Dynamic Power Management in Environmentally Powered Systems

Clemens Moser, Jian-Jia Chen, and Lothar Thiele
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
Email: cmoser@tik.ee.ethz.ch, jchen@tik.ee.ethz.ch, thiele@tik.ee.ethz.ch

ABSTRACT

In this paper a framework for energy management in energy harvesting embedded systems is presented. As a possible example scenario, we focus on wireless sensor nodes which 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 different 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 allows the usage of, e.g., smaller solar cells and smaller batteries. Our theoretical results are supported by simulations using long-term measurements of solar energy in an outdoor environment. Furthermore, to demonstrate the practical relevance of our approaches, we measured the implementation overhead of our algorithms on real sensor nodes.

Keywords: Power management, embedded systems, energy harvesting, model predictive control, real-time scheduling, reward maximization.

I. 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, like 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. 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 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 [9]. 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 [10] and Prometheus [11].

In this paper, a general approach to optimize the utilization of solar energy will be presented which builds on our works in [15–19]. The class of linear programs presented is capable of modeling a large variety of application scenarios, constraints and optimization objectives. Instead of solving the optimization problem online which may be prohibitively complex in terms of running time and energy consumption, we propose a parameterized specification and the computation of a corresponding optimal online controller. 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 which execute applications with real-time constraints. For this purpose, we have constructed an optimal scheduling algorithm that jointly handles constraints from both energy and time domain.

The paper is organized as follows: In the next section, the system concept and the used models and methods are discussed. In Section III, we demonstrate how to obtain simple but optimal controllers which adjust the application rates on a sensor node. In Section IV, we present algorithms to assign service levels with user defined rewards. In Section V, we present Lazy Scheduling Algorithms for optimal online scheduling of real-time applications. Section VI provides a summary and the conclusions.

978-1-4244-5767-0/10/$26.00 2010 IEEE

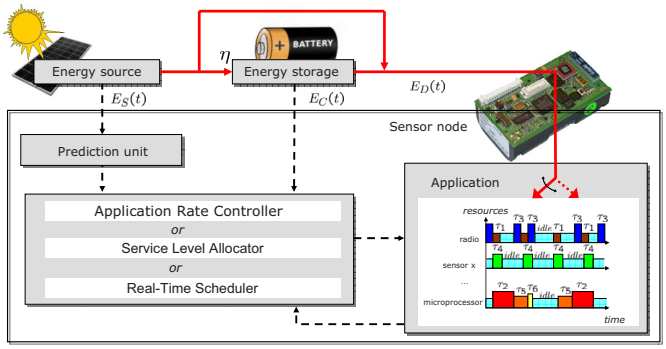


Fig. 1. Illustration of the system model.

II. 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. The capability to bypass the storage device is a typical feature of 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.

As illustrated in Figure 1, this paper 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 III, IV and V, 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. At this, parameters of the application are adapted for the next 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 V 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 VI. 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.

III. Application Rate Controller

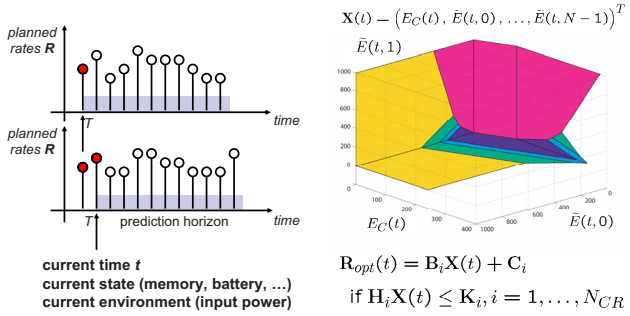
In this section we are adapting parameters of the application to optimize the performance in a long-term perspective. By adapting the activation rates of tasks, we control the average power consumption of the embedded system. As a main contribution, a framework for adaptive power management is proposed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

A. 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 $r_i(t)$, i.e., 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 [21].

B. 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 [12] and [20], 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 [21], 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



(a) Receding Horizon Control. (b) Multiparametric Linear Programming.

Fig. 2. Application Rate Controller for an example application.

mistakes, we refer the reader to related work. The mentioned degradation effects due to energy miss prediction 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 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.

C. 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 e.g. [12,20]. However, solving at each time step t a LP in a resource constraint system like, e.g., 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 [20], the fundamental observation made is that the considered LPs are parameterized and can be solved offline. As illustrated in 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 3-dimensional 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, e.g., the algorithm in [5].

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 a 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.

