Real-time Scheduling of periodic tasks in a monoprocessor system with a rechargeable battery

Maryline Chetto, Hussein El Ghor

Real-time Scheduling of periodic tasks in a monoprocessor system with rechargeable energy storage

Maryline Chetto and Hussein El Ghor
IRCCyN - University of Nantes
1 Rue de la Noë, F-44321 Nantes FRANCE
maryline.chetto@univ-nantes.fr, elghorh@irccyn.ec-nantes.fr

***Abstract*—We are interested in a real-time computing system that is powered through a renewable energy storage device. In this context, two constraints need to be addressed: energy and deadlines. Classical task scheduling, in particular Earliest Deadline First, only accounts for timing parameters of the tasks and consequently is not suitable when considering energy constraints. We show here how to modify Earliest Deadline so as to account for the properties of the energy source, capacity of the energy storage as well as energy consumption of the tasks. We present a scheduling framework called EDeg (Earliest Deadline with energy guarantee) and an exact feasibility test that decides for periodic task sets, whether they can be scheduled without deadline violations. To this end, we introduce the concepts of energy demand and slack energy.**

***Keywords*-scheduling; periodic tasks; earliest deadline; renewable energy;**

I. Introduction

The problem of scheduling tasks on one processor to meet deadlines and energy constraints has been the focus of great interest for about ten years. However, few papers have been devoted to emerging harvesting systems which need to operate perennially thanks to the environmental energy. A key consideration that affects power management in an energy harvesting system is that instead of minimizing the energy consumption and maximizing the lifetime achieved as in classical battery operated devices, the system operates in an energy neutral mode by consuming only as much energy as harvested.

The system we target here consists of a processing unit, an energy harvester such as a solar panel or a furl cell, and a rechargeable energy storage such as a battery or a supercapacitor. We consider a single processor that has to execute a set of independent periodic tasks.

So the problem we have to deal with is: How can we schedule the tasks so as to guarantee their timing constraints perpetually by suitably exploiting both the processor and the available ambient energy.

The goal of this work is first to construct an optimal scheduling algorithm and second, to provide an exact schedulability test. Finally, we show how to dimension the capacity of the storage unit, provided that the worst case recharging rate of the storage unit is enough for ensuring neutral operation.

II. Related works

In [1], the authors are interested in the problem of scheduling periodic tasks in the so-called frame-based systems with a rechargeable battery. In this model, all task periods are identical, all task deadlines are equal to the common period. Consequently, the order of task execution within a frame is not crucial for whether the task set is schedulable or not. Moreover, the power scavenged by the energy source is assumed to be constant and all tasks consume energy at a constant rate. A solution is presented that schedules tasks in such a way that the wasted recharging energy is minimized and the battery level is at all times within two limits, starting with a battery fully charged. The idea behind this algorithm is to insert as little idle time as necessary for recharging the battery and minimizing the length of the schedule. This work is certainly the first one to concentrate on a rechargeable system with hard real-time constraints. However, the solution only deals with frame based systems under the restrictive hypothesis that each task is characterized by an instantaneous consumption power which is constant along time.

More recently, in [7] the so-called LSA scheduling algorithm was proved to be optimal under a more generalized model including hard deadline tasks, periodic or not. LSA is a variation of the famous Earliest Deadline First scheduler: the system starts executing a task only if the task is ready and has the earliest deadline among all ready tasks and the system is able to keep on running at the maximum power until the deadline of the task. In that work, the consumption power of the computing system is characterized by some maximum value which implies that for every task, its total energy consumption is directly connected to its execution time through the constant power of the processing device.

However, in practice, the total energy which can be consumed by a task has no correlation with the worst case execution time [6]. For every task, the worst case instantaneous consumption power depends on the circuitry that is used by the task. It clearly appears as impractical to determine the energy consumption of a task from the worst case consumption power for the computing system. While it is easy to determine the average consumption power of a task

(given by the execution time and the energy consumption), this parameter is of no interest in the so-called hard real-time applications.

