An Energy Management Framework for Energy Harvesting Embedded Systems

CLEMENS MOSER, JIAN-JIA CHEN, and LOTHAR THIELE
Swiss Federal Institute of Technology (ETH) Zürich

Energy harvesting (also known as energy scavenging) is the process of generating electrical energy from environmental energy sources. There exists a variety of different energy sources such as solar energy, kinetic energy, or thermal energy. In recent years, this term has been frequently applied in the context of small autonomous devices such as wireless sensor nodes. In this article, a framework for energy management in energy harvesting embedded systems is presented. As a possible scenario, we focus on wireless sensor nodes that are powered by solar cells. We demonstrate that classical power management solutions have to be reconceived and/or new problems arise if perpetual operation of the system is required. In particular, we provide a set of algorithms and methods for various application scenarios, including real-time scheduling, application rate control, as well as reward maximization. The goal is to optimize the performance of the application subject to given energy constraints. Our methods optimize the system performance which, for example, allows the usage of smaller solar cells and smaller batteries. Furthermore, we show how to dimension important system parameters like the minimum battery capacity or a sufficient prediction horizon. Our theoretical results are supported by simulations using long-term measurements of solar energy in an outdoor environment. In contrast to previous works, we present a formal framework which is able to capture the performance, the parameters, and the energy model of various energy harvesting systems. We combine different viewpoints, include corresponding simulation results, and provide a thorough discussion of implementation aspects.

Categories and Subject Descriptors: D.4.1 [**Operating Systems**]: Process Management—*Scheduling*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Power management, embedded systems, energy harvesting, model predictive control, real-time scheduling, reward maximization

This work was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. This research has also been supported by the European Network of Excellence on Embedded System Design ARTISTDesign.
Authors' addresses: C. Moser, J.-J. Chen, and L. Thiele, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland; email: {cmoser, jchen, thiele}@tik.ee.ethz.ch.

DOI 10.1145/1773814.1773818 http://doi.acm.org/10.1145/1773814.1773818

ACM Reference Format:
Moser, C., Chen, J.-J., and Thiele, L. 2010. An energy management framework for energy harvesting embedded systems. ACM J. Emerg. Technol. Comput. Syst. 6, 2, Article 7 (June 2010), 21 pages. DOI = 10.1145/1773814.1773818 http://doi.acm.org/10.1145/1773814.1773818

1. INTRODUCTION

Power and energy management has played an important role in modern embedded system design to prolong the battery lifetime without sacrificing too much performance. The emerging technology of energy harvesting has earned much interest recently to provide a means for sustainable embedded systems. For systems with expensive deployment cost, energy harvested from the environment could provide sustainable services for data gathering applications. Among renewable energy resources, energy harvesting with solar panels is one of the most popular applied technologies, and there have been many energy harvesting circuits that are designed to efficiently convert and store solar energy.

Clearly, one may just use solar energy to recharge a primary energy source, for example, a battery. In this way, the point in time when the system runs out of energy is simply postponed. If, however, one strives for perpetual operation, common power management techniques have to be reconceived. Then the embedded system has to adapt to the stochastic nature of the solar energy. The goal of this adaptation is to maximize the utility of the application in a long-term perspective. The resulting mode of operation is sometimes also called an energy neutral operation: The performance of the application is not predetermined a priori, but adjusted in a best effort manner during runtime and ultimately dictated by the power source.

Efficient solar harvesting systems adapt the electrical operating point of the solar cell to the given light condition, using techniques called Maximum Power Point Tracking (MPPT). For industrial, large-scale solar panels these techniques are well-understood and extensively used [Esram and Chapman 2007]. For solar cells the size of a few cm^2, however, particular care has to be taken in order not to waste the few mW generated by the solar cell.

Concerning the software algorithms running on an embedded system, similar considerations as for the hardware hold. The energy required for sophisticated control algorithms may introduce a high control overhead for low-power applications. For example, a simple sensor node, which periodically senses and transmits data, might spend most of the time in power-saving sleep modes, and, hence, we need simple and low-complexity solutions. Two of the first prototype sensor nodes with energy harvesting capabilities were Heliomote [Hsu et al. 2005] and Prometheus [Jiang et al. 2005].

A first step in this direction has been made in Kansal et al. [2007]. The authors point out how the problem of adapting the duty cycle of a solar powered sensor can be modeled by a linear program. As the objective, the average duty cycle shall be optimized. Instead of periodically solving this linear program online, a heuristic algorithm of reduced complexity is proposed. The work in Vigorito et al. [2007] improves on the results in Kansal et al. [2007]; however, the assumed optimization objective and application remain very specific.

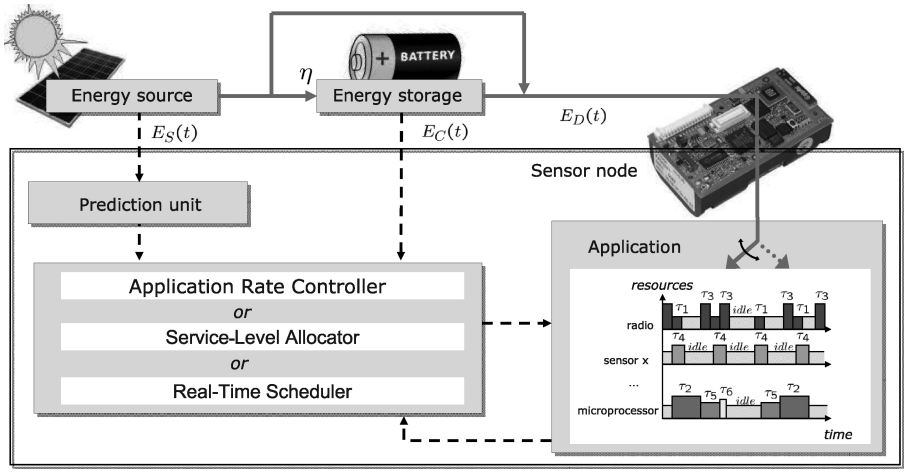


Fig. 1. Illustration of the system model.

In this article, a more general approach to optimize the utilization of solar energy will be presented which builds on our works in Moser et al. [2007a, 2008, 2009a, 2009b].

In contrast to previous work, the methods presented in this article are capable of modeling a much larger variety of application scenarios, constraints, and optimization objectives. To optimally adjust the application rates according to the environmental energy, we demonstrate how multiparametric programming techniques can be applied to compute efficient online controllers. In doing so, we avoid solving optimization problems online which may be prohibitively complex in terms of running time and energy consumption. Instead, a precomputation of the control laws is performed according to the results in Moser et al. [2010]. Furthermore, we investigate the fundamental problem of assigning harvested energy to services, which have user-defined rewards. In order to optimize the overall reward, we present algorithms which carefully plan the future energy consumption of the system. Finally, we address the case of systems that execute applications with real-time constraints. For this purpose, we have constructed an optimal scheduling algorithm that jointly handles constraints from both the energy and time domains.

We organize this article as follows. In the next section, the system concept and the models and methods used are discussed. In Section 3, we demonstrate how to obtain simple but optimal controllers which adjust the application rates on a sensor node. In Section 4, we present algorithms to assign service levels with user-defined rewards. In Section 5, we present lazy scheduling algorithms for optimal online scheduling of real-time applications. Section 6 provides a summary and the conclusions.

2. SYSTEM MODEL AND PROBLEM STATEMENT

The system model is shown in Figure 1. The whole hardware/software system is powered by an energy harvesting device (e.g., a solar cell) that delivers energy

$E_S(t)$ and loads the energy storage. The sensor node may use the energy $E_S(t)$ directly to drive the application with energy $E_D(t)$. Surplus energy is stored in a storage device with efficiency η. At time t, there is the stored energy $E_C(t)$ available. Note that the subscript C in $E_C(t)$ refers to capacity. The capability to bypass the storage device is a typical feature of the latest prototypes. It offers the opportunity to save substantial energy by using the solar energy directly when available. The energy storage may be charged up to its capacity C. If the application consumes no energy $E_D(t)$ and the storage is consecutively replenished, an energy overflow occurs and surplus energy is wasted.

