permits to determine whenever necessary for how long time the processor could be let either idle or busy executing additional tasks such as aperiodic ones.
The slack energy of τ at time t is defined as the maximum energy which is available at t after executing timely the tasks of τ. Slack energy is also a dynamic value that expresses variation in energy surplus. Its computation permits to determine whenever necessary how much energy could be either wasted or consumed executing additional tasks such as aperiodic ones.
The slack stealer SSP can be viewed as a task which is ready for execution whenever the aperiodic queue is non-empty. This task is suspended when the queue is empty. The slack stealer receives the highest priority whenever there is slack i.e both slack time and slack energy. Details for computing slack time and slack energy are given in [7]. It receives the lowest priority whenever there is either no slack time or no slack energy. The slack stealer SSP selects the aperiodic tasks in FCFS order.

The framework of the SSP server is described by the pseudo-code of Algorithm 2.

Algorithm 2 The Slack Stealing server SSP

Require:
t: current time
$L(t)$: list of periodic tasks at t
$Ap(t)$: list of aperiodic tasks at t
while TRUE **do**
  if Ap(t) is not empty AND energy reservoir is not empty AND $SlackEnergy(t) > 0$ AND $SlackTime(t) > 0$ **then**
    Schedule the tasks in $Ap(t)$ according to First Come First Serve
  else
    Schedule the tasks in $\Gamma(t)$ according to ED-H
  end if
  $t := t + 1$
end while

B. Implementation considerations

The complexity of SSP is $O(m.n)$, where m is the number of iterations and n is the number of periodic tasks. We note that the number of iterations, m, depends on the periods and deadlines of the hard deadline tasks, thus the complexity of algorithm is pseudo-polynomial. SSP suffers from relatively high time overheads that result from repeated slack time and slack energy calculations. Nevertheless, additional processing time and energy are exploited in the optimal slack stealing approach, whenever possible, for the aperiodic activities, by procrastination of the periodic activities.

V. Simulations and discussion

A. Introduction to the experiment

When implementing an aperiodic task algorithm, many factors affect aperiodic response time performance:

- The ratio of processor utilization and the ratio of harvested energy used by the periodic tasks determine the quantity of energy and processing times available for the aperiodic tasks.
- The aperiodic processing load and the ratio of energy needed by the aperiodic tasks have a direct effect upon aperiodic response time performance. The smaller both the quantity of processing times and energy required by the aperiodic tasks are, then the more likely it is that the aperiodic tasks are served immediately.

The experiment is devoted to compare and evaluate the performance of the aperiodic task servers through simulations. We will compare the SSP server against two background servers: Background with Energy Surplus (BES) and Background with Energy Preserving (BEP). BES serves the aperiodic tasks whenever no periodic tasks are present in the system and the energy storage is fully replenished. Under BEP, the enhanced version of BES, any aperiodic task is authorized to execute when its execution does not involve energy shortage for future occurring periodic tasks. We will consider different application profiles in terms of processing loads and energy requirements.

The experimentation aims in addition to exhibit the influence of different parameters (including the power produced by the environmental source) on the response time of the soft aperiodic tasks.

Let us recall that the ED-H algorithm serves to schedule the periodic tasks. Aperiodic tasks are served according to the FCFS policy. We now introduce the metrics applied to evaluate the performance of the SSP, BEP and BES aperiodic servers. The results that we report here assume that

- the total processing load U_p incorporates 50% of the periodic processor utilization U_{pp} and 50% of the aperiodic utilization U_{ps}.
- Identically, the total energy load U_e includes 50% of the periodic energy utilization U_{ep} and 50% of the aperiodic energy utilization U_{es}.

We will show the behavior of each aperiodic task server under different perspectives including average response time of aperiodic tasks. It is worth noting that the objective is firstly to minimize mean response time of the soft aperiodic tasks without jeopardizing schedulability of the periodic tasks with the lowest possible implementation costs i.e. minimum number of preemptions and minimum number of computing operations.

B. Simulation Environment

We have developed a simulator with Matlab. The resulting two-dimensional and three-dimensional graphs are plotted with high resolution. Our code is able to produce any simulation with given parameters that are specified by the user. The input data of the generator of periodic tasks are number *n* of tasks, hyper-period *H*, processing utilization U_{pp}, and energy utilization U_{ep}. The simulator automatically generates a periodic task set τ where each task τ_i is characterized by quadruple $(C_i, E_i, D_i, T_i) \ \mid 1 \leq i \leq n\}$. Periods and computation times are uniformly distributed, in dependance of $U_{pp} = \sum_{i=1}^{n} \frac{C_i}{T_i}$. Energy consumption of every task is proportional to its period and depends on the setting of $U_{ep} = \sum_{i=1}^{n} \frac{E_i}{T_i}$. Periodic task sets are generated so as to guarantee feasibility in terms of processing time and energy consumption i.e. $U_{pp} \leq 1$ and $U_{ep} \leq P_p$ where P_p is the average recharging power.
The input data of the aperiodic tasks are number of desired tasks *m*, processing utilization factor U_{ps} and energy utilization U_{es}. A stream of aperiodic tasks with uniform distribution is generated by simulating a Poisson aperiodic arrival pattern.

