Real-time Resource Management in Smart Energy-Harvesting Systems

Mohamed Irfanulla, Audrey Queudet, Maryline Chetto, Lamia Belouaer

Real-time Resource Management in Smart Energy-Harvesting Systems

Mohamed Irfanulla MOHAMED ABDULLA
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004
F-44000 Nantes, France
mohamed-irfanulla.mohamed-abdulla@ls2n.fr

Audrey QUEUDET
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004
F-44000 Nantes, France
audrey.queudet@ls2n.fr

Maryline CHETTO
Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004
F-44000 Nantes, France
maryline.chetto@ls2n.fr

Lamia BELOUAER
E-Cobot
Nantes, France
l.belouaer@e-cobot.com

***Abstract*—Energy harvesting is an emerging technology that enhances the lifetime of Internet-of-Things (IoT) applications. Satisfying real-time requirements for these systems is challenging. Dedicated real-time schedulers integrating both timing and energy constraints are required, such as the ED-H scheduling algorithm[1]. However, this algorithm has been proved to be optimal for independent tasks only (i.e., without considering any shared resources), thus preventing its confident deployment into computing infrastructures in which tasks are mostly interdependent. In this paper, we first derive worst-case blocking times and worst-case blocking energy for tasks sharing resources managed by the DPCP protocol[2] and scheduled under the ED-H scheme. Then, we provide a sufficient schedulability test for ED-H–DPCP guaranteeing off-line that both timing and energy constraints will be satisfied, even in the presence of shared resources.**

***Index Terms*—Energy Harvesting, ED-H, DPCP, Shared resource, Schedulability test**

I. Introduction

Battery-operated Cyber-physical systems (CPSs) have to address the problem of energy scarcity. These system can additionally benefit from energy harvesting mechanism to deal with energy constraints. In this paper, we consider Real-time Energy Harvesting (RTEH) systems as defined in [1]. Such smart energy systems in charge of collecting and processing the raw data gathered from sensors, must satisfy timing constraints through a smart real-time scheduler which usually relies on a preemptive priority-based algorithm. We are interested in smart energy-harvesting systems that consist of an *energy harvester* that scavenges energy from ambient sources and stores it in an *energy storage unit* such as a battery or super-capacitor. Fig. 1 presents the system architecture of a typical real-time energy harvesting system. As an autonomous system, a smart energy-harvesting system should embed a smart energy management mechanism so as to optimize the use of the energy retrieved from the environment and guarantee an *energy neutrality* operation to the system (i.e., the system never consumes more energy than harvested). Most of real-time systems are controlling ones and consequently, they implement periodic tasks. The online Earliest Deadline First (EDF) scheduler [3] that executes first the task with the closest deadline has been proved optimal when energy is not limited. A variant of EDF, namely ED-H [1], was more recently proposed so as to provide an optimal strategy to schedule real-time tasks under energy harvesting settings. An exact feasibility test was also given with robustness properties.

Motivation. For a large number of embedded real-time systems, applications exhibit resource constraints sharing both virtual and physical objects such as data structures, variables, main memory area, file, set of registers, I/O devices or database. Many shared resources do not allow simultaneous accesses and require *mutual exclusion*. The underlying system must then be able to ensure mutually exclusive access to these shared resources, otherwise data inconsistency will appear. To achieve mutual exclusion, the shared resources subject to mutual exclusion constraints must be serially executed, causing additional delays due to *priority inversion*. Indeed, a higher-priority task requesting a shared resource already used by a lower-priority task can be blocked, thus delaying the completion time of the task and leading eventually to a deadline miss. The ED-H scheduling algorithm has been proved *optimal* under the assumption that all tasks are independent (i.e., their executions do not depend on the execution of other tasks). However, this is not a realistic assumption. In the case of smart systems like mobile robots, tasks implicitly or explicitly synchronize through resource sharing. Up to now, the ED-H scheduling algorithm could not satisfactorily be implemented in applications with shared critical resources.

This work is partially supported by a French grant from the Isite NExT (Project entitled "Optimisation et Gestion en Temps Réel de la Consommation Energétique d'un Système Cobotique") within the New Partnerships framework.