Note that the presented model of the power flow even holds for the simplest connection between the energy source, the energy storage, and the load: a direct, parallel connection. If, for example, a solar cell, a supercapacitor, and a sensor node are connected in parallel, they will experience the same voltage and, due to Kirchhoff's law, the currents will split up at the junction. As a consequence, the power flow (as a product of voltage and current) will show the described behavior. For a detailed discussion on more sophisticated charging circuitry, we refer the reader to Brunelli et al. [2009] and Moser et al. [2010].

As illustrated in Figure 1, this article investigates three different application scenarios, namely application rate control, service-level allocation, as well as real-time scheduling. We provide dedicated algorithms for these application scenarios, which will be presented in Section 3, 4, and 5, respectively. For all application scenarios, an estimation of the future energy harvesting is required to optimize the system performance. As illustrated in Figure 1, the estimation of the prediction unit is used as an input to the online scheduler which controls the application. In addition, the currently stored energy in the energy storage is measured.

Both the application rate controller and the service-level allocator decide on the usage of harvested energy in a long-term perspective. In doing this, parameters of the application are adapted for the coming days or even weeks. As a result, the average power consumption of the system is optimized. While the application rate controller can be designed for a wide variety of system dynamics and constraints, the major advantage of the service-level controller is the explicit formulation of user-defined *rewards* for the different services. Furthermore, the service-level controller can handle a finite set of service levels, which can only be done approximately with the application rate controller.

The real-time task schedulers presented in Section 5 guarantee optimal task ordering in a short-term perspective. For instance, for time scales of milliseconds or seconds, the scheduler decides how to assign energy to time-critical tasks. Application parameters as well as average power consumption are not influenced by the real-time task scheduler.

In summary, the application rate controller or service-level allocator decides which and how many tasks are executed on the long run and the real-time task scheduler decides about the short-term task ordering. All three application models and corresponding solution methods turn out to be of practical concern. They can be applied independently as presented in the respective sections. Furthermore, it is possible to combine the proposed application models. For a

discussion on how to combine the different models and methods, the reader is referred to Section 6. The problem statement can be written as follows.

—We want to adapt parameters of the application such that a maximal utility is obtained while respecting the limited and time-varying amount of energy.

—In the presence of real-time constraints, if the task set of an application is schedulable with the available energy and time, we want to determine a feasible schedule.

3. APPLICATION RATE CONTROLLER

In this section we are adapting parameters of the application to optimize the performance in a long-term perspective. By adapting the activation rates of tasks, we control the average power consumption of the embedded system. As a main contribution, a framework for adaptive power management is proposed which builds on multiparametric programming techniques. In doing so, we link methods from control theory with the software design of environmentally powered systems.

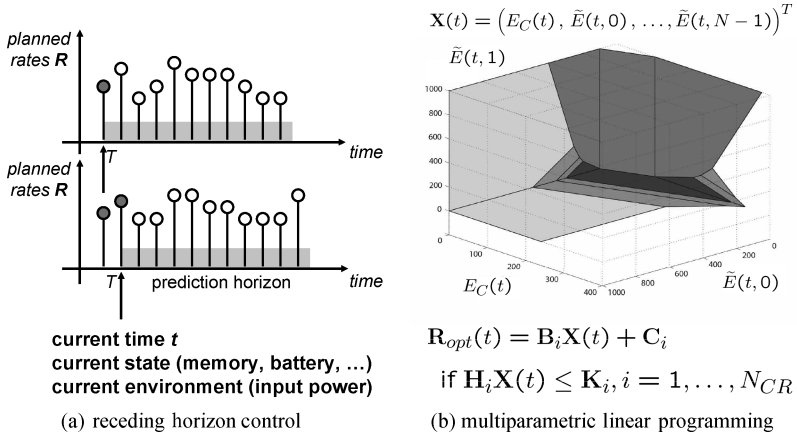
3.1 Task Model

The energy $E_D(t)$ is used to execute tasks on various system components. A task τ_i from the set of tasks needs energy e_i for completing a single instance. We suppose that a task is activated with a time-variant rate $R_i(t)$, that is, during the basic time interval T, the task is executed $R_i(t)$ times. Therefore, a task needs energy $e_i \cdot R_i(t)$ in time interval T for successful execution. Finally, we denote $\mathbf{R}(t)$ the vector of all task rates $R_i(t)$ at time t. For more sophisticated task and application models, the reader is referred to Moser et al. [2008b].

3.2 Energy Prediction and Receding Horizon Control

The estimation unit receives tuples $(t, E_S(t))$ for all times and delivers N predictions on the energy production of the energy source over the so-called prediction horizon (see Figure 2(a)). In the following, the energy prediction will be denoted $\widetilde{E}(t, 0), \widetilde{E}(t, 1), \ldots, \widetilde{E}(t, N-1)$. The estimation algorithm should depend on the type of the energy source and the system environment. Standard techniques from automatic control and signal processing can be applied here. For sensors with solar cells deployed in an outdoor environment, even the weather forecast can be used. In Kansal et al. [2007] and Moser et al. [2007b], prediction algorithms based on a weighted sum of historical data are demonstrated to perform well. In addition, a short-term factor is accounting for the momentarily harvested energy. In Moser et al. [2008b], a worst-case energy prediction algorithm which guarantees sustainable operation is presented. For a discussion about suitable energy prediction algorithms or how to handle prediction mistakes, we refer the reader to related work. The mentioned degradation effects due to energy misprediction have been studied extensively elsewhere.

At time t, the controller is computing the control sequence $\mathbf{R}(t + k \cdot T)$ for all prediction intervals $0 \leq k < N$ based on the energy estimation as well as the current system state (e.g., $E_C(t)$). However, only the first



(a) receding horizon control (b) multiparametric linear programming

Fig. 2. Application rate controller for an example application.

control rates $\mathbf{R}(t)$ are applied to the system during the first time-step T. The rest of the control sequence is discarded. At time $t + T$, a new vector $\mathbf{R}(t)$ is computed which extends the validity of the previous vector by one time-step. Again only the first control is used, yielding a Receding Horizon Control (RHC) strategy.

3.3 Multiparametric Control Design

It has been shown that many optimization problems for energy harvesting systems can be modeled by the class of Linear Programs (LP), see, for example, Moser et al. [2007b] and Kansal et al. [2007]. However, solving at each time-step t an LP in a resource constrained system like a sensor node is prohibitive in general. In the following, we will describe a method for solving the respective LP offline and using the result for constructing an optimal online controller.

In Moser et al. [2007b], the fundamental observation made is that the considered LPs are parameterized and can be solved offline. As illustrated in Figure 2(b), a state vector $\mathbf{X}$ is defined consisting of the actual system state, the level of the energy storage, as well as the estimation of the incoming energy over the finite prediction horizon. Furthermore, let us denote the vector of optimal control rates $\mathbf{R}_{opt}$. As it turns out, the state space of vector $\mathbf{X}$ can be subdivided into a number N_{CR} of critical regions, in which the optimal control rates can be obtained by evaluating a simple linear function of $\mathbf{X}$. The polyhedral partition of the state space is illustrated in Figure 2(b) in a 3D cut using different colors for the different regions. The computation of the vectors and matrices in the control law in Figure 2(b) is done offline using, for example, the algorithm in Borrelli et al. [2003].