Furthermore, some recent studies focused on how to precisely compute the energy that is consumed by a program, independently of the average or the worst case consumption power of the computing system. More particularly, in [6] a bound on worst-case energy consumption of a task is computed through a static analysis by estimating an upper bound on the energy consumption of all individual basic blocks that make up the task.

III. Model and terminology

A. Task Set

We study the case of a Hard Real-Time system which is composed of periodic tasks. The arrival times, energy demands and deadlines of these tasks are known in advance. Such a periodic task set can be denoted as follows: $\tau = \{\tau_i, i = 1, \ldots, n\}$. A four-tuple (C_i, E_i, D_i, T_i) is associated with each τ_i. In this characterization, task τ_i makes its initial request at time 0 and its subsequent requests at times kT_i, $k = 1, 2, ...$ 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 $1 \leq i \leq n$. We define:

- the processor utilization as $U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$.
- and the energy utilization as $U_e = \sum_{i=1}^{n} \frac{E_i}{T_i}$

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

B. Energy

Our system uses an energy storage unit that has a nominal capacity, namely E, corresponding to a maximum energy (expressed in Joules or Watts-hour). The energy level has to remain between two boundaries E_{min} and E_{max} with $E = E_{max} - E_{min}$. 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.

In order to characterize the energy source, we define the WCCR (Worst Case Charging Rate), namely P_r, which is a lower bound on the harvested source power output. P_r is then the instantaneous charging rate that incorporates all losses caused by power conversion and charging process. We assume that energy production times can overlap with the consumption times.

C. Definitions

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

IV. Background materials

The problem of scheduling periodic tasks on one processor with no energy constraint has been an active area of research for more than thirty years (see, e.g., [3]). In [5], Dertouzos showed that Earliest Deadline First (EDF) is optimal among all preemptive scheduling algorithms. EDF schedules at each instant of time t, the ready task (i.e the task that may be processed and is not yet completed) whose deadline is closest to t. The EDF algorithm is typically preemptive, in the sense that, a newly arrived task can preempt the running task if its absolute deadline is shorter. This dynamic priority assignment allows EDF to exploit the full processor, reaching up to 100% of the available processing time.

In general, the implementation of EDF consists in executing tasks according to their urgency, as soon as possible with no inserted idle time. Such implementation is known as EDS (Earliest Deadline as Soon as possible). Nevertheless, in some applications, it can be preferable to postpone execution of periodic tasks, executing them by the so called EDL (Earliest Deadline as Late as possible) strategy, for example when some additional aperiodic tasks with unexpected arrival times require to be run as soon as possible [4].

V. Feasibility analysis under energy constraints

In order to develop a procedure for the feasibility assessment of a periodic task set with energy constraints, we give some definitions. Let us consider the periodic task set τ and the interval $[t_1, t_2)$.

- The processor demand of τ in $[t_1, t_2)$, is $h(t_1, t_2) = \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 - t_1 - D_i}{T_i} \right\rfloor\right) C_i$
- The energy demand of τ in $[t_1, t_2)$, is $g(t_1, t_2) = \sum_{D_i \leq t_2 - t_1} \left(1 + \left\lfloor \frac{t_2 - t_1 - D_i}{T_i} \right\rfloor\right) E_i$

The processor demand (resp. the energy demand) is a measure of how much computation (resp. energy) is requested by all the jobs which have both their release times and their deadlines, in a given interval of time. It is clear

that for a given time length and among all intervals, the initial one has the maximum fraction of processor and energy demanded by the jobs i.e. $h(t_1, t_2) \leq h(0, t_2 - t_1)$ and $g(t_1, t_2) \leq g(0, t_2 - t_1)$.

For simplicity, we respectively denote as $h(t)$ and $g(t)$ the processor demand and the energy demand of the task set τ in $[0, t)$. So, $h(t) = \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t-D_i}{T_i} \right\rfloor\right) C_i$ and $g(t) = \sum_{D_i \leq t} \left(1 + \left\lfloor \frac{t-D_i}{T_i} \right\rfloor\right) E_i$.