978-1-6654-9792-3/22/$31.00 ©2022 IEEE

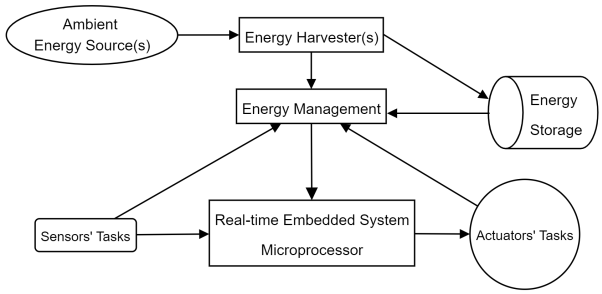

Fig. 1. A typical smart energy-harvesting system

Contributions. This paper is concerned with a less restrictive model in which the application tasks need to access critical resources. We consider a set of tasks which are scheduled by ED-H while concurrent accesses to critical shared resources are managed by DPCP [2]. Our contribution is threefold: (i) we show that the ED-H feasibility test is not applicable anymore in the presence of shared resources, (ii) we determine the worst-case blocking times and the worst-case blocking energy of the tasks when jointly using the ED-H scheduler and the DPCP protocol for managing the accesses to shared resources and (iii) we propose a new sufficient schedulability test in order to take into account blocking delays incurred by the resources in the offline analysis of the system. To the best of our knowledge, this is the first work that studies the problem of resource sharing when a real-time energy-harvesting scheduling scheme is considered.

Paper organization. The paper is organized as follows. In section II we review some related work and present some requirements for smart energy-harvesting systems. In section III, the system model is described by focusing on the resource-related concerns. We review some background material related to ED-H and DPCP in section IV. In section V, the schedulability analysis under ED-H is detailed and we show its non-applicability to systems with shared resources. In section VI we give new formulas for computing the worst-case blocking computation of tasks from both timing and energy points of view, and we derive the feasibility test for the ED-H scheduling algorithm under resource sharing. Examples are given for illustrative purpose. Finally, we conclude the paper and give some future works in section VII.

II. Related Work

Dynamic Voltage Scaling (DVS) [4] and Dynamic Power Management (DPM) [5] are famous techniques for minimizing the processor energy consumption. In the DVS approach, the minimization of the energy consumption is performed by slowing down the operating voltage and clock frequency of the processor. As regards DPM techniques, they consist in switching the processor to a low-power state (i.e., sleep mode) during idle time intervals (i.e., periods of time during which there is no tasks to execute). Although used in various energy-aware scheduling algorithms proposed for both uniprocessor and multicore systems (see [6] for a survey), relatively few studies have considered task synchronization issues. *Synchronization access protocols* are the well-know mechanisms for supporting data consistency. Resource sharing protocols are independent mechanisms that can be combined with base scheduling algorithms including the well-known priority driven ones such as RM (Rate Monotonic), DM (Deadline Monotonic) and EDF (Earliest Deadline First) [3]. They not only enforce mutually exclusive accesses to shared variables to avoid race conditions. They also avoid deadlocks, starvations and priority inversions. For uniprocessor preemptive real-time systems, protocols such as the so-called *PCP (Priority Ceiling Protocol)* [7] (and its dynamic-priority based version – *DPCP* [2]) and *SRP (Stack Resources Protocol)* [8] have been proposed.

The problem of determining whether a real-time task set with resource constraints is schedulable or not has been proved as *NP*-hard [9]. Some papers have addressed the problem of scheduling task sets that share resources in exclusive mode [10]–[12]. They mainly focus on enhancing the schedulability analysis to provide tighter schedulability conditions. However, no energy constraints are considered in the schedulability analysis.

Some works addressed the impact of energy when scheduling dependant tasks [13]–[16]. Above works use the classical resource access protocols to ensure mutual exclusion w.r.t shared resources. Some recent work [17] present an operating system kernel named EN*OS* for energy-neutral real-time systems. Depending on the energy level, the system switches between different energy modes which also changes the task set to execute according to the available energy. However, this approach deals only with independent tasks and mainly relies on hardware interrupt handling routines.

To the best of our knowledge there are no existing works that take into consideration energy-harvesting constraints in the scheduling problem of dependent real-time tasks.

III. System Model

Our system model is divided into three sub-models: (i) a *task model* which gathers the key parameters of the tasks, (ii) a *resource model* which describes the characteristics of the resources concurrently accessed by tasks, and (iii) the *energy model* which defines how the energy is produced, stored, and consumed in the system.

A. Task model

We consider a set of n periodic tasks denoted by $\tau = \{\tau_1, \tau_2, .., \tau_n\}$, on a uniprocessor architecture. Each task τ_i is characterized by its initial release time ϕ_i, its worst-case computation time C_i, its period T_i, its relative deadline D_i (with $D_i \leq T_i$), and its worst-case energy consumption E_i which is not necessarily proportional to C_i. We consider a synchronous task set (i.e. all offsets are set equal to 0). The periodic task τ generates a (potentially infinite) sequence of jobs that inherit timing parameters from the periodic tasks, denoted by ω. Each job J_k in ω is assigned a release time $r_k = \phi_i + j \times T_i$ (with $j \geq 0$) and an absolute deadline $d_k = r_k + D_i$. Let d_{Max} be the greatest absolute deadline among all active jobs. The processor