In the online case, the controller has to identify to which region i the current state vector $\mathbf{X}$ belongs. After this simple membership test, the optimal control rates $\mathbf{R}_{opt}$ for the next time-step are computed by evaluating a linear function of $\mathbf{X}$. These rates $\mathbf{R}_{opt}$ are identical to the rates one would obtain by solving the linear program. However, the computational demand is greatly reduced

compared to solving an LP online. After having solved the mp-LP in advance, a set of N_{CR} polyhedra with associated control laws has to be stored and evaluated at each time-step t.

3.4 Experimental Evaluation

In this section, both feasibility and practical relevance of the proposed approach are demonstrated by means of simulation and measurements. For this purpose, we implemented the computation of online controllers for an exemplary case study using the Matlab toolbox in Kvasnica et al. [2004]. We simulated the behavior of the controlled system and measured the resulting controller overhead on a sensor node.

Measurements of solar light intensity during nearly 5 years recorded at [sol 2007] serve as energy input $E_S(t)$. The time interval between two samples is 5 minutes, so we set the basic time interval $T = 5$ min. Using this data, we could extensively test the performance of our algorithms for time scales one usually wants to achieve with solar powered sensor networks.

Let us consider the following example application. Let us assume a sensor node is expected to observe some phenomenon of interest in an environmental monitoring application. For this purpose, an image has to be recorded with a camera sensor and the data has to be transmitted to a base station. We can model these requirements using a data sensing task τ_1 and a data transmission task τ_2. The data sensing task τ_1 is operated with rate $R_1(t)$. At every instantiation, the sensing task τ_1 drains $e_1 = 0.1$ energy units and stores an image in some local memory. The transmission task τ_2 is transmitting images with a rate $R_2(t)$. Task τ_2 reduces the occupied memory by one image per instantiation. In general, radio communication is the main energy consumer in sensor networks and we choose $e_2 = 0.9$. According to our experience with hardware prototype systems, we opted for a storage efficiency of $\eta = 0.8$. The sensor node can store a maximum of $M_{max} = 1000$ images. The energy prediction algorithm we used is the same as in Moser et al. [2007b] with parameter $\alpha = 0.5$.

The optimization objective is to maximize the minimal rate with which the task τ_1 is operated in the finite horizon. In terms of intervals, the objective translates into a minimization of the maximum interval between any two consecutive measurements. This could be a reasonable objective if one attempts to minimize the unobserved time periods between two recorded images.

In Figure 3, the generated controller is optimizing the sampling rate R_1 and the transmission rate R_2 during 5 days. The controller achieves to optimize the rate R_1 in spite of the unstable power supply $E_S(t)$. Consequently, the stored energy $E_C(t)$ is highly varying: It is increasing during day and decreasing at night. Also the transmission rate R_2 is oscillating around the sampling rate R_1. Since it is favorable to use energy when it is available, data is stored at night and transmitted during day.

In order to evaluate the implementation overhead, we implemented the controller consisting of $N_{CR} = 1049$ critical regions on a BTnode [Beutel et al. 2004] and measured its running time as well as power consumption. In terms of physical memory, the storage demand of the controller amounts to 9,4 Kbyte using

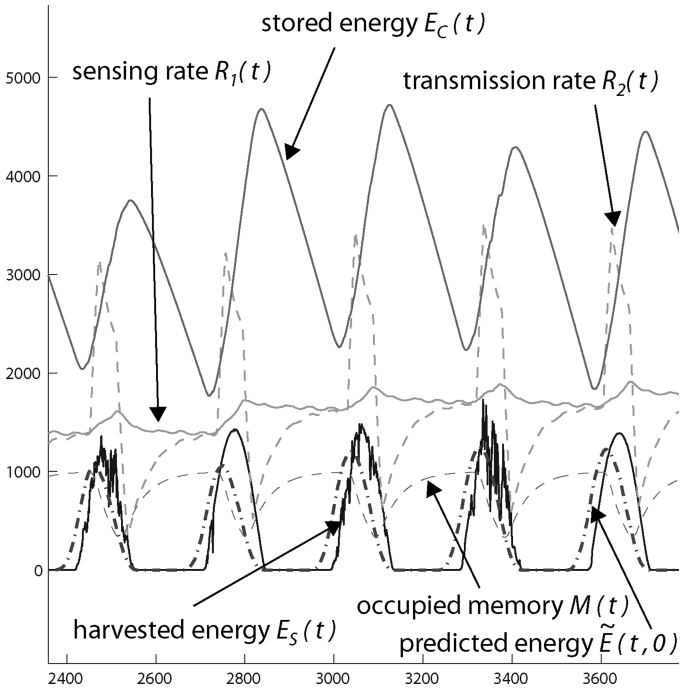


Fig. 3. Adaption of the sensing rate r_1 and the data transmission rate r_2 by a multiparametric linear controller for five days.

the techniques described in Moser et al. [2008b] and a 16-bit integer representation per coefficient. In other words, the algorithm is applicable to commonly used sensor nodes like the TmoteSky [Corporation 2006], which exhibits 48 Kbyte Flash ROM or the BTnode, with 128 Kbyte. In our implementation on a BTnode, we measured the worst-case computation time <190 ms, yielding a neglectable control overhead every $T = 5$ min. Also the energy consumption for evaluating the control law can be neglected compared to the energy consumption caused by the application rates R_1 and R_2 during the interval T.

4. SERVICE-LEVEL ALLOCATOR

In this section, we assume that a service executed by the system is associated with a certain reward which can be defined by the user. We present algorithms which maximize the overall reward of the task subject to the given energy constraints. We will investigate the case of continuously adaptable service levels with concave reward functions in Section 4.2.1 and discrete service levels in Section 4.2.2.

For real-time systems without energy harvesting, reward maximization under energy constraints has been studied extensively as, for example, in Chen and Kuo [2005], Alenawy and Aydin [2004], and Rusu et al. [2002, 2003]. In general, the reward associated with a service increases with the amount of computation required to provide the service. Prominent models used in literature are the imprecise computation model [Liu et al. 1991] and the Increasing

Reward with Increasing Service (IRIS) model [Dey et al. 1996; Shih et al. 1991]. For numerous practical tasks, such as image and speech processing, time-dependent planning, and multimedia applications, the reward function is modeled as a concave function of the amount of computation [Aydin et al. 1999].

4.1 Service Allocation Models

—*Energy Harvesting and Storage Models*. For service-level allocation, we assume that the energy harvesting device has a prediction unit to estimate the amounts of energy harvested in each of the next K prediction intervals in the future. For brevity, a prediction interval is denoted as a *frame*, while K refers to *number of frames of the prediction horizon*. Each frame is assumed to have the same length and is the basic time unit for scheduling. As in the previous section, the length of a frame in physical time and the number of frames K are chosen depending on the given energy source (see Section 4.5). The energy storage, for example, a supercapacitor or a battery, stores the energy harvested from the environment. To simplify the discussion, we assume a charging efficiency of $\eta = 1$ for this section. Note, however, that the whole formulation allows to handle also more general problems. The amount of the harvested energy that will be stored in the energy storage in the kth frame is denoted by $E_S(k)$. The energy storage is constrained by the maximum capacity C of the energy in the storage. If the energy storage is full, the additional harvested energy dissipates.

