Dynamic power management for fixed priority real-time systems with regenerative energy

Maryline Chetto

Dynamic power management for fixed priority real-time systems with regenerative energy

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

***Abstract*—This paper addresses scheduling and power management issues in energy harvesting embedded systems. We focus on a fixed priority based operating system such as one found in a wireless sensor node which is powered by a solar cell. We show that a classical fixed priority scheduling solution has to be reconceived. This is because new problems arise when perpetual operation is required with no deadline missing and no energy starvation being authorized. We provide a method to assign processing time and energy to tasks in a short-term perspective. The concepts of slack energy and slack time defined for dynamic priority systems are revisited for fixed priority systems.**

***Index Terms*—Real-time, Energy harvesting, Fixed priority Scheduling, Slack energy, Slack time**

I. Introduction

Energy consumption is a critical issue for real-time embedded systems. Usually, embedded devices should adopt real-time behavior, in that a task i.e an application program must complete its execution before some deadline. Most of devices such as sensor nodes do not allow any failure in meeting the deadline. They are said to be hard real-time. Thus, such embedded systems have to be provided with specific real-time operating systems able to guarantee a predictable behavior for the execution of all the tasks despite energy limitations. Firstly, a schedulability analysis is important to decide if a set of tasks can satisfy its timing requirements. The most common idea is to compute the worst-case response time of every task and compare it with its deadline. If so, secondly the scheduling algorithm should indicate how to arrange the tasks.

Classical real-time scheduling algorithms are online, preemptive, priority driven and non idling (also called work-conserving). They make their decisions on the fly, based on the list of tasks ready to be processed but ignoring the tasks which will arrive in the future. Tasks are arranged according to a priority driven policy, i.e., the ready task with the highest priority executes first and the processor never idles if at least one task is pending for execution. Over the past 50 years, various scheduling algorithms have been developed to improve the performance of systems subject to the timing constraints. One of them is Rate Monotonic (RM). The other is Earliest Deadline First (EDF) [1]. Although RM and EDF algorithms are both effective for uniprocessing systems with no energy limitation, they behave poorly in autonomous devices powered with regenerative energy.

In a so-called RTEH (Real Time Energy Harvesting) system, generally deployed in a wide wild area for surveillance, replacing battery is either costly or impractical. The RTEH device should be designed to operate perpetually in energy neutral mode only consuming the energy generated from one or several natural resources including sunlight, heat, etc. Energy harvesting implies first to collect and convert renewable energy, to transiently store electrical energy in a reservoir (super-capacitor, battery, etc.) and second to adapt the consumption of energy to its availability, dynamically [2]. As a consequence, there are a lot of technical challenges so as to make an RTEH system satisfy all its specifications expressed in terms of timing and energy constraints [3] [4].

The conventional scheduling techniques are not designed for RTEH systems because they are not energy aware: they do not handle the uncertainty in available energy and they do not predict future energy production [5]. In addition, classical schedulers are non conserving in that they do not let the processor idle if at least one task is pending for execution. In contrast, the energy harvesting aware scheduler that will be presented may decide to put the processor in the sleep mode so as, either the energy reservoir recharges or the processor is prevented to consume energy and involve an energy starvation in future. The time of necessary idleness is precisely estimated through the computations of the so-called energy laxities of the deadline constrained tasks. In summary, this paper describes the main principles of a harvesting-aware scheduler under fixed priority assumptions, which permits to prevent both energy starvation for the tasks and overflow of the energy reservoir.

The rest of this paper is organized as follows. The energy harvesting system model and some assumptions are described in Section II. Section III gives background materials. Section IV introduces the proposed scheduling scheme, namely FP-H. The concepts of slack times and slack energy under fixed priority settings are presented in Section V. Finally Section VI

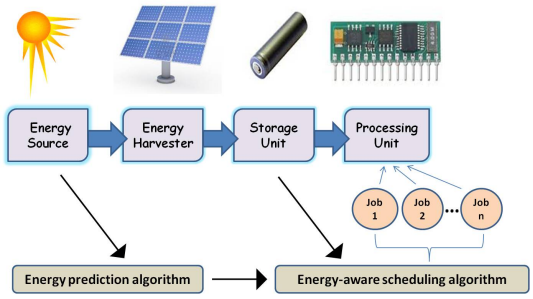

Fig. 1. A Real-Time Energy Harvesting System

gives a summary of this paper.