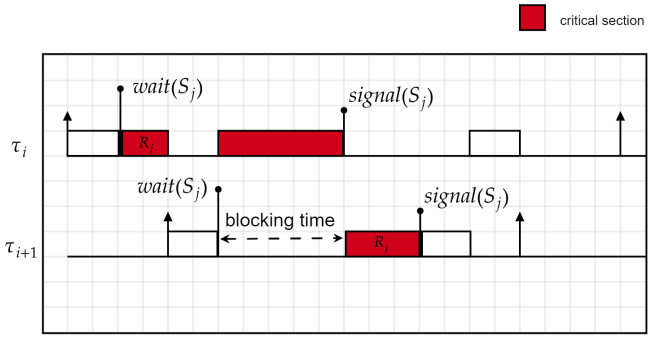

Fig. 2. Blocking time caused by a shared critical section [18]

utilization factor is defined as $U_\tau^p = \sum_{i=1}^{n} \frac{C_i}{T_i}$. The average energy consumption of task set τ per unit time is defined as $U_\tau^e = \sum_{i=1}^{n} \frac{E_i}{T_i}$. Thus U_τ^p cannot exceed one and U_τ^e cannot be greater than the average power drawn from the energy source during one unit time.

B. Resource model

We consider a set $\Omega = \{R_1, ..., R_p\}$ of p shared resources in the system. Each periodic task may access one or several of these critical resources. Each resource R_j is guarded by a distinct semaphore S_j a task must lock (resp. unlock) when entering (resp. leaving) a critical section. It uses the two classical primitives *wait*() and *signal*() for requesting and releasing the access to the resource respectively, as illustrated in Fig. 2. All tasks blocked on the same resource are kept in a queue associated with the semaphore. When a running task executes a wait on a locked semaphore, it enters a blocked state, until another task executes a signal primitive that unlocks the semaphore (see Fig. 2). σ_i denotes the set of semaphores used by τ_i. We denote the blocking set of task τ_i as Z_i (i.e. the set of semaphores that can block τ_i). The duration (resp. the energy consumption) of the longest critical section of task τ_i guarded by S_k is denoted by $\delta_{i,k}$ (resp. $\varepsilon_{i,k}$). The instant at which a task τ_i accesses the resource R_k is denoted by $a_{i,k}$.

C. Energy model

The energy harvested from the ambient source is converted into electrical power with an instantaneous charging rate $P_p(t)$. $E_p(t_1,t_2) = \int_{t_1}^{t_2} P_p(t)dt$ represents the amount of energy harvested in time interval $[t_1,t_2)$. The energy storage unit has a maximal capacity denoted by C. Its energy level at time t is denoted by $E(t)$. The total energy consumed by active jobs of τ on the time interval $[t_1,t_2)$ is denoted by $E_c(t_1,t_2)$.

IV. Background materials

A. The ED-H scheduler

An optimal dynamic priority scheduling algorithm, namely ED-H [1] was proposed in 2014 to take into account both real-time and energy harvesting constraints. ED-H is a variant of the famous Earliest Deadline First (EDF) scheduling algorithm [3]. It is a dynamic priority-driven scheduler that makes its decisions online considering both timing and energy constraints. It requires an accurate prediction of the near future of both the harvested energy and the energy consumed by the tasks. ED-H is semi-online, more exactly online with lookahead-D [19]. The scheduler manages the *idle* or *busy* state of the processor dynamically, depending on two key parameters: the *slack time* (i.e., the maximum continuous processor time that could be available from current time while still guaranteeing the deadlines of all the jobs) and the *slack energy* (i.e., the maximum amount of energy that could be consumed from any time instant while still satisfying all the energy and timing constraints of all the jobs). In other words, the scheduler executes the active jobs in view of their absolute deadline, their energy consumption, the remaining energy in the storage unit, and the energy produced by the source. While the storage unit is empty, the processor must be inactive. On the contrary, if there is a ready job waiting for execution and if the energy level in the storage unit is maximum, the processor must be busy executing the ready job so as not to waste energy.

Let $L_p(t_c)$ be the list of jobs ready for execution at current time t_c. The scheduler ED-H algorithm can be described by the following rules:

- **Rule 1:** The order of priority of EDF is used to select the future running job in $L_p(t_c)$.
- **Rule 2:** The processor is imperatively *idle* in $[t_c;t_c+1)$ if $L_p(t_c) = \emptyset$.
- **Rule 3:** The processor is imperatively *idle* in $[t_c;t_c+1)$ if $L_p(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1) $E(t_c) \approx 0$
  2) $PreemptionSlackEnergy(t_c) \approx 0$
- **Rule 4:** The processor is imperatively *busy* in $[t_c;t_c+1)$ if $L_p(t_c) \neq \emptyset$ and one of the following conditions is satisfied:
  1) $E(t_c) \approx C$
  2) $SlackTime(t_c) \approx 0$
- **Rule 5:** The processor can equally be *idle* or *busy* if $L_p(t_c) = \emptyset$, $0 < E(t_c) < C$, $SlackTime(t_c) > 0$ and $PreemptionSlackEnergy(t_c) \approx 0$.