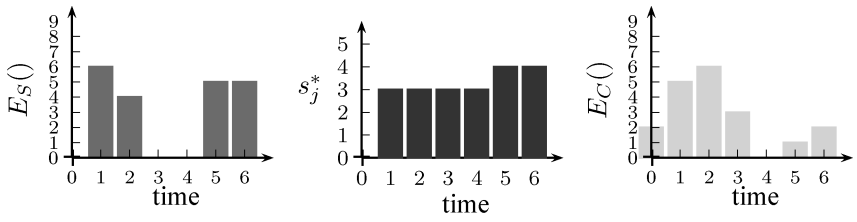
—*Service and Task Models*. We investigate how to achieve performance maximization for periodically executed services in environmentally powered systems. To this purpose, the scheduler receives predictions of the future energy $E_S(k)$ generated in the next $1 \leq k \leq K$ frames. We consider a task with either: (1) continuous service levels or (2) M discrete service levels. For every frame k, the scheduler has to assign exactly *one* service level s_k. For continuous service levels, a service level with s amount of energy consumption contributes $r(s)$ reward, in which we assume $r(s)$ is a concave and increasing function. Specifically, we assume that the reward $r(s)$ is strictly concave in s, that is,

$$\alpha \cdot r(\sigma_1) + (1-\alpha) \cdot r(\sigma_2) < r\left(\alpha \cdot \sigma_1 + (1-\alpha) \cdot \sigma_2\right),$$

for any service levels σ_1, σ_2 with $\sigma_1 \cdot \sigma_2 \geq 0$, $\sigma_1 \neq \sigma_2$, and $1 > \alpha > 0$. For high service levels the reward saturates, that is, the increment in reward is getting smaller. This property is typical for human perception of, for example, a multimedia application where at a certain point the difference in resolution becomes imperceptible. As already mentioned, this reward model has been used extensively in the literature.

For discrete service levels, the jth service level is associated with a corresponding reward r_j and an energy consumption e_j which is drawn from the energy storage. Note that for the discrete service levels, the user may define arbitrary rewards which are not required to be concave.

—*Problem Statement*. We are given the prediction for K frames at time t. Without loss of generality, we scale time t to 0. The energy in the energy



(a) $E_S()$ (b) the assignment (c) energy in the storage

Fig. 4. An example for service allocation.

storage at time 0 is specified as $E_C(0)$ and the energy harvested in the kth frame is $E_S(k)$ with charging efficiency $\eta = 1$. For brevity, the kth frame starts from time $k-1$ to time k. After the last frame, we require to have at least energy E_ℓ in the energy storage which can be used in the future. The objective is to find an assignment $\vec{s} = (s_1, s_2, \ldots, s_K)$ of service levels for these K frames such that the sum of rewards $\sum_{k=1}^{K} r_{s_k}$ (or $\sum_{k=1}^{K} r(s_k)$ for the continuous case) is maximized while respecting the required energy constraints. Note that we output $E_D(t) = e_{s_k}$ (respectively, $E_D(t) = s_k$) for the kth frame for the discrete (respectively, continuous) cases.

4.2 Service-Level Allocation Algorithms

4.2.1 *Service Allocation for Continuous Service Levels.* Because the service is a concave and increasing function of the energy consumption, an optimal assignment for the reward maximization on energy harvesting problem should consume a constant amount of energy to maximize the achieved reward. However, the harvested energy might not be enough to support the energy consumption. Therefore, an optimal assignment for the reward maximization on energy harvesting problem should try to consume some constant amounts of energy without leading to energy underflow. For example, as shown in Figure 4(a), if K is 6 with $E_\ell = E_C(0) = 2$, $E_S(1) = 6$, $E_S(2) = 4$, $E_S(3) = 0$, $E_S(4) = 0$, $E_S(5) = 5$, $E_S(6) = 5$. Running at a constant energy consumption with an assignment $\vec{s}$ with $\frac{6+4+5+5}{6} = \frac{10}{3}$ unit of energy consumption for all these six frames. However, the resulting assignment is not feasible since there is an energy underflow at the fourth frame. Figure 4(b) gives an optimal energy consumption assignment which tries to consume constant amounts of energy without incurring energy underflow.

If there is no constraint on the battery capacity, that is, C is $+\infty$, we just have to prevent energy underflows. The strategy is to find the bottleneck that incurs energy underflow. Suppose that $E_\ell(k)$ is the minimal energy leftover after frame k, where $E_\ell(K)$ is defined as E_ℓ and $E_\ell(k)$ is 0 for $k = 1, 2, 3, \ldots, K-1$. Let $\widetilde{s}_j$ be $\frac{E_C(0) - E_\ell(j) + \sum_{i=1}^{j} E_S(i)}{j}$, $\forall j = 1, 2, \ldots, K$. To optimize the reward, we have to find the index j^* such that $\widetilde{s}_{j^*} \leq \widetilde{s}_k, \forall 1 \leq k \leq K$. Then we assign the first j^* frames with the energy consumption equal to $\widetilde{s}_{j^*}$ and set $E_C(j^*)$ as 0. That is, s_j is set to $\widetilde{s}_{j^*}$ for $j \leq j^*$. For example, in Figure 4(b), j^* is 4. For the rest of the frames, we apply the same procedure by only considering the remaining frames from

j^* to K. Due to the concavity of reward functions, the aforesaid approach can be proved to derive optimal solutions.

For the case that there is a battery constraint, that is, C is not $+\infty$, the problem is more complicated. First, we derive an optimal solution $\vec{s}^*$ with infinite battery capacity as shown in the previous paragraph. The solution for infinite battery capacity divides the K frames into some segments, in which a segment $[k_{n-1}, k_n]$ is defined such that solution $\vec{s}^*$ has the same energy consumption for all the frames between $k_{n-1}+1$ to k_n. However, as it is possible that a segment has energy overflow, we will present how to revise the solution in a segment $[k_{n-1}, k_n]$. There are three cases for energy consumption.

—The energy consumption is a constant, denoted by λ_1, in the segment such that the energy in the battery at the end of the segment is $E_\ell(k_n)$.

—The constant energy consumption $\lambda_2(k)$ from frame k to frame k_n by assuming that the energy leftover in the battery after frame k is C. We take the frame k^* with the minimum $\lambda_2(k^*)$.

—The constant energy consumption $\lambda_3(k)$ from frame k_{n-1} to frame k by assuming that the energy leftover in the battery after frame k is 0. We take the frame k' with the minimum $\lambda_3(k')$.

Based on the preceding cases, we take the following actions.

—If $\lambda_2(k^*) < \lambda_1$, we divide the segment into two segments $[k_{n-1}, k^*]$ and $[k^*+1, k_n]$ such that the energy leftover $E_\ell(k^*)$ in frame k^* is specified to be C.

—If $\lambda_3(k') < \lambda_1$, we divide the segment into two segments $[k_{n-1}, k']$ and $[k'+1, k_n]$ such that the energy leftover $E_\ell(k')$ in frame k' is specified to be 0.

—Otherwise, there is no overflow or underflow, and we can return the energy assignment for the segment.

The previous algorithm is denoted as algorithm Recursive Decomposition (abbreviated as RD), which requires a recursive call by dividing a segment into two to decide the energy consumption assignment. The flow chart is presented in Figure 5. Moreover, it was proved in Moser et al. [2008a] that algorithm RD derives optimal solutions for reward maximization with continuous service levels in $O(K^2)$.

4.2.2 *Service Allocation for Discrete Service Levels.* For a task with discrete service levels, we present two algorithms. The first algorithm is a greedy algorithm extended from algorithm RD in Section 4.2.1. It derives discrete service levels by rounding the continuous service levels computed by algorithm RD. The second algorithm is an approximation algorithm with adjustable time complexity which is based on the principles of dynamic programming.