II. Terminology, Model and Assumption

A. Terminology

We need to introduce some terminology relative to scheduling theory.

- *Optimal scheduling algorithm*: if a task set can be scheduled to meet all its deadlines, then it will be feasibly scheduled by the optimal algorithm.
- *Clairvoyant (or omniscient) algorithm* : if the future knowledge of arriving jobs and/or the energy production is required for on line decisions.
- *Lookahead-ld algorithm*: property of a clairvoyant algorithm with ld being the shortest time interval for clairvoyance.
- *Idling (or non work-conserving) algorithm*: ability to stay idle even if there are one or more tasks pending for execution.

B. System Model and Assumptions

Hereafter, we consider a real-time energy scavenging system that consists of three major units: energy source module, energy dissipation module and energy reservoir of limited capacity.

C. System Model

1) Energy production: The real-time embedded system is powered by an energy source unit that harvests the energy from external environmental sources like wind, sun, etc. We assume that the energy harvested from time t_1 to t_2 can be calculated using the following formula:

$$E_s(t_1, t_2) = \int_{t_1}^{t_2} P_s(t)dt \tag{1}$$

Where $P_s(t)$ is the worst case charging rate (WCCR) on the harvested source power output.
We assume that it is possible to approximate the amount of energy harvested in near future.

2) Energy storage : An energy reservoir is required to continue operation even in absence of energy harvested from the environment. It can be a rechargeable battery with nominal capacity C. The reservoir stores the extra amount of energy harvested for immediate or future use. To simplify, we consider an ideal energy storage unit and so, we ignore energy wasted in charging and discharging the reservoir.
There is an upper limit to the storage device denoted by C_{max} which is the maximum capacity of the energy reservoir. The lower limit of the capacitor, denoted as C_{min}, cannot be zero since there is an energy reserved in the capacitor for worst case scenarios. During the normal operation mode and at a given time t we have

$$C_{min} \leq C(t) \leq C_{max} \tag{2}$$

3) Energy consumption: A number of tasks are statically assigned to the single processor powered with regenerative energy. the run-time system provides pre-emptive priority-based dispatching. We consider a set of n independent and preemptive periodic tasks that can be denoted as follows: $\Gamma = \{\tau_i | 1 \leq i \leq n\}$. A A four-tuple (C_i, D_i, T_i, E_i) is used to characterize a periodic real-time task τ_i, where C_i, D_i, T_i and E_i indicate the worst case execution time, the relative deadline, the period and the worst case energy consumption of task τ_i, respectively. We assume that the task set is synchronous at time 0, exhibits no synchronisation and that each invocation of a task takes its worst case execution time and its worst case energy consumption. Each task has a unique priority i where 1 is the highest priority level and n the lowest. Each task gives rise to an infinite sequence of jobs (or invocation requests), separated by an inter-arrival time T_i.

III. Background Materials

Let us describe the main results relative to scheduling in RTEH systems.

- *Any optimal algorithm is necessarily clairvoyant* for scheduling tasks on a monoprocessor RTEH system.
  As a consequence, clairvoyance i.e knowledge of at least short term future is necessary to compute the optimal scheduling sequence.
- *No lookahead-ld scheduling algorithm can be optimal if* $ld \prec D$ *where D is the longest relative task deadline*.
  From this result, a lower bound on the clairvoyance interval that is required by any optimal scheduler. This time bound has been proved to be the longest relative deadline of the application. It appears as a key parameter in the selection of the scheduler for a given application. Only a sub-optimal version of the algorithm could be implemented if the prediction technique in use is imprecise.