Clearly, ED-H implements a smart dynamic power management procedure that specifies when to put the processor *busy* (resp. *idle*) so as not to waste energy (resp. not to consume energy too greedily). Optimality of ED-H means that if a task set cannot be feasibly scheduled on a given RTEH platform, this task set cannot be feasibly scheduled by any other scheduler on the same platform.

B. DPCP protocol

In real-time systems, tasks may exchange data through shared memory and concurrent accesses may lead to data inconsistency. Without any resource access protocol, tasks may be blocked many times before the completion of their jobs, thus leading to deadline violations.

We focus here on *DPCP (Dynamic Priority Ceiling Protocol)* [2]. DPCP associates a ceiling value $Ceil(S_i)$ with every semaphore S_i associated to resource R_i, which is equal to the maximum of the priority values of all jobs that might manipulate R_i. An additional variable called $CurCeil(t)$ represents, at any time instant, the maximum ceiling value of

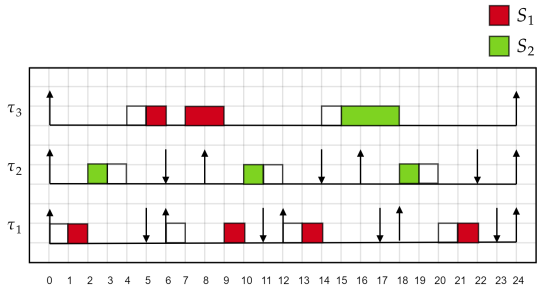

Fig. 3. EDF-DPCP schedule example

all semaphores that have been locked. At initialization time, this value is equal to zero (i.e., $CurCeil(0) = 0$).

A task in a critical section can be preempted by a higher priority task that is not waiting to enter the same critical section. However, due to conflicts w.r.t shared resources, a high-priority job may have to wait for the execution of a low-priority job holding the resource. Such waiting time is referred to as *worst-case blocking time*. Authors in [7] have shown that, in the worst-case, each job has to wait for at most one lower-priority job in critical section. Consequently, for any job of task τ_i, the worst-case blocking length B_i is equal to the duration of the longest critical section among the set of lower-priority tasks that may block τ_i.

Example 1. Let us consider three periodic tasks τ_1, τ_2 and τ_3 with two semaphores S_1 and S_2 protecting the access to shared resources R_1 and R_2 respectively. The task set $\tau_i = (C_i, T_i, D_i)$ parameters are the followings : $\tau_1 = (2,6,5)$ $\tau_2 = (2,8,6)$ and $\tau_3 = (8,24,20)$. Task τ_1 manipulates R_1 at time $a_{1,1} = 1$, task τ_2 manipulates R_2 at time $a_{2,2} = 0$. Task τ_3 requests R_1 at time $a_{3,1} = 1$ and R_2 at time $a_{3,2} = 5$.

Fig. 3 shows how the periodic task set is scheduled by EDF using DPCP for managing shared resource accesses. The ceiling priority of S_1 is $Ceil(S_1) = 1$ and S_2 is $Ceil(S_2) = 2$. Worst-case blocking times that tasks may experience are $B_1 = 3$, $B_2 = 3$ and $B_3 = 0$ (see [2] for computation). First, higher priority task τ_1 starts its execution and accesses the resource R_1 at time $t = 1$. When task τ_2 starts at time $t = 2$ the resource R_2 is available. Next, task τ_3 executes and at time $t = 5$, locks the semaphore S_1 which is available. At time $t = 6$, job J_2 of task τ_1 is released. As τ_1 priority is higher, it preempts task τ_3 and executes. When τ_1 requests the resource at time $t = 7$, it is blocked as S_1 has already been locked by τ_3, the ceiling priority is increased to 1 and τ_3 continues using S_1 until the resource is released by τ_3. τ_1 completes its execution right after the semaphore S_1 is available. Likewise, at time $t = 16$, when τ_2 requests semaphore S_2, it is blocked until τ_3 releases the resource.

V. Schedulability Analysis under ED-H

The schedulability analysis at the design phase of a real-time system aims at checking whether the tasks can consistently meet their deadlines when scheduled by a given scheduling algorithm. For RTEH systems, such an analysis requires an accurate knowledge of the profile of the energy source and that of all the characteristics of both the real-time workload and the hardware platform. The analysis should be performed off-line whenever possible so as to guarantee *a priori* any deadline missing caused by energy and/or time starvation. When the scheduling algorithm A considered for the analysis is optimal, a schedulability test reduces to a feasibility test, a stronger condition guaranteeing that any feasible task set can be scheduled by A. A task set is referred to as schedulable according to a given scheduling algorithm if all of its jobs are schedulable. A jobset ω is feasible if and only if it is *time-feasible* (i.e., timing constraints are met for all tasks without considering energy constraints) and *energy-feasible* (i.e., energy constraints are satisfied for all tasks without considering timing constraints).