—*Greedy Algorithm*. Suppose that the assignment by applying algorithm RD is $\vec{s}^*$. The greedy algorithm for discrete service levels simply applies algorithm RD as a subroutine to determine the assignment for discrete service levels by rounding down the energy consumption. A naïve extension of algorithm RD

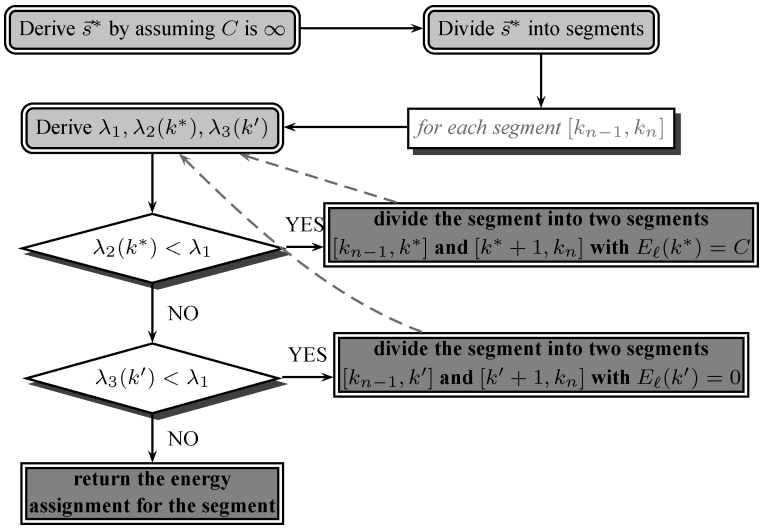


Fig. 5. Flowchart of algorithm RD.

is to choose service level $s_k^\dagger$ for the kth frame such that $e_{s_k^\dagger} \leq e_{s_k^*} < e_{s_k^\dagger+1}$. This guarantees the feasibility of the derived assignment $\vec{s}^\dagger$, but loses optimality. Therefore, to optimize the system reward, we need to be more careful. Some available discrete service levels might be energy inefficient. That is, service level j might consume much more energy but have marginal improvement on the reward. To conquer this, we have to eliminate those service levels j with $\frac{r_j - r_{j-1}}{e_j - e_{j-1}} < \frac{r_{j+1} - r_j}{e_{j+1} - e_j}$. As a result, after the elimination, for $2 \leq j \leq M' - 1$, we have $\frac{r_{g_j} - r_{g_{j-1}}}{e_{g_j} - e_{g_{j-1}}} \geq \frac{r_{g_{j+1}} - r_{g_j}}{e_{g_{j+1}} - e_{g_j}}$.

After eliminating the energy-inefficient service levels, the greedy algorithm iteratively decides the service level of a frame based on the solution derived from algorithm RD. That is, for the kth iteration, algorithm RD is applied to derive the service level of the kth frame by assuming that the service levels of the first $k-1$ frames have been determined, and the greedy algorithm greedily chooses the service level g_j with $e_{g_j} \leq e_{s_k^*} < e_{g_{j+1}}$ for the kth frame, where s_k^* is the service level derived from algorithm RD for the kth frame.

—*Dynamic Programming*. We now present a dynamic programming algorithm that derives optimal service levels in case the reward values are integers. For noninteger reward values, an approximation is provided. Let ϵ be a user-specified real number between 0 and 1 for the tolerance of approximated solutions. Specifically, by choosing a smaller ϵ, the approximated algorithm will derive a solution which is guaranteed to be closer to the optimal solution, but the complexity will be higher. Therefore, how to choose a proper ϵ to trade the performance with the running time is a task left to system designers (see also Section 4.6). Then, for each service level j, we derive the *rounded reward* r'_j of service level j by applying $r'_j = \lfloor \frac{r_j}{\epsilon \cdot r_M} \rfloor$.

Suppose that $\tilde{E}(k, \rho)$ is the maximum energy residual in the energy storage after servicing the kth frame with total rounded reward (for the first k frames) no less than ρ. For notational simplicity, let $\tilde{E}(k, \rho)$ be $-\infty$ if $\rho < 0$ or $\rho > kr'_M$. Moreover, by the definition of feasible assignments, the total reward of the first k frames must be at least kr'_1. Hence, when $\rho < kr'_1$

$$\tilde{E}(k, \rho) = -\infty. \tag{1}$$

Then, we can have the dynamic programming as follows.

$$\tilde{E}(k, \rho) = \min\left\{C, \max_{j=1,2,\ldots,M}\{\tilde{E}(k-1, \rho - r'_j) + E_S(k) - e_j\}\right\} \tag{2}$$

$$\tilde{E}(1, \rho) = \begin{cases} \min\left\{C, \hat{e}\right\}, & \text{if } \hat{e} = E_C(0) + E_S(1) - e_{j'_\rho} \geq 0 \\ -\infty, & \text{otherwise,} \end{cases} \tag{3}$$

where j'_ρ is the service level j with $r'_{j-1} \leq \rho \leq r'_j$, where r'_0 is assumed 0 here for calculating j'_ρ. Note that the other boundary conditions must also be applied for $\tilde{E}(k, \rho)$.

We can build a dynamic programming table by applying the preceding recursive function in Eqs. (1), (2), and (3). Suppose that R' is the maximum value with $\tilde{E}(K, R') \geq E_\ell$. By back-tracking the dynamic programming table, we can derive an assignment $\vec{s}^\sharp$ such that the sum of the rounded reward of $\vec{s}^\sharp$ is R' and the residual energy in the energy storage is no less than E_ℓ at the end of the Kth frame. The following theorem shows that the quality of the derived solution $\vec{s}^\sharp$ of the previous dynamic programming is not too far away from the optimal assignment, even in the worse case. In addition, the time complexity is significantly reduced compared to straightforward approaches like Integer-Linear Programming or a full search.

THEOREM 1. *Deriving $\vec{s}^\sharp$ takes $O(\frac{K^2M}{\epsilon})$ time complexity and $O(\frac{K^2}{\epsilon})$ space complexity, and for any input instance with feasible assignment $\vec{s}$, $\frac{1}{1-\epsilon}\sum_{k=1}^{K} r_{s_k^\sharp} \geq \sum_{k=1}^{K} r_{s_k}$.*

If integer reward values are given, one may use the original reward values instead of the rounded reward r'_j by setting $\epsilon = \frac{1}{r_M}$. As has been shown in Moser et al. [2009b], the dynamic programming algorithm derives optimal discrete service levels for integer reward values. Of course, also for integer reward values, the computational complexity can be further reduced by choosing a smaller ϵ.

4.3 Dimensioning the Storage Capacity

To design the energy supply of an embedded system, it is important to estimate how to dimension the energy storage device. Given an initial energy $E_C(0)$, an energy source $E_S(k), 1 \leq k \leq K$, and a final energy constraint E_ℓ, we are interested in the minimum storage capacity which is needed to achieve the maximum possible reward. We denote $C_{\min}$ the minimum capacity C for which

the optimal reward equals the reward for an unconstrained system which $C = +\infty$. For a single horizon, we can now determine

$$C_{\min} = \max_{k=1,2,\ldots,K} \left\{ E_C(0) + \sum_{j=1}^{k} (E_S(j) - s_j^*) \right\},$$

where $\vec{s}^*$ is an assignment computed by algorithm RD in Section 4.2.1 or the dynamic algorithms in Section 4.2.2 for specified $E_S(k)$ for $k = 1, 2, \ldots, K$, $E_C(0)$, E_ℓ and $C = \infty$. By choosing the maximum of all capacities $C_{\min}$, it is possible to determine the minimum capacity C to optimally exploit a given environmental source.

4.4 Experimental Evaluation

4.5 Choosing Sufficient Parameters *K* and *C*

A fundamental question one might ask when designing a system in a given environment is: How many days should the horizon span to obtain reasonable rewards? Since solar energy shows a similar pattern every 24 hours, multiples of a day are reasonable choices for the prediction horizon. The presented results in this article are conducted by setting 16 frames per day, which means a frame lasts for 1.5 hours. The results for different numbers of frames per day are quite similar. When investigating different values for the number of frames of the prediction horizon $K \in \{16, 32, 48, \ldots\}$, we use the shorter notation $K \in \{1d, 2d, 3d, \ldots\}$ but keep in mind that we have 16 frames per day. At the end of the prediction horizon, we want the remaining energy $E_C(K)$ to be at least equal to the initial energy $E_C(0)$, that is, $E_\ell = E_C(0)$.