A simulation run consists of one task set composed of 20 periodic tasks. Simulations are performed on 10 hyperperiods to reduce the bias effect of random generation procedure. Each point on the curves corresponds to 100 runs. The energy storage unit is assumed to be initially full. Its capacity is equal to E_{min}, defined as the minimum size of the energy reservoir that guarantees feasibility. The energy produced by the source is not controllable. We assume that it is possible to accurately estimate short-term energy within some prediction errors margin. Four different energy profiles have been considered in our performance evaluation: constant, sine wave signal of period $\pi = 2$, a rectifier, and a pulse signal with a 20% duty-cycle (Figure 2). The power for the constant profile was supposed constant and equal to 5. The output power of the three profiles is supposed variable between 2 and 17. It is worth mentioning that E^p_{min} of each profile p should not be less than the area A_p.

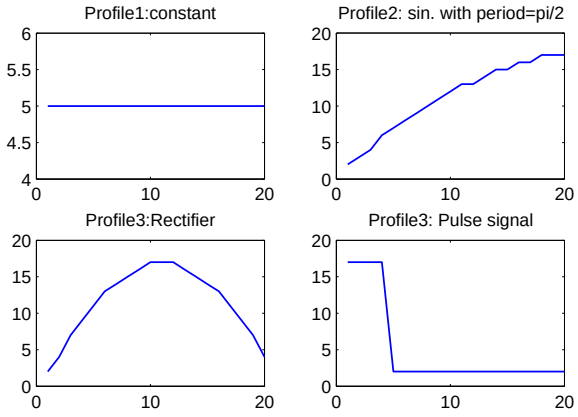

Fig. 2. Energy source profiles under study.

In our study, we address the following metrics:

1) *Average response time of aperiodic tasks* defined as average duration between arrival time and finishing time, normalized with respect to average computation time. For example, value 10 on the y-axis means average response time which is 10 times longer than computation time.
2) *Average jitter of aperiodic tasks*: Real-time systems, especially software control systems, are developed to meet the requirements of real-time automation systems. One issue is to minimize the delay and jitter of tasks. Jitter represents the induced offset between the release time of the aperiodic task and its actual starting time. Therefore, we evaluate the average jitter which is normalized with respect to its response time. Hence a value of 1 on the y-axis corresponds to a jitter equal to its response time; a value of 0 corresponds to the minimum achievable jitter time and means that the task was immediately serviced without being blocked.

C. Experiment Results for Average response time of aperiodic tasks

In this first set of experiments, the evaluation are performed for a total energy utilization applied to the system, which varies from 5% to 100%, while the total processing utilization load remains constant ($U_p = 0.6$). The four simulation experiments depicted in Figures 3, 4, 5 and 6 refer to the results obtained for constant profile, Sine wave with period $\pi = 2$, Rectifier signal and Pulse signal of 20% duty-cycle

respectively. They illustrate the mean aperiodic response time which is normalized with respect to the aperiodic computation time.

The graphs plainly show that SSP outperforms the other algorithms under all energy profiles since it benefits from time slack stealing to optimize the processor utilization. This confirms that our theoretical analysis was been performed without any restrictive assumption on the production of energy along time. BEP performs better than BES with a small deviation and exhibits a significant degradation with respect to the BEP algorithm (Figure 6) under the Pulse signal profile. It is expected that the results of the Pulse model show that the performance obtained for the three algorithms and in particular for BES is slightly less than ones obtained under the three other models. For example, the responsiveness by BES under the Pulse signal profile is at least 12.5% less than the other profiles. The reason is that the power is only harvested on 20% duty-cycle of the overall signal and BES permits aperiodic tasks to be only executed when the energy reservoir is full, which leads to an increase in aperiodic responsiveness.

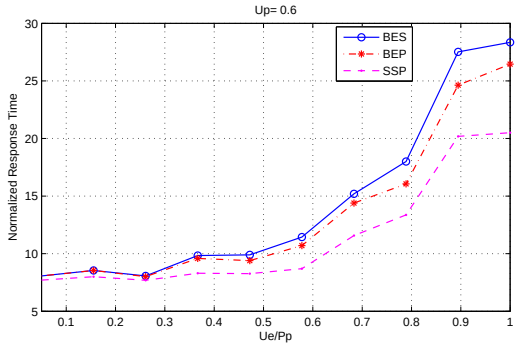

Fig. 3. Aperiodic response time with respect to U_e/P_p, for U_p=0.6 under constant profile.