A. Terminology and Notation

We first recall basic definitions for real-time scheduling concepts introduced in [1] used for checking the schedulability of a jobset under ED-H.

Definition 1. The processor demand of jobset ω on the time interval $[t_1, t_2)$ is

$$h_\omega(t_1,t_2) = \sum_{t_1 \le r_k, d_k \le t_2} C_k \tag{1}$$

When the processor utilization factor associated to a given jobset ω is less than 100%, the processor is idle from time to time, hence the notion of *slack time*.

Definition 2. The static slack time of jobset ω on the time interval $[t_1, t_2)$ is

$$SST_\omega(t_1,t_2) = t_2 - t_1 - h_\omega(t_1,t_2) \tag{2}$$

Definition 3. The static slack time of jobset ω is

$$SST_\omega = \min_{0 \le t_1 < t_2 \le d_{Max}} SST_\omega(t_1,t_2) \tag{3}$$

Similarly, in order to analyse the the schedulability of a jobset from the energy point of view, we recall hereafter how *the energy demand* and *the slack energy* are defined.

Definition 4. The energy demand of jobset ω on the time interval $[t_1, t_2)$ is

$$g_\omega(t_1,t_2) = \sum_{t_1 \le r_k, d_k \le t_2} E_k \tag{4}$$

Definition 5. The static slack energy of jobset ω on the time interval $[t_1, t_2)$ is

$$SSE_\omega(t_1,t_2) = C + E_p(t_1,t_2) - g_\omega(t_1,t_2) \tag{5}$$

Definition 6. The static slack energy of jobset ω is

$$SSE_\omega = \min_{0 \le t_1 < t_2 \le d_{Max}} SSE_\omega(t_1,t_2) \tag{6}$$

B. ED-H feasibility test

Based on the new concepts introduced in Section V-A, the author in [1] proposed an exact (i.e. necessary and sufficient) schedulability test for ED-H.

Theorem 1. [1] A jobset ω is schedulable by ED-H if and only if $SST_\omega \geq 0$ *and* $SSE_\omega \geq 0$.

As ED-H is an optimal scheduler, Theorem 1 provides us a feasibility condition.

Let us see now to what extent this test can be applied to a periodic task set sharing resources managed by DPCP. In particular, we will show that the exact ED-H schedulability test is no longer applicable because of the presence of additional blocking times due to the concurrent accesses to shared resources performed by the tasks.

Example 2. Let us illustrate by considering the periodic task set described in Table I. Assume that energy storage capacity is such that $C = 8$ and $E(0) = 8$. We assume that the incoming power P_p is constant over time and equals to 1. Off-line feasibility checking of the task set indicates that the task set is feasible, as Theorem 1 is satisfied. Let us consider now that tasks share semaphores S_k. Table I also indicates both the duration of the critical sections and their starting time w.r.t the job release time. A zero value means that task τ_i uses semaphore S_k as soon as it is released. The ceiling priority attached to a semaphore is defined by the highest priority task that uses the semaphore. Here, τ_1 starts its execution and locks (resp. unlocks) the semaphore S_1 at time $t = 1$ (resp. $t = 3$). τ_2 locks the resource at $t = 4$ and unlocks it right after at $t = 5$. Note that in the time interval $[9, 14)$, task τ_1 is released; it preempts τ_3 and then requests to lock semaphore S_1. As S_1 has already been locked by τ_3, the ceiling priority is increased to 1 and τ_3 continues using S_1 until $t = 13$. After, τ_1 locks S_1 and we can note that there is no time left to complete the execution of τ_1 before its deadline. Clearly Fig. 4, shows that task τ_1 misses its deadline at time $t = 14$.

TABLE I
TASK SET PARAMETERS FOR EXAMPLE 2

Task	ϕ_i	C_i	D_i	T_i	E_i	S_1	S_2	$a_{i,k}$	
								S_1	S_2
τ_1	0	3	5	9	3	2	-	1	-
τ_2	0	3	6	12	6	-	2	-	0
τ_3	0	9	27	36	9	4	3	2	6

It clearly appears that blocking times should be accounted for when designing schedulability tests for real-time energy-harvesting systems having resource access constraints. It has been proved in [2], that the worst-case blocking situation for every task should be considered for the derivation of a new consistent schedulability condition.

VI. SCHEDULABILITY ANALYSIS UNDER ED-H–DPCP

We prove that a jobset is schedulable under ED-H-DPCP if the blocking times on resources added to the processor demand on any interval does not exceed the length of the interval and if the blocking energy added to the energy demand on any interval does not exceed the maximum energy available in the interval.