We simulated algorithm RD for $K \in \{1d, 2d, 3d, 5d, 10d, 15d, 30d, 70d, 105d, 210d\}$ for the continuous problem. To obtain a total simulated time of 210 days for each experiment, we repeateady computed $\{210, 105, 70, 42, 21, 14, 7, 3, 2, 1\}$ horizons, respectively. The resulting energy assignments $\vec{s}$ are evaluated by applying the reward function $r(s) = \ln(0.01 + \frac{s}{1000})$ which assigns negative rewards (i.e., penalties) to energies $s < 990$. For each experiment, we calculate the accumulated reward for 210 days.

As a matter of fact, the accumulated reward depends both on the number d of days of the prediction horizon *and* the energy storage capacity C. In Figure 6 we see that the accumulated reward increases quickly with the parameters d and C. The minimum energy capacity $C_{\min}$ required to optimally exploit this energy source is $C_{\min} = 759450$. Using this value for the battery, a horizon of $d = 15$ days is sufficient to achieve 93.4% of the maximum possible reward (i.e., the reward for $d = C = \infty$). For this particular reward function, however, also smaller capacities C are possible to achieve a similar reward.

Using the presented algorithms and simulation methods, one can easily dimension parameters like d and C before deploying the embedded system in a certain environment. As a requirement, a trace of the energy profile has to be provided which is representative for the actual energy source E_S.

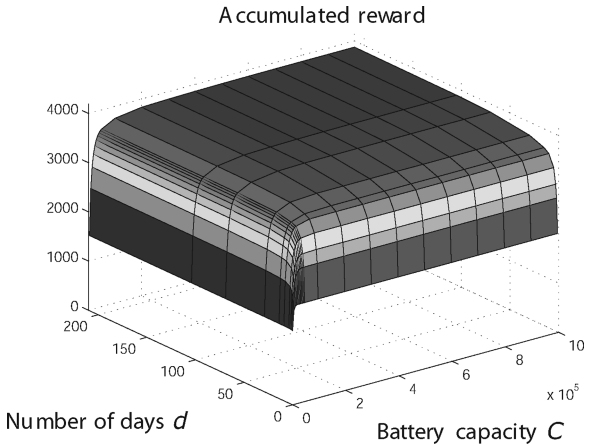


Fig. 6. Accumulated reward for 210 days computed by algorithm RD, $E_C(0) = E_\ell = 3000$.

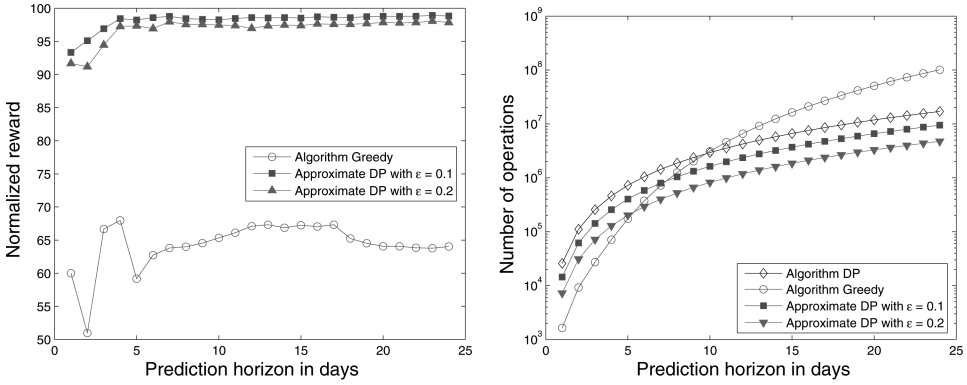
4.6 Greedy Algorithm vs. Dynamic Programming

In the following, we will compare the presented algorithms for the discrete reward maximization problem. We choose an example task with 5 service levels with energies $e_j \in \{1000, 3000, 5000, 6000, 8000\}$ and reward values $r_j \in \{4, 5, 12, 13, 18\}$. Since all reward values are integers, we can compute the optimal service levels using the dynamic programming algorithm with $\epsilon = \frac{1}{8000}$. We will simply denote this algorithm as algorithm DP from now on.

We experienced that for tasks with many service levels, the greedy algorithm closely approximates the optimal solution derived by dynamic programming. However, for the examples with few service levels, the greedy algorithm can only roughly approximate the optimal solution. As displayed in Figure 7(a), the average reward of the greedy algorithm is converging towards ≈63% of the optimal reward value for the mentioned example application. Algorithm DP achieves a normalized reward of 100% for all prediction horizons. On the other hand, Figure 7(a) also displays two approximations of the dynamic programming with tolerances $\epsilon = 0.1$ and $\epsilon = 0.2$. They compute service levels whose rewards are only a few percent from the optimal ones. Note that this problem cannot be solved in acceptable time using a full search.

Instead of measuring the running time on a certain platform, we decided to choose a platform-independent evaluation of our algorithms. For this purpose, we counted the number of operations (both arithmetic as well as logic operations) which have to be executed for each algorithm. As depicted in Figure 7(b), the greedy algorithm is only more efficient for short prediction horizons. For our implementations of the algorithms, however, the greedy algorithm even has a higher computational load for longer prediction horizons. It becomes obvious that the two approximation algorithms can substantially reduce the computation load.

For the relevant horizons, we counted at most several millions of operations for all algorithms. In consideration of the fact that current sensor node platforms like the Tmote Sky [Polastre et al. 2005] or the BTnode [Polastre et al.



(a) normalized reward in %

(b) computational overhead

Fig. 7. Comparison of the greedy algorithm, the dynamic programming algorithm, and two approximated DP solutions with $\epsilon = 0.1$ and $\epsilon = 0.2$.

2005] are operated with frequencies between 4 MHz and 16 MHz, we believe that our algorithms are executed efficiently on common sensor nodes.

In summary, the greedy approximation of the continous solution may give quite poor solutions for the discrete problem. For a practical implementation, one would rather choose a suitable dynamic programming approximation. In doing so, one can easily trade performance versus computation time and give performance guarantees for worst-case scenarios. Concerning the memory requirement for internal variables, we found that the introduced overhead is neglectable for all presented algorithms. Like that, our algorithms can be implemented efficiently on embedded systems, even on resource-constrained systems, for example, sensor nodes.

5. REAL-TIME SCHEDULER

In this section, we consider applications with real-time requirements where tasks are given by arrival times, deadlines, computation times, as well as a certain amount of energy which is required to complete each task. We point out that greedy scheduling is not suitable if tasks are processed using regenerative energy. We present lazy scheduling LSA, an optimal real-time scheduling algorithm for energy harvesting systems.

This algorithm could be applied in scenarios where the application parameters are either fixed or determined by an application rate controller or a service-level allocator. For instance, we know from our experiments that reasonable values T for updates of the application rate controller range from several minutes up to one hour (compare Figure 2(a)). It is now the responsibility of the real-time scheduler to find a feasible schedule on a short-term basis, that is, for time intervals smaller than T.