Without energy constraint, the exact schedulability analysis is based on the processor demand criterion and is stated as follows [2]:

Theorem 1. *A task set τ is temporally-feasible if and only if $U_p \leq 1$ and $\forall t > 0, h(t) \leq t$.*

This exact feasibility test is of pseudo-polynomial complexity since the points in which the test has to be performed correspond to deadlines within the hyperperiod T_{LCM}. Suggestions for practical improvements in testing Theorem 1 have been given in [8] and [9].

Here, we extend Theorem 1 to account for energy consumption:

Theorem 2. *A task set τ is feasible if and only if*

- *τ is temporally-feasible,*
- *$U_e \leq P_r$ and $\forall t > 0, g(t) \leq E + P_r t$*

VI. The optimal scheduling algorithm

The intuition behind the scheduling algorithm is to run tasks according to the earliest deadline first rule. However, before authorizing a task to execute, the storage level must be sufficient to provide energy for all future occurring tasks, considering their timing and energy requirements and the replenishment rate of the storage unit. And if this condition is not verified, the processor has to be idle so that the storage unit recharges as much as possible and as long as the system will be able to meet all the deadlines i.e. the system will have available time to remain idle. Following the idea described above, we propose the $EDeg$ (Earliest Deadline with energy guarantee) algorithm. To formally present the algorithm, we need to introduce two concepts:

- The *slack time of the system* 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.
- The *slack energy of the system* at current time t, is the maximum amount of energy that can be consumed from t continuously while still satisfying all the timing constraints.

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

The framework of the optimal scheduling algorithm is as follows:

Algorithm 1 Earliest Deadline with energy guarantee algorithm (EDeg)

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

We notice that:

- $EDeg$ degenerates to an EDS policy if $E_{max} = 0$ and an EDL (Earliest Deadline as Late as possible) policy if $E_{max} = \infty$.
- We never run out of storage (that is, we never dispatch tasks when there is no energy); this is obvious from the algorithm that does not allow tasks to run after E_{min}.
- We start charging the storage unit when, either it is empty or there is no more sufficient energy to guarantee the feasible execution of all future occurring tasks i.e. the system has no more slack energy.
- the charging process aims to charge at the maximum level provided there is sufficient slack time.
- We only waste recharging power when there are no pending tasks and the storage unit is full.

As a consequence, we can prove the following theorem.

Theorem 3. *Algorithm EDeg is optimal.*

VII. Practical considerations

The computations of Slack.energy(t) and Slack.time(t) are thus the keys to the operation of the EDeg algorithm. As shown in [10], the slack time of a periodic task set at a given time instant can be obtained on-line by computing the dynamic EDL schedule, with complexity O(K.n). n is the number of periodic tasks, and K is equal to $\lfloor R/p \rfloor$, where R and p are respectively the longest deadline and the shortest period of current ready tasks.

The slack energy at time t is computed only when there is at least one job, say J_j which will be released after t and has a deadline d_j that is less than or equal to that of the

highest priority job, ready at t. For such job, we compute Slack.energy(J_j,t), given by $E(t)+P_r(d_j-t)-A_j$ where A_j is the processor demand within $[t, d_j)$. Slack.energy(J_j,t) clearly represents the amount of energy surplus in the storage that can be used from t until the start time of J_j while still guaranteeing its timing and energy requirements. The slack energy of the system is determined by the minimum slack energy of all the jobs. The complexity for computing the slack energy is O(K.n) too.

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

VIII. Illustrative Example

Consider a task set τ that is composed of the tree following tasks: $\tau_1(2, 16, 7, 20)$, $\tau_2(2, 10, 4, 5)$ and $\tau_3(1, 6, 9, 10)$. The storage capacity is $E = 10$ and we assume that $E_{min} = 0$ and $E_{max} = E$. The recharging power P_r is constant and equal to 4. To simplify the illustration, we assume that tasks consume energy at constant rate. We note that the processor utilization and the energy utilization are respectively 0.6 and 3.4, consequently no more than 1 and 4. By scheduling the task set τ according to EDF on the first hyper-period i.e. from 0 to 20, we can verify that τ is temporally-feasible. The schedule which is produced by the EDeg scheduler for τ in the first hyper-period is described on *(Figure 1)*. Let us explain how this schedule is constructed in the first steps.