The preemptive Earliest Deadline First (EDF) algorithm is optimal with no energy limitation: It can successfully schedule any feasible set of periodic, sporadic or hard deadline aperiodic tasks. EDF is non clairvoyant since it makes processing

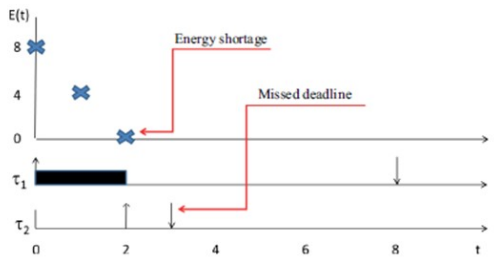

Fig. 2. Inefficiency of a non-idling scheduler for RTEH systems

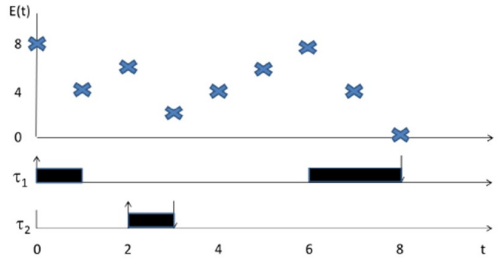

Fig. 3. Efficiency of idling and clairvoyant schedulers for RTEH systems

decisions without any knowledge of future jobs at any time. From the previous two results, we immediately conclude that neither EDF, nor RM (Rate Monotonic) can behave efficiently under energy harvesting settings (see figure 2).

The intuition behind the power management scheme is to arrange the tasks according to a given priority assignment rule. However, before authorizing the highest priority task to start execution, the residual energy capacity of the reservoir must be sufficient to supply this task for at least the next unit time-slot. Furthermore, the energy consumption in that time-slot must guarantee the energy-feasibility of all future tasks. This can be verified by considering their timing and energy requirements as well as the replenishment rate of the reservoir. If one of these conditions is not fulfilled, the processor has to idle so that the reservoir recharges sufficiently.
Following this idea, a modified version of EDF called ED-H has been presented. It was shown how the scheduler ED-H takes advantage of lookahead to improve the performance of EDF (see figure 3). The optimality of the so-called ED-H scheduler was proved in [6].

The main intuition behind ED-H is that it dynamically adapts the processor state to the run time harvested power variations without violating the deadlines of the tasks. This means that ED-H has to verify the following two conditions to let the processor busy:

1) The available energy in the reservoir is enough to execute the ready task with the highest priority even for the next unit time slot.
2) The energy consumption in that time-slot must guarantee the energy feasibility of all future occurring tasks considering their timing and energy requirements and the replenishment rate of the storage. In other words, the so-called preemption slack energy should be positive.

When these conditions are verified, ED-H arranges the tasks according to the earliest deadline first policy. However, ED-H may result in unnecessary deadline violations if one of these conditions is not fulfilled. The processor will then stay idle so that the energy reservoir is sufficiently replenished as long as the slack time does not become equal to zero.

IV. A GENERAL POWER MANAGEMENT SCHEME FOR RTEH SYSTEMS

Most of the works on RTEH systems were based on the EDF (Earliest Deadline First) dynamic priority scheme. Although EDF can support systems with higher utilization than fixed-priority (FP) algorithms, it reveals to be more difficult to implement in commercial kernels that do not provide explicit support for timing constraints, such as periods and deadlines. Moreover, when the system is overloaded, EDF can produce unbounded and unpredictable deadline misses. In contrast, FP provides better stability behaviors in such cases because it is always known in advance which task will miss its deadline (the lowest priority one). As a consequence of high predictability, low overhead, and ease of implementation, the Fixed Priority scheduling schemes are widely adopted in real-time operating systems.

A. Principles of FP-H

With the so-called Fixed Priority (FP) assignment rule, each task statically receives a priority off-line and all the jobs generated at run time by the same periodic task inherits its priority. In classical applications with no energy limitations that use the FP assignment rule, jobs in periodic tasks can be guaranteed statically with the Rate-Monotonic (RM) priority assignment initially studied in Liu and Layland [1], or with the Deadline Monotonic (DM) proposed by Leung and Whitehead [7]. The RM and DM priority assignment schemes were proved to be optimal in the class of fixed priority based schedulers, in the sense that, if there exists any fixed priority assignment rule that generates a feasible schedule, then RM and DM scheduling policies generate a feasible schedule for the same task set. An exact feasibility test for synchronous task sets on a single processor can be performed in pseudo-polynomial time (using critical instant analysis). As EDF, these schedulers are work conservative (also called non idling) since they never let the processor inactive when at least one job is pending for execution. And idling the processor does not improve the schedulability . Under energy harvesting settings, priority-driven schedulers differ from each other not only on how priorities are assigned to jobs but also on how to decide when to execute a job and when to let the processor inactive.