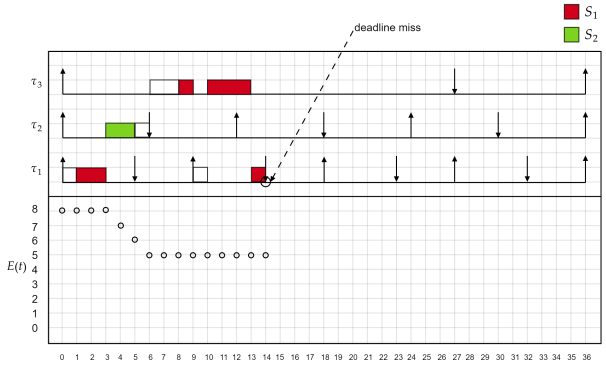

Fig. 4. Non-valid schedule under ED-H–DPCP

A. Blocking Time Computation

The worst-case blocking delay can be treated as an extra computation time for each job in addition to its normal computation time. Using DPCP, the authors in [2] showed that chained blocking is impossible. More precisely, they proved that the worst-case blocking time B_i of any job is bounded by the duration of the longest critical section among lower-priority jobs that may access the same resources.

Definition 7. The worst-case blocking time of all the jobs released by task τ_i is

$$B_i = \max_{i,k} \{\delta_{i,k} | S_k \in Z_i\} \tag{7}$$

with $\delta_{i,k}$ the set of lengths among the longest critical sections guarded by a semaphore S_k that can block τ_i.

Similarly to the concept introduced by Baruah in [20], let us define a *blocking function* $b(t_1, t_2)$ that represents the maximum blocking time experienced by a jobset on any given time interval $[t_1, t_2)$.

Definition 8. The maximum blocking time of jobset ω on the time interval $[t_1, t_2)$ is

$$b_\omega(t_1, t_2) = \sum_{\tau_i} \sum_{\substack{J_k \\ 0 < r_k \leq t_1, d_k \leq t_2}} B_i \tag{8}$$

where B_i is the worst-case blocking time experienced by the jobs as computed in equation 7.

B. Blocking Energy Computation

Blocking times can lead not only to timing violations for a given task τ_i but also to an energy starvation, the available energy being consumed by tasks that can block τ_i. The worst-case blocking energy BE_i of any task τ_i, is bounded by the energy consumption of the longest critical section among lower-priority jobs that may access the same resources.

Definition 9. The worst-case blocking energy of all the jobs released by task τ_i is

$$BE_i = \max_{i,k} \{\varepsilon_{i,k} | S_k \in Z_i\} \quad (9)$$

with $\varepsilon_{i,k}$ the set of energy consumption among the longest critical sections guarded by a semaphore S_k that can block τ_i.

Let us define a *blocking energy function* $be(t_1,t_2)$ that represents the maximum blocking energy experienced by a jobset on any given time interval $[t_1,t_2)$.

Definition 10. The maximum blocking energy of jobset ω on the time interval $[t_1,t_2)$ is

$$be_\omega(t_1,t_2) = \sum_{\tau_i} \sum_{\substack{J_k \\ 0<r_k\leq t_1, d_k \leq t_2}} BE_i \quad (10)$$

where BE_i is the worst-case blocking energy experienced by the jobs as computed in equation 9.

C. ED-H–DPCP Schedulability Test

We must check if both the static slack time and the static slack energy in any interval are no less than zero, even in the presence of blocking times. For that purpose, we first extend the definition of the static slack time on any given time interval.

Definition 11. For a jobset ω with resource access managed by DPCP, the static slack time on the time interval $[t_1,t_2)$ is

$$SSTR_\omega(t_1,t_2) = t_2 - t_1 - h_\omega(t_1,t_2) - b_\omega(t_1,t_2) \quad (11)$$

Note that $SSTR_\omega(t_1,t_2)$ gives an upper bound on the surplus of processing time that could be made available within the interval $[t_1,t_2)$ after executing the jobs and considering their worst-case blocking times on shared resources.

Similarly, we extend the definition of the static slack time of a given jobset:

Definition 12. For a jobset ω with resource access managed by DPCP, the static slack time is

$$SSTR_\omega = \min_{0\leq t_1<t_2\leq d_{Max}} SSTR_\omega(t_1,t_2) \quad (12)$$

Equation 12 will have to be checked so as to evaluate if the system does not exhibit a timing overflow in any time interval during the whole lifetime of the application. It intuitively appears that this time schedulability test will need to be evaluated for every time interval that starts with a release time and ends with a task deadline. Let n be the number of tasks. The computational complexity is clearly in $O(n^2)$.