At time 0, the storage is full. τ_2 is the highest priority task, executes until time 2 and consumes 10. At time 2, $E_2 = 8$. τ_1 is the highest priority task and the slack energy is undefined (no job released after 2 with deadline less than 4). τ_1 executes completely until time 4 and consumes 16. 8 units of energy are produced. At time 4, $E_4 = 0$. The processor has to remain idle as long as the storage has not refilled it (predicted at time 6.5) and the latest start time has not been attained (at time 6 which is computed using EDL). At time 6, $E_6 = 8$. τ_2 is the highest priority task, executes until time 8 and consumes 10. At time 8, $E_8 = 6$. τ_3 is the highest priority task, executes and completes exactly at time 9 that coincides with its deadline.

IX. Conclusion

We have presented the framework of a monoprocessor preemptive scheduling algorithm, namely EDeg, that is a variation of EDF able to cope with energy constraints. EDeg has been designed to schedule any set of time critical tasks, periodic or not, given any energy source profile with constant power production or not and given an energy storage unit with limited capacity. Proof of optimality and validation of the exact feasibility condition attached to EDeg are the object of a Work in Progress.
This paper specifically focussed on a system that receives energy at constant rate and has to run periodic tasks with non related computation and energy requirements.

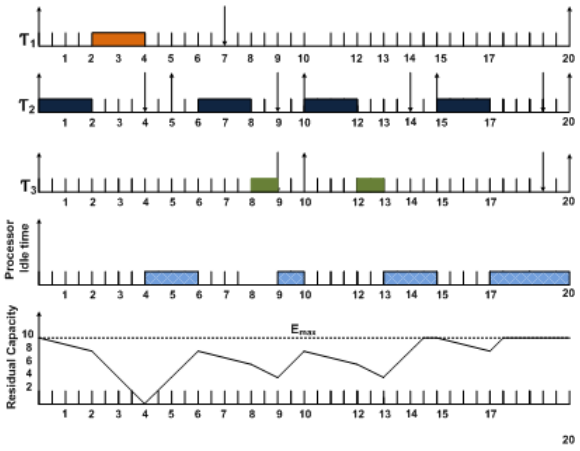

Figure 1. The EDeg schedule

References

[1] A. Allavena and D. Moss, "Scheduling of Frame-based Embedded Systems with Rechargeable batteries", *Workshop on Power Management for Real-Time and Embedded Systems*, 2001.

[2] S.K. Baruah, A.K. Mok and L.E. Rosier, "Preemptively Scheduling Hard Real-Time Sporadic Tasks on One Processor", *Proc. 11th IEEE Real-Time System Symp.*, pp. 182-190, 1990.

[3] G.C. Buttazzo, Hard Real-Time Computing Systems, Springer, 2005

[4] H. Chetto, M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm". *In Proceedings of the IEEE Transactions on Software Engineering*, Vol. 15, No. 10, pp 1261-1269, 1989.

[5] M.L. Dertouzos, "Control Robotics: The Procedural Control of Physical Processes", *Proc. Int'l Federation for Information Processing Congress*, pp. 807-813, 1974.

[6] R. Jayaseelan, T. Mitra, X. Li, "Estimating the Worst-Case Energy Consumption of Embedded Software," *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pp.81-90, 2006

[7] C. Moser, D. Brunelli, L. Thiele, L. Benini, "Real-time scheduling for energy harvesting sensor nodes", *Real-Time Systems*, Volume 37 , Issue 3, Pages: 233 - 260,December 2007

[8] I. Ripoll, A. Crespo and A.K. Mok, "Improvement in Feasibility Testing for Real-Time Tasks", *Real-Time Systems* 11, 1996.

[9] F. Zhang and A. Burns, "Schedulability Analysis for real-time systems with EDF scheduling", *IEEE Transactions on Computers*, Vol. 58, N 9, September 2009.

[10] M. Silly, "The EDL Server for Scheduling Periodic and Soft AperiodicTasks with Resource Constraints", *Real-Time Systems*, Volume 17 , Issue 1, July 1999.