5.1 Task Model and Problem Definition

On an embedded system, a resource, for example, the microprocessor drains energy E_D and uses it to process tasks with arrival times a_i, energy demands

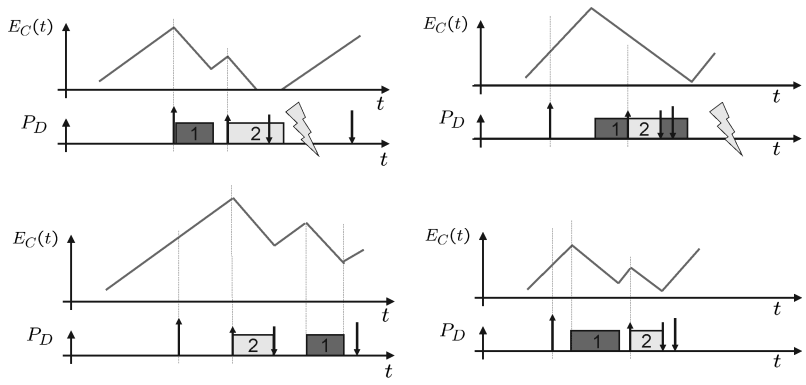
e_i, and deadlines d_i. We assume that only one task is executed at time t and preemptions are allowed. There may be tasks which are activated directly or indirectly by an application rate controller or a service-level allocator. In addition, tasks may be invoked by external events or the communication protocol.

We utilize the notion of a computing device that assigns energy E_D to dynamically arriving tasks. We assume that the corresponding power consumption $P_D(t)$ is limited by some maximum value $P_{\max}$. In other words, the processing device determines at any point in time t how much power it uses, that is, $0 \leq P_D(t) \leq P_{\max}$. If the node decides to assign power $P_i(t)$ to the execution of task i during the interval $[t_1, t_2]$, we denote the corresponding energy $E_i(t_1, t_2) = \int_{t_1}^{t_2} P_i(t)dt$. The effective starting time ϕ_i and finishing time f_i of task i are dependent on the scheduling strategy used: A task starting at time ϕ_i will finish as soon as the required amount of energy e_i has been consumed by it. We can write $f_i = \min\{t : E_i(\phi_i, t) = e_i\}$. The actual running time $(f_i - \phi_i)$ of a task i directly depends on the amount of power $P_i(t)$ which is driving the task during $\phi_i \leq t \leq f_i$. In the best case, a task may finish after the execution time $w_i = \frac{e_i}{P_{\max}}$ if it is processed without interrupts and with the maximum power $P_{\max}$. For a more detailed discussion about practical task processing and the system model in general, see Moser et al. [2007a].

The problem statement presented in this section comprises two major constraints which have to be satisfied: First, tasks can be processed exclusively with energy E_S generated by the energy source. And second, timing constraints in terms of task deadlines d_i must be respected. For this purpose, two degrees of freedom can be exploited. The scheduler may decide which task of all ready tasks to execute and what amount of power P_D to assign to this task.

5.2 Why Naïve Approaches Fail

In the following, we will demonstrate that conventional algorithms from scheduling theory are unsuitable for energy harvesting systems. The example in Figure 8(a) illustrates why greedy scheduling algorithms (like Earliest Deadline First EDF) are not suited in the context of regenerative energy. Let us consider a sensor node with an energy harvesting unit that replenishes a battery. For the sake of simplicity, assume that the harvesting unit provides a constant power output. Now, this node has to execute an arriving Task 1 that has to be finished until a certain deadline. Meanwhile, a second Task 2 arrives that has to respect a deadline which is earlier than the one of Task 1. In Figure 8(a), the arrival times and deadlines of both tasks are indicated by up and down arrows, respectively. As depicted in the top diagrams, a greedy scheduling strategy violates the deadline of Task 2 since it dispenses overhasty the stored energy by driving Task 1. When the energy is required to execute the second task, the battery level is not sufficient to meet the deadline. In this example, however, a scheduling strategy that hesitates to spend energy on Task 1 meets both deadlines. The bottom plots illustrate how a lazy scheduling algorithm described in this article outperforms a naïve, greedy approach like EDF in this situation. It will be showed in the next section how an optimal starting time ϕ_i for a task i can be computed.



(a) Greedy vs. lazy scheduling. (b) ALAP vs. lazy scheduling.

Fig. 8. Why naïve approaches fail: A greedy scheduling discipline, like, for example, EDF scheduling, will lead to energy conflicts 8(a). On the other hand, scheduling tasks as late as possible will lead to timing conflicts 8(b).

On the other hand, starting a task As Late As Possible (ALAP) seems to be a promising approach. The upper plots in Figure 8(b) display a straightforward ALAP translation of the starting time for Task 1: To fulfill its time condition, Task 1 begins to execute at starting time $\phi_1 = d_1 - \frac{e_1}{P_{\max}}$. As illustrated, it may happen that shortly after ϕ_1 an unexpected second task arrives. Assume that this unexpected Task 2 is nested in Task 1, that is, it also has an earlier deadline than Task 1. This scenario inevitably leads to a deadline violation, although plenty of energy is available. This kind of timing conflict can be solved by shifting ϕ_1 to earlier times and thereby reserving time for the unpredictable Task 2 (see lower plots Figure 8(b)). But starting earlier, we risk to "steal" energy that might be needed at later times.

5.3 Optimal Lazy Scheduling Algorithm LSA

From the examples in the previous section, we learned that it may be disadvantageous to prearrange a starting time in such a way that the stored energy E_C cannot be used before the deadline of a task. If the processing device starts running at time ϕ_i with $P_{\max}$ and cannot consume all the available energy before the deadline d_i, time conflicts may occur. On the other hand, energy conflicts are possible if the stored energy $E_C(t)$ is 0 at some time $t < d_i$. Hence we can conclude the following: The optimal starting time ϕ_i must guarantee that the processor could continuously use $P_{\max}$ in the interval $[\phi_i, d_i]$ and empty the energy storage $E_C(d_i) = 0$ exactly at time d_i. Before the optimal starting time ϕ_i, the scheduler has to conserve energy and keep the storage level E_C as high as possible. It can be shown that the optimal starting time ϕ_i can be written as

$$\phi_i = d_i - \frac{\min(E_C(a_i) + \hat{E}(a_i, d_i), C + \hat{E}(\phi_i, d_i))}{P_{\max}},$$

where $\hat{E}(a_i, d_i)$ denotes the estimation of the accumulated energy E_S between the arrival and the deadline of task i. The pseudocode of the Lazy Scheduling

Algorithm 1

Require: maintain a set of indices $i \in Q$ of all ready but not finished tasks J_i
  $P_D(t) \Leftarrow 0$;
  while (true) **do**
    $d_j \Leftarrow \min\{d_i : i \in Q\}$;
    calculate ϕ_j;
    process task J_j with power $P_D(t)$;
    $t \Leftarrow$ current time;
    if $t = a_k$ **then** add index k to Q;
    if $t = f_j$ **then** remove index j from Q;
    if $E_C(t) = C$ **then** $P_D(t) \Leftarrow P_S(t)$;
    if $t \geq \phi_j$ **then** $P_D(t) \Leftarrow P_{\max}$;

Algorithm LSA is depicted in Algorithm 1. It is based on the following rules.

—*Rule* 1. EDF scheduling is used at time t for assigning the computing device to all waiting tasks with $\phi_i \leq t$. The currently running task is powered with $P_D(t) = P_{\max}$.
—*Rule* 2. If there is no waiting task i with $\phi_i \leq t$ and if $E_C(t) = C$, then all incoming power P_S is used to process the task with the smallest deadline, where P_S denotes the current power generated by the energy source.

If no prediction mistakes occur, that is, if $\hat{E} = E_S$ holds for all times t, we proved the optimality of lazy scheduling as follows: If LSA cannot schedule a given task set, then no other scheduling algorithm is able to schedule it [Moser et al. 2007a].