The work reported in this paper aims to adapt any priority driven work-conserving scheduler commonly used in energy non-constrained environments so as to make them as efficient in RTEH environment in terms of optimality and robustness. From the results listed previously, our approach to dynamic processor management is necessarily idling and lookahead . In the so-called FP-H scheduler, as in ED-H, the dynamic management of idle processor capacity exploits idle time intervals to obtain a power adaptation for the processor while ensuring that all jobs adhere to their timing constraints. It can be demonstrated that FP-H permits us to guarantee the so-called *energy neutral operation mode* in which the system never consumes more energy than harvested while satisfying all its timing requirements.

B. Fundamental concepts

Let us list and define the new concepts which are necessary for designing an optimal scheduler under RTEH considerations. Let t be the current time in the schedule produced for the task set τ by a certain scheduling algorithm.

- *The slack time of a task set* Γ at current time t, denoted $ST_\Gamma(t)$ is the maximum continuous processor time that could be available from time t while still guaranteeing the feasability of all the tasks in the set Γ.
- *The slack energy of a task set* Γ at current time t, denoted $SE_\Gamma(t)$ is the maximum continuous energy that could be made available and consumed from time t by tasks other than that of $\Gamma(t)$ while still guaranteeing the feasability of all the task set $\Gamma(t)$.
- *The preemption slack energy of a task set* Γ at current time t, denoted $PSE_\Gamma(t)$ is the maximum continuous energy that could be consumed from time t by the current active task while still guaranteeing the feasability of all the tasks with a higher priority i.e. the tasks that may preempt the current active one.

In other terms, making the processor idle from time t when the slack time is zero leads to jeopardize the deadline constraints due to time starvation. And making the processor busy for task execution from time t when the so-called slack energy is zero leads to jeopardize the deadline constraints of some higher priority tasks due to energy starvation.

C. The Power management algorithm

Let us give an informal description of the FP-H algorithm. FP-H says that:

- the processor cannot be active if either the reservoir is deplenished or executing any task would prevent at least one future task from being executed timely because of energy starvation i.e. the system has no preemption slack energy at t.
- the processor should not be inactive if the reservoir is fully replenished to avoid wasting energy. When the reservoir is neither full nor empty and the system has both slack time and preemption slack energy, the scheduler may decide on the processor state.
- the processor should not be inactive if the system has no slack time.
- no tasks are dispatched when there is no energy
- charging the reservoir is achieved either if it is empty or if there is not enough available energy to guarantee the feasible execution of all the tasks.
- the charging process is flexible since it authorizes to charge the storage unit during any time period provided there is slack time and the storage unit has not replenished. We only waste recharging power when there are no ready jobs and the storage unit is full.

Let consider a given task set Γ that is known to be feasible for the real-time energy harvesting model. Let $Q_r(t)$ be the queue of uncompleted tasks ready for execution at t. From what precedes, the FP-H algorithm obeys the following rules.

- **Rule 1:** The ready tasks are ordered by decreasing priority in $Q_r(t)$.
- **Rule 2:** The processor is imperatively idle in $[t, t+1)$ if $Q_r(t) = \phi$ or $C(t) \approx 0$ or $SE_\Gamma(t) = 0$.
- **Rule 3:** The incoming power is wasted in $[t, t+1)$ if $Q_r(t) = \phi$ and $C(t) = C$.
- **Rule 4:** The processor is imperatively busy in $[t, t+1)$ if $Q_r(t) \neq \phi$ and, either $C(t) \approx C$ or $ST_\Gamma(t) = 0$.
- **Rule 5:** The processor can be idle or busy if $Q_r(t) \neq \phi$, $ST_\Gamma(t) > 0$ and $PSE_\Gamma(t) > 0$.