We are now concerned with the energy schedulability issue. In other terms, *how to guarantee that a given task set will be schedulable regarding the energy production, the storage capacity, and its consumption of energy?*

Definition 13. For a jobset ω with resource sharing managed by DPCP, the static slack energy on the time interval $[t_1,t_2)$ is

$$SSER_\omega(t_1,t_2) = C + E_p(t_1,t_2) - g_\omega(t_1,t_2) - be_\omega(t_1,t2) \quad (13)$$

TABLE II
TASK SET PARAMETERS FOR EXAMPLE 3

Task	ϕ_i	C_i	D_i	T_i	E_i	S_1	S_2	$a_{i,k}$	
								S_1	S_2
τ_1	0	3	8	12	6	2	-	1	-
τ_2	0	1	10	16	2	2	-	1	-
τ_3	0	12	39	48	12	3	3	7	0

$SSER_\omega(t_1,t_2)$ gives the maximum energy that could be made available within the interval after executing the jobs, including their worst-case blocking times.

Definition 14. For a jobset ω with resource sharing managed by DPCP, the static slack energy is

$$SSER_\omega = \min_{0\leq t_1<t_2\leq d_{Max}} SSER_\omega(t_1,t_2) \quad (14)$$

$SSER_\omega$ gives the maximum amount of energy that could be consumed (i.e., remaining energy) from any time instant still satisfying the timing and energy constraints of all jobs.

We present here the schedulability test for guaranteeing that, given the capacity of the energy storage unit and the incoming power from the ambient source, a given periodic task set τ scheduled by ED-H with shared resources managed by DPCP, will meet both its timing and energy requirements (i.e., no deadline violations nor energy starvation will occur).

Theorem 2. A jobset ω with resource access managed by DPCP, is schedulable by ED-H if the following conditions are satisfied:

$$SSTR_\omega \geq 0 \; and \; SSER_\omega \geq 0 \quad (15)$$

Proof. The proof is identical to that of Theorem 1. [See [1]]. □

Remark. The blocking function $b_\omega(t_1,t_2)$ (resp. $be_\omega(t_1,t_2)$) used in the computation of $SSTR_\omega$ (resp. $SSER_\omega$) represents the worst-case conditions and hence the necessary and sufficient conditions of Theorem 1 becomes a sufficient condition in Theorem 2.

Note, the schedulability test cannot be performed offline if the energy cannot be predicted. In that case, the schedulability test is done online on sliding windows determined by the prediction technique. On the contrary if the power is constant we may have an offline schedulability test that consists in taking all the jobs of the first hyperperiod.

D. Illustrative example

Let us consider an illustrative example so as to explain the interest of the new schedulability test shown in Theorem 2.

Example 3. We consider the task set with resource parameters presented in Table II. We assume that at initialization time, the energy level is maximum (i.e., $E(0) = C = 7$) and that the instantaneous power rate is constant with $Pp = 1$. Tasks τ_1 and τ_2 can be blocked by τ_3 for at most 3 units of time. Based on Equation 7, tasks' blocking times are computed as follows: $B_1 = B_2 = 3$ and $B_3 = 0$. As

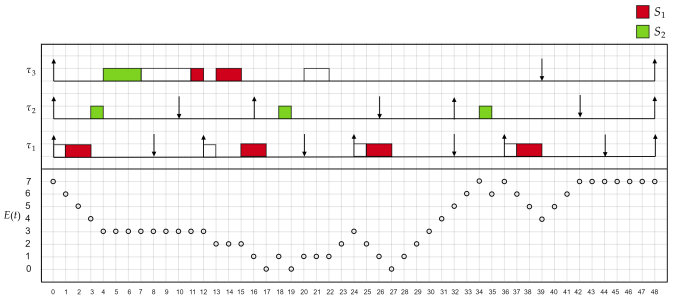

Fig. 5. Valid schedule under ED-H–DPCP

tasks share resources, we check the sufficient condition given in Theorem 2. Applying Equation 12 and Equation 14, we can verify that $SSTR_\omega \geq 0$ and $SSER_\omega \geq 0$. Therefore, we conclude that the task set is schedulable under ED-H–DPCP. The resulting schedule obtained with ED-H with shared resources managed by DPCP is depicted in Fig. 5. The highest priority task τ_1 begins its execution and locks semaphore S_1 at time $t = 1$ for 2 units. Then, task τ_2 starts its execution locking semaphore S_2. At time $t = 4$, task τ_3 executes locking first S_2 (unlocked 3 units of time later) and then S_1 at time $t = 11$. At time $t = 12$, the second job of task τ_1 is released and since τ_1 has a higher priority than τ_3, it preempts the execution of τ_3. Then, τ_1 requests semaphore S_1 but it must wait for its release by τ_3 at time $t = 15$. Access to semaphore S_1 is then granted to task τ_1 which successfully completes within its deadline with enough available energy. After the execution of the second job of task τ_1, the energy in the storage capacity is empty. Consequently, the processor state is left idle to recharge the energy reservoir. Note that, at the end of the hyperperiod, the reservoir is again fully replenished, thus ensuring that the schedule will also be valid on the next hyperperiods.