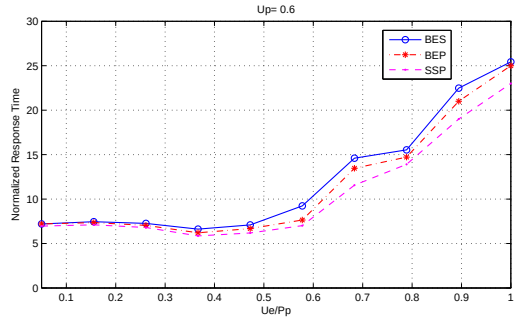

Fig. 4. Aperiodic response time with respect to U_e/P_p, for U_p=0.6 under sinusoidal signal of period $\pi/2$.

D. Experiment Results for Average jitter of aperiodic tasks

This section presents a set of experiments which shows how much SSP reduces delays and jitters of aperiodic tasks in energy constrained real-time systems, which are enforced by the operating system, control tasks, kernel mechanisms, etc. Jitter represents the induced offset between the release time of the aperiodic task and its start of execution.

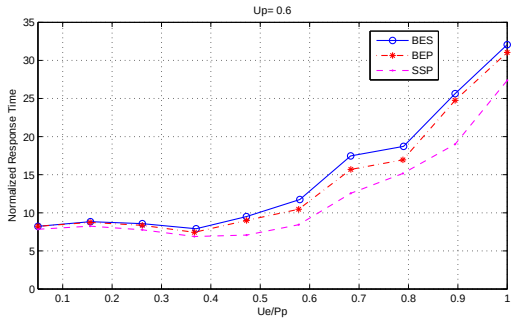

Fig. 5. Aperiodic response time with respect to U_e/P_p, for U_p=0.6 under rectifier signal.

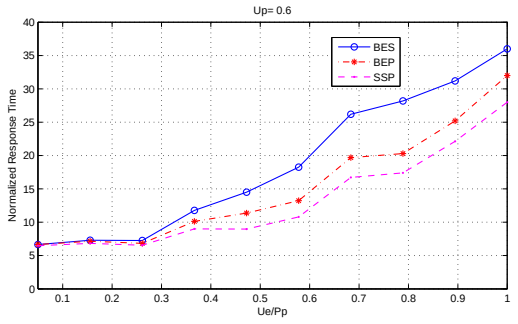

Fig. 6. Aperiodic response time with respect to U_e/P_p, for U_p=0.6 under pulse signal.

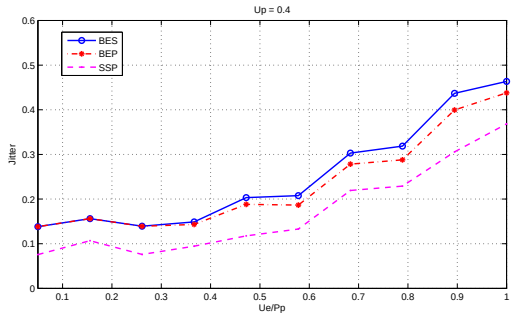

Fig. 7. Jitter time with respect to U_e/P_p, for U_p=0.4 under constant profile.

SSP is evaluated as a function of the total energy utilization U_e/P_p and is compared with the background policies in terms of jitter for a fixed total processing utilization equal to 0.4. Average jitter time is normalized with respect to its response time. Simulation results are reported in Figures 7, 8, 9, and 10 for constant profile, Sine wave with period $\pi = 2$, Rectifier signal and Pulse signal of 20% duty-cycle respectively.

From the graphs, we observe that the SSP server outstands the background servers by reducing the delay and jitter

of the aperiodic tasks for all energy loads and under all energy profiles. For example, the results in Figure 7 show that the jitter of SSP is at least 16% lower than BEP and BES. Furthermore, higher is the energy load U_e/P_p, more important is this advantage.

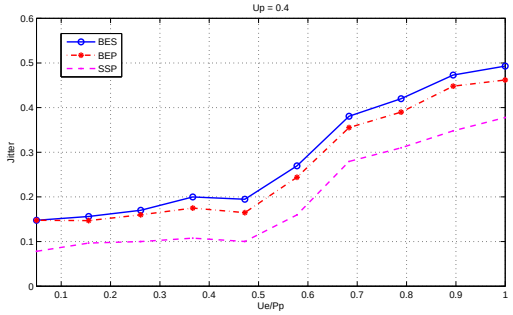

Fig. 8. Jitter time with respect to U_e/P_p, for U_p=0.4 under sinusoidal signal of period $\pi/2$.