This power management strategy under the dynamic priority scheduler EDF, called ED-H was proposed in [6]. ED-H was proved optimal and robust. The implementation costs of ED-H mainly are due to the run-time computations of both the slack time and the slack energy of the system. Computing the slack time of a periodic task set at run-time, under the dynamic priority rule EDF has been described in [8]. Efficient implementations have been proposed in [9]. Some off-line computations can be done in order to compute efficiently the EDL (Earliest Deadline as Late as possible) schedule without wasting too much time and involving acceptable overhead at run-time. Under EDL, the tasks are processed as late as possible so as to guarantee the maximum idle time in a given interval.
Before the system begins to operate, we compute the static EDL schedule for the given task set. More precisely, we estimate the localization and the duration of the idle times within the EDL schedule produced at time $t = 0$ till the end of the hyperperiod. The EDL schedule can be described by means of two vectors respectively called static deadline vector and static idle time vector. We proved that the complexity of the approach is O(kn) where k is the number of iterations and n is the number of periodic tasks. The number k depends on the periods and deadlines of the tasks, thus the complexity of the algorithm is pseudo-polynomial.

The complexity for calculation of the slack energy depends

on how the amount of environmental energy produced between two instants is predicted and estimated. Depending on whether we have an energy source at constant power or not, this calculation will therefore be more or less complex. The common energy sources are in different forms: for example, solar energy varies according to the parabolic law with a very large period, the piezoelectric energy is often in the form of pulsations, the energy dissipated by the human body can be considered constant, etc.

The optimality of FP-H signifies that if it cannot produce a valid schedule for any FP-schedulable task set Γ, then no other processor management policy respecting the priority assignment FP is able to do it on the same energy harvesting platform [13].

V. Computing the slacks in FP-H

A. Definitions

Let us define the notions of priority level, busy periods and idle periods as in [11].

- *Level i busy periods* are defined as periods where the processor serves tasks with priorities higher than or equal to i.
- *level i idle periods* are defined as periods where the processor serves tasks with priorities lower than i
- Given at time t, *the slack time of a job set* $\Gamma(t)$, (denoted as $ST_\Gamma(t)$) is the longest time interval such that, if the execution of any job in $\Gamma(t)$ does not start before the end of that interval , all jobs will meet their deadlines.
- Given at time t, the level-i slack time denoted as $ST_i(t)$ is defined to be the maximal time interval after t such that, if the execution of τ_i and any other tasks with priorities higher than τ_i and arrival times later than t start no later than $t + ST_i(t)$),τ_i can still meet its deadline.

B. Slack time under fixed priority assignment

The slack time of a system represents the surplus of time the system has either to execute additional tasks either to stay idle without jeopardizing the timing constraints of the periodic tasks. In a fixed priority context, there is slack time in the system at a given time instant if there is slack available at all priority levels. In order to compute $ST_\Gamma(t)$, we need to calculate,for each priority level, the available slack time, denoted $ST_i(t)$. Finally, $ST_\Gamma(t) = \min_i ST_i(t)$. $ST_i(t)$ gives the maximum amount of time the task τ_i can be delayed from time t without missing its deadline d_i. The value $ST_i(t)$ is consequently equal to the total number of unused time units at priorities higher than or equal to i between t and the deadline of the current job of the task τ_i ready at time t. The Dynamic Slack Stealing (DSS) algorithm described in [10] permits to compute $ST_\Gamma(t)$ for periodic tasks as the algorithm described in [11] and [12].

The implementation can also be achieved through off-line mapping the processor schedule for the hard periodic tasks over their hyperperiod (the least common multiple of task periods). The mapping is then inspected to determine the slack present between the deadline on one job of a task and the deadline on the next. The values found are stored in a table. At run-time, counters keep track of the slack which may be stolen at each priority level. These counters are decremented depending on which tasks, if any, are executing and updated thanks to the table whenever each task completes [5].