VII. Summary and Future Works

Energy harvesting is a promising technology that is a good alternative to enhance the lifetime of battery-operated IoT devices. Scheduling real-time tasks on such energy-constrained systems is a complex problem and ensuring the timing and energy requirements is very challenging. The optimal ED-H scheduling algorithm provides a framework for scheduling periodic tasks with both real-time and energy constraints. Originally designed for independent real-time tasks, we showed that the feasibility test associated to ED-H is not applicable anymore in the presence of shared resources. Resource sharing can give rise to blocking times, thus delaying the completion of tasks due to resource contention. DPCP is an interesting and efficient protocol to manage shared resources because it prevents priority inversions and chained blocking while being deadlock-free. In this paper, we studied the properties of real-time energy-harvesting systems scheduled using ED-H scheduling algorithm, when used in conjunction with DPCP for arbitrating access to shared resources. We have derived both worst-case blocking times and blocking energy for tasks. We performed a schedulabitity analysis for guaranteeing off-line that both timing and energy constraints will be satisfied.

Taken together, ED-H and DPCP, offer an efficient scheduling framework. We plan to implement it on a real mobile robot platform (the HUSKY cobot [21]) in order to have guaranteed time and energy-driven schedule. Xenomai [22] is the real-time executive support chosen for ensuring the hard real-time constraints of the embedded robotic application. Consequently, our immediate future works include the integration of the ED-H–DPCP scheduling support into Xenomai core.

References

[1] 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.

[2] M. I. Chen and K. J. Lin, "Dynamic priority ceilings: A concurrency control protocol for real-time systems," *Real-Time Systems*, vol. 2, no. 4, pp. 325–346, 1990.

[3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, pp. 46–61, 1973.

[4] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "Dynamic voltage scaled microprocessor system," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, 2000.

[5] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, *System-level design techniques for energy-efficient embedded systems*. Springer US, 2005.

[6] J. J. Chen and C. F. Kuo, "Energy-efficient scheduling for real-time systems on dvs platforms," *RTCSA*, pp. 28–35, 2007.

[7] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[8] T. P. Baker, "A stack-based resource allocation policy for realtime processes," *Real-Time Systems Symposium*, pp. 191–200, 1990.

[9] A. K.-L. Mok, "Fundamental design problems of distributed systems for the hard-real-time environment," Ph.D. dissertation, 1983.

[10] K. wa Lam, S. H. Son, and S. lun Hung, "Priority ceiling protocol with dynamic adjustment of serialization order," *Proceedings - International Conference on Data Engineering*, pp. 552–561, 1997.

[11] P. Martineau and M. Silly, "Scheduling in a hard real-time system with shared resources," *Euromicro Conference on Real-Time Systems*, pp. 234–239, 1994.

[12] J. Y. Kim and K. Koh, "Pessimistic deadline ceiling protocol: A concurrency control protocol under earliest deadline first scheduling," *Euromicro Conference on Real-Time Systems*, pp. 80–86, 1995.

[13] J. Wu and K. L. Kao, "Energy-Efficient Scheduling of Real-time Tasks with Abortable Critical Sections," *IEEE International Conference on Embedded Software and Systems*, pp. 1788–1793, 2012.

[14] J. Wu, "Energy-efficient scheduling of real-time tasks with shared resources," *Future Generation Computer Systems*, pp. 179–191, 2016.

[15] F. Zhang and S. T. Chanson, "Processor voltage scheduling for real-time tasks with non-preemptible sections," *Proceedings - Real-Time Systems Symposium*, pp. 235–245, 2002.

[16] R. Jejurikar and R. Gupta, "Energy aware non-preemptive scheduling for hard real-time systems," *Proceedings - Euromicro Conference on Real-Time Systems*, vol. 2005, pp. 21–30, 2005.

[17] P. Wägemann, T. Distler, H. Janker, P. Raffeck, and V. Sieh, "A Kernel for Energy-Neutral Real-Time Systems with Mixed Criticalities," *RTAS - Proceedings*, 2016.

[18] G. C. Buttazzo, *Hard Real-Time Computing Systems*, ser. Real-Time Systems Series. Boston, MA: Springer US, 2011, vol. 24.

[19] M. Chetto and A. Queudet, "Clairvoyance and online scheduling in real-time energy harvesting systems," *Real-Time Systems*, vol. 50, no. 2, pp. 179–184, 2014.

[20] S. K. Baruah, "Resource sharing in EDF-scheduled systems: A closer look," *Proceedings - Real-Time Systems Symposium*, pp. 379–387, 2006.

[21] M. I. M. Abdulla, M. Chetto, A. Queudet, and L. Belouaer, "On designing cyber-physical-social systems with energy-neutrality and real-time capabilities," *ICPS*, pp. 369–374, 2021.

[22] Xenomai 4. [Online]. Available: https://evlproject.org/