Simulation results show that also with imperfect energy estimation LSA may reduce the deadline miss-ratio significantly. Although it is based on an estimation of the future incoming energy E_S, LSA remains an online algorithm. The calculation of ϕ_i must be performed once the scheduler selects the task with the earliest deadline. If the scheduler is not energy-constrained, that is, if the available energy is more than the device can consume with power $P_{\max}$ within $[a_i, d_i]$, the starting time ϕ_i will be before the current time t. Then, the resulting scheduling policy is EDF, which is reasonable, because only time constraints have to be satisfied. If, however, the sum of stored energy E_C plus generated energy E_S is small, the scheduling policy changes towards an ALAP policy. In doing so, LSA avoids spending scarce energy on the "wrong" tasks too early.

In summary, LSA can be classified as an harvesting-aware adaptation of the earliest deadline first algorithm. It changes its behavior according to the amount of available energy, the capacity C, as well as the maximum power consumption $P_{\max}$ of the device. For example, the lower the power $P_{\max}$ gets, the greedier LSA gets. On the other hand, high values of $P_{\max}$ force LSA to hesitate and postpone the starting time ϕ_i. For $P_{\max} = \infty$, all starting times collapse to the respective deadlines, and LSA degenerates to an ALAP policy.

6. CONCLUSIONS

The application scenarios investigated in this article give fundamental insight in the challenges of power management for energy harvesting systems. While most conventional power management solutions aim to save energy subject to given performance constraints, performance constraints are not given a priori for the energy harvesting systems discussed in this article. Rather, the performance is adapted in a best-effort manner according to the availability of environmental energy. The goal is to optimize the performance of the application subject to given energy constraints. We demonstrate that classical power management solutions have to be reconceived and/or new problems arise if perpetual operation of the system is required. In particular, we provide a set of algorithms and methods for different application scenarios, including real-time scheduling, application rate control, as well as service-level allocation. The purpose of the latter two approaches is to decide which and how many tasks are executed in a long-term perspective. Based on an estimation of the energy harvested in the future, long-term decisions on the use of the available energy are made. On a task level, we provide real-time scheduling algorithms to assign energy to upcoming tasks in a short-term perspective. By taking into account both available time and energy, an optimal task ordering is computed to avoid deadline violations. For nonreal-time applications, an application rate controller or service-level allocator may be sufficient. The other way round, if the application parameters are determined by external conditions (e.g., sensor data, events, or the protocol between neighbor nodes) a real-time scheduler for local signal processing may be required. The proposed theory leads to practical techniques for scheduling processes in energy harvesting systems.

REFERENCES

BERNER FACHHOCHSCHULE. 2007. Bern University of Applied Sciences, Engineering and Information Technologies, Photovoltaic Lab: Recordings of solar light intensity at Mont Soleil from 01/01/2002 to 31/09/2006. www.pvtest.ch.

ALENAWY, T. A. AND AYDIN, H. 2004. On energy-constrained real-time scheduling. In *Proceedings of the EuroMicro Conference on Real-Time Systems (ECRTS'04)*. 165–174.

AYDIN, H., MELHEM, R., MOSSE, D., AND ALVAREZ, P. 1999. Optimal reward-based scheduling for periodic real-time tasks. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*. 79–89.

BEUTEL, J., DYER, M., HINZ, M., MEIER, L., AND RINGWALD, M. 2004. Next-Generation prototyping of sensor networks. In *Proceedings of the 2nd ACM Conference Embedded Networked Sensor Systems (SenSys'04)*. ACM Press, New York, 291–292.

BORRELLI, F., BEMPORAD, A., AND MORARI, M. 2003. A geometric algorithm for multi-parametric linear programming. *J. Optimiz. Theory Appl. 118*, 3, 515–540.

BRUNELLI, D., MOSER, C., THIELE, L., AND BENINI, L. 2009. Design of a solar harvesting circuit for battery-less embedded systems. *IEEE Trans. Circ. Syst.* To appear.

CHEN, J.-J. AND KUO, T.-W. 2005. Voltage-Scaling scheduling for periodic real-time tasks in reward maximization. In *Proceedings of the 26th IEEE Real-Time Systems Symposium (RTSS)*. 345–355.

DEY, J. K., KUROSE, J. F., AND TOWSLEY, D. F. 1996. On-Line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE Trans. Comput. 45*, 7, 802–813.

ESRAM, T. AND CHAPMAN, P. 2007. Comparison of photovoltaic array maximum power point tracking techniques. *IEEE Trans. Energy Conver. 2*, 439–339.

HSU, J., KANSAL, A., FRIEDMAN, J., RAGHUNATHAN, V., AND SRIVASTAVA, M. 2005. Energy harvesting support for sensor networks. In *SPOTS track at Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*.

IEEE. 2006. Tmote sky—Ultra low power. IEEE 802.15.4 Compliant Wireless Sensor Module, Datasheet.

JIANG, X., POLASTRE, J., AND CULLER, D. E. 2005. Perpetual environmentally powered sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN'05)*. 463–468.

KANSAL, A., HSU, J., ZAHEDI, S., AND SRIVASTAVA, M. B. 2007. Power management in energy harvesting sensor networks. *Trans. Embed. Comput. Syst. 6*, 4, 32.

KVASNICA, M., GRIEDER, P., AND BAOTIĆ, M. 2004. Multi-Parametric Toolbox (MPT).

LIU, J. W.-S., LIN, K.-J., SHIH, W.-K., YU, A. C.-S., CHUNG, C., YAO, J., AND ZHAO, W. 1991. Algorithms for scheduling imprecise computations. *IEEE Comput. 24,* 5, 58–68.

MOSER, C., BRUNELLI, D., THIELE, L., AND BENINI, L. 2007a. Real-Time scheduling for energy harvesting sensor nodes. In *Real-Time Systems*. Vol. 37. Kluwer Academic Publishers, Norwell, MA, 233–260.

MOSER, C., CHEN, J.-J., AND THIELE, L. 2008a. Reward maximization for embedded systems with renewable energies. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'08) 0*. 247–256.

MOSER, C., CHEN, J.-J., AND THIELE, L. 2009a. Optimal service level allocation in environmentally powered embedded systems. In *Proceedings of the ACM Symposium on Applied Computing (SAC'09)*. ACM, New York, 1650–1657.

MOSER, C., CHEN, J.-J., AND THIELE, L. 2009b. Power management in energy harvesting embedded systems with discrete service levels. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED'09)*. ACM, New York, 413–418.

MOSER, C., THIELE, L., BRUNELLI, D., AND BENINI, L. 2010. Adaptive power management for environmentally powered systems. *IEEE Trans. Comput.* To appear.

MOSER, C., THIELE, L., BRUNELLI, D., AND BENINI, L. 2007b. Adaptive power management in energy harvesting systems. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'07)*. ACM Press, 773–778.

MOSER, C., THIELE, L., BRUNELLI, D., AND BENINI, L. 2008b. Robust and low complexity rate control for solar powered sensors. In *Proceedings of the Design, Automation and Test in Europe (DATE'08)*.

POLASTRE, J., SZEWCZYK, R., AND CULLER, D. 2005. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Conference on Information Processing in Sensor Networks (IPSN'05)*. 364–369.

RUSU, C., MELHEM, R., AND MOSSE, D. 2002. Maximizing the system value while satisfying time and energy constraints. In *Proceedings of the IEEE 23th Real-Time System Symposium*. 246–255.

RUSU, C., MELHEM, R., AND MOSSÉ, D. 2003. Multiversion scheduling in rechargeable energy-aware real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems (ECRTS'03)*. 95–104.

SHIH, W.-K., LIU, J. W.-S., AND CHUNG, J.-Y. 1991. Algorithms for scheduling imprecise computations with timing constraints. *SIAM J. Comput. 20*, 3, 537–552.

VIGORITO, C. M., GANESAN, D., AND BARTO, A. G. 2007. Adaptive control of duty cycling in energy-harvesting wireless sensor networks. In *Proceedings of the 4th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON'07)*. 21–30.

Received October 2009; revised February 2010; accepted February 2010