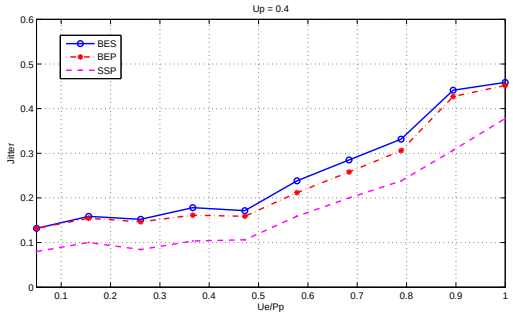

Fig. 9. Jitter time with respect to U_e/P_p, for U_p=0.4 under rectifier signal.

It is worth mentioning that BES exemplifies the inferior server for all energy profiles, and shows a significant degradation compared to BEP (Figure 10) under the Pulse signal profile owing to the power harvested by the Pulse profile and to the performance of the BES algorithm. For example, the jitter time variance between BES and BEP under the Pulse signal profile is 26% at U_e/P_p=6 (Figure 10), while it is equal to 10% under the constant profile (Figure 7).

VI. Concluding Remarks

Energy autonomous systems are becoming an important class of embedded real-time systems that utilize ambient energy to perform their computations. They do not need to charge the battery cyclically since they rely on an external energy supply.

In this paper, we investigated the problem of scheduling real-time tasks implemented on a single processor supplied with energy harvesting. We considered hard periodic and soft aperiodic real-time tasks. Periodic tasks are scheduled according to the optimal scheduler ED-H. We have described a novel aperiodic task server, proved to be theoretically optimal in

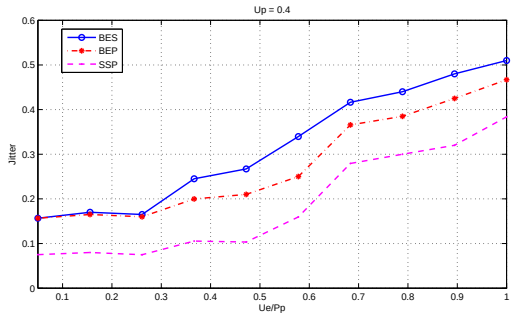

Fig. 10. Jitter time with respect to U_e/P_p, for U_p=0.4 under pulse signal.

terms of aperiodic responsiveness [13]. The so-called aperiodic task servicing algorithm, SSP, suggests to steal both processing time and environmental energy so as to execute the aperiodic tasks as soon as possible with no deadline violation and no energy starvation.

The experimental study reported in the paper permits to measure the actual performance gain of SSP in comparison to classical Background servicing techniques. Additional simulation results and details for the optimality statement of the aperiodic task server SSP can be found in [7].

References

[1] S. Chalasani and J. M. Conrad, "A survey of energy harvesting sources for embedded systems", *In Proceedings of the IEEE Southeastcon 2008*, pp. 442447, April 2008.

[2] F. Yildiz, "Potential ambient energy-harvesting sources and techniques", *The Journal of Technology Studies*, vol. 35, no 1, pp. 4048, 2009.

[3] J. Liu, *Real-Time Systems*, Prentice Hall, 2000.

[4] 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, 1973.

[5] M. Chetto and A. Queudet, "A Note on EDF Scheduling for Real-Time Energy Harvesting Systems", *IEEE Transactions on Computers*, vol. 63, no. 4, pp. 1037-1040, April 2014.

[6] M. Chetto, "Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems", *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 122-133, 2014.

[7] R. EL Osta, "Contribution to real time scheduling for energy autonomous systems", PhD thesis, University of Nantes, France, October 26, 2017.

[8] R. El Osta, M. Chetto and H. El Ghor, "An Optimal Approach for Minimizing Aperiodic Response Times in Real-Time Energy Harvesting Systems", in *Proc. of IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, 2018.

[9] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Transactions on Software Engineering*, vol. 15, no. 10, pp. 1261-1269, 1989.

[10] R. Jayaseelan, T. Mitra, and X. Li, "Estimating the Worst-Case Energy Consumption of Embedded Software", in *Proc. of 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 81-90, 2006.

[11] G.C. Buttazzo and F. Sensini, "Optimal Deadline Assignment for Scheduling Soft Aperiodic Tasks in Hard Real-Time Enviroments," in *Proc. of the 3rd IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'97)*, Como, Italy, pp. 39-48, September 1997.

[12] R. El Osta, M. Chetto and H. El Ghor, "An efficient aperiodic task server for energy harvesting embedded systems", *Proc of IEEE International Conference on Internet of Things and Intelligence System (IoTaIS)*, 2019.

[13] R. El Osta, M. Chetto and H. El Ghor, "Optimal Slack Stealing Servicing for Real-Time Energy Harvesting Systems", *The Computer Journal*, to appear, 2020.