C. Slack energy under fixed priority assignment

By definition, the slack energy at a given time instant gives the energy surplus of the system at that time instant without jeopardizing the feasibility of the system [13]. In a fixed priority context, there is slack energy in the system at a given time instant if there is slack energy available at all priority levels. In order to compute $SE_\Gamma(t)$, for each priority level i, calculation of the available slack energy, denoted $SE_i(t)$ should be performed. $SE_i(t)$ gives the maximum amount of energy that the tasks with lower priorities can consume from time t without involving energy starvation for τ_i. This value is consequently equal to the total number of unused energy units consumed at priorities higher than or equal to i between t and the deadline of the current job of the task τ_i ready at time t. $SE_i(t)$ should be computed from:

- the amount of energy immediately available in the reservoir at time t, say $C(t)$
- the amount of energy which will be later produced by the environmental source from t to the effective deadline of τ_i
- and the energy required by all the tasks with higher priorities released at or after t up to the effective deadline of τ_i

As a consequence, the preemption slack energy,$PSE_\Gamma(t)$, is calculated from the slack energy $SE_i(t)$ for each priority level i which is greater than the priority of the current active task. Thus, $PSE_\Gamma(t) = \min_{i \geq a} SE_i(t)$ where a is the priority of the current active task.

VI. Conclusion

An embedded system such as sensor should be designed to operate perpetually as it is often impractical or costly to recharge or replace batteries. Energy harvesting technology can serve to satisfy this objective by taking profit of the energy available in the surrounding environment of the sensor.

To develop an energy aware real-time scheduler that guarantees an energy neutral operation, both energy stored in the storage unit, energy produced by the source and energy consumed by the tasks need to be considered with the deadlines of the tasks guaranteed. Any energy-aware scheduling algorithm consists of a set of rules that govern, first how the jobs are ordered to access to the computing unit (priority driven scheduling capability) and, second when and under what conditions the processor is authorized to consume energy for job execution (dynamic power management capability).

In this work, we focussed on fixed priority driven systems. We have described FP-H, an idling and clairvoyant scheme which provides power management and scheduling support. We have shown that it is based on the dynamic computation of two fundamental values: the slack time and the preemption

slack energy.

Our current works aim at formulating an exact schedulability test for FP-H and at proving the optimality of the FP-H algorithm among the class of fixed priority schedulers under energy harvesting settings.

REFERENCES

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

[2] S. Priya, D.-J. Inman. Energy Harvesting Technologies. Springer-Verlag, New York (USA), 2009.

[3] H. Aydin, R. Melhem, D. Mosse, P. Mejia-Alvarez. P*ower-aware scheduling for periodic real-time tasks*. IEEE Transactions on Computers, vol.53, no.5, pp.584-600, 2004.

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

[5] J. Liu. *Real time Systems*, NJ:Prentice Hall, 2000.

[6] M. Chetto, *Optimal Scheduling for Real-Time Jobs in Energy Harvesting Computing Systems*, IEEE Transactions on Emerging Topics in Computing, 2(2), pp. 122-133, June 2014.

[7] J. Y. T. Leung, J. Whitehead. *On the complexity of fixed-priority scheduling of periodic, real-time tasks*. Performance Evaluation, vol.2, no.4, pp.237-250, 1982.

[8] H. Chetto, M. Chetto. *Some Results of the Earliest Deadline Scheduling Algorithm*, IEEE Transactions on Software Engineering, Volume 15, Issue 10, pp. 1261-1270, 1989.

[9] M.Chetto-Silly. *The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints*, Real-Time Systems 17(1), pp 87-111, 1999.

[10] S. Midonnet, D. Masson, R.Lassalle. *Slack-Time Computation for Temporal Robustness in Embedded Systems*. IEEE Embedded Systems Letters, Institute of Electrical and Electronics Engineers, 2 (4), pp.119-122, 2010.

[11] J. P. Lehoczky. *Fixed Priority Scheduling of Periodic Task Sets With Arbitrary Deadlines*, Proceedings 11th IEEE Real-Time Systems Symposium, Lake Buena Vista, FL, USA, pp. 201-209, 1990.

[12] L. Niu. *Energy efficient scheduling for hard real-time systems with fixed-priority assignment*, International Performance Computing and Communications Conference, Albuquerque, NM, USA, 2010, pp. 153-160

[13] M. Chetto. *Optimal Dynamic Processor Management for Priority Driven Real-Time Systems with Renewable Energy*, Technical report, LS2N laboratory, 15 pages, November 2017.