D. Experimental Evaluation

In this section, both feasibility and practical relevance of the proposed approach is 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 [13]. 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 [1] 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 [20] 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 [4] and measured its running time as well as power consumption. In terms of physical memory, the storage demand of the controller amounts 9,4 Kbyte using the techniques described in [21] and a 16 bit integer representation per

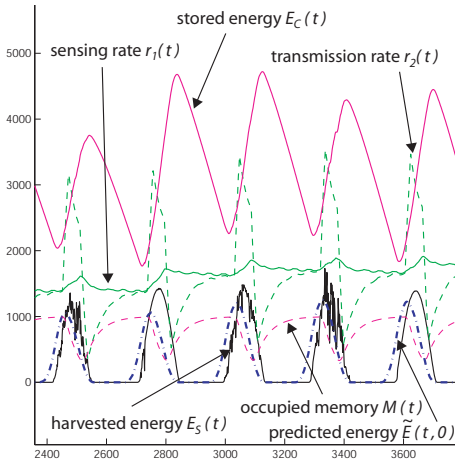


Fig. 3. Adaption of the sensing rate r_1 and the data transmission rate r_2 by a multiparametric linear controller for five days.

coefficient. In other words, the algorithm is applicable to commonly used sensor nodes like the TmoteSky [7], which exhibits 48 Kbyte Flash ROM or the BTnode, with 128 Kbyte. In our implementation on a BTnode, we measured a worst case computation time < 190 ms, yielding a neglectable control overhead every $T = 5$ min. Also the energy consumption for evaluating the control law is neglectable compared to the energy consumption caused by the application rates r_1 and r_2 during the interval T.

IV. 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 B.1 and discrete service levels in Section B.2.

For real-time systems without energy harvesting, reward maximization under energy constraints has been studied extensively as, e.g., in [2,6,23,24]. 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 [14] and the increasing reward with increasing service (IRIS) model [8, 25]. 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 [3].

A. 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 is referred 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 in dependence of the given energy source.

The energy storage, e.g., a supercapacitor or a battery then 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 to the energy storage in the k-th 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.

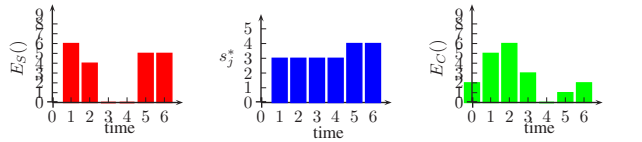
Service and Task Models We investigate how to achieve performance maximization for periodically executed services in environmentally powered systems. To this purpose, the scheduler receives predictions of the future energy $E_S(k)$ generated in the next $1 \leq k \leq K$ frames. We consider a task with 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. For discrete service levels, the j-th 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 storage at time 0 is specified as $E_C(0)$ and the energy harvested in the k-th frame is $E_S(k)$. For brevity, the k-th 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 $E_D(t) = e_{s_k}$ (resp., $E_D(t) = s_k$) for the k-th frame for the discrete (resp., continuous) cases.

B. Service-Level Allocation Algorithms

B.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 this energy consumption. Let us consider the example in Figure 4(a), where 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



(a) $E_S()$ (b) The assignment (c) Energy in the storage

Fig. 4. An example for service allocation.

all these six frames will result in an assignment which is not feasible since there is an energy underflow at the 4-th frame. Figure 4(b) gives an optimal energy consumption assignment, which tries to consume constant amounts of energy without incurring energy underflow.

For the case that there is a battery constraint, i.e., C is not $+\infty$, the problem is more complicated. First, we derive an optimal solution $\vec{s}^*$ with infinite battery capacity as indicated in the previous paragraph. The solution for infinite battery capacity then divides the K frames into some segments, and tries to avoid energy underflows and overflows recursively within these segments. The algorithm is denoted as Algorithm Recursive Decomposition (abbreviated as RD) and has been presented recently in [16]. Moreover, it can be proved that Algorithm RD derives optimal solutions for reward maximization in $O(K^2)$.

B.2 Service Allocation for Discrete Service Levels

For a task with discrete service levels, we present two algorithms: one is a greedy algorithm extended from Algorithm RD in Section B.1, and another is an approximation algorithm with adjustable time complexity.

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 is to choose service level $s_k^\dagger$ for the k-th 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 k-th iteration, Algorithm RD is applied to derive the service level of the k-th 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 k-th frame, where s_k^* is the service level derived from Algorithm RD for the k-th frame.

Dynamic Programming We now present a dynamic programming algorithm that derives better results. 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. Then, for each service level j, we derive the *rounded reward* r'_j of service level j by applying $r'_j = \left\lfloor \frac{r_j}{\epsilon \cdot r_M} \right\rfloor$.

Suppose that $\tilde{E}(k, \rho)$ is the maximum energy residual in the energy storage after servicing the k-th 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:

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

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

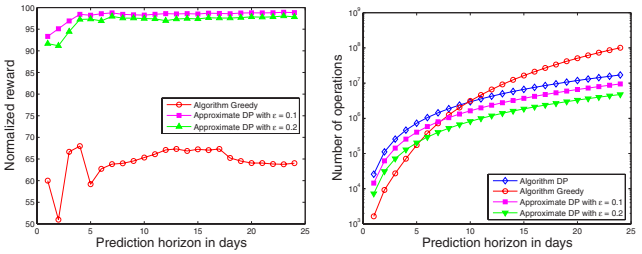
where j'_ρ is the service level j with $r'_{j-1} \leq \rho \leq r'_j$, where r'_0 is assumed 0 here for calculating j'_ρ. Note that the other boundary conditions must also be applied for $\tilde{E}(k, \rho)$.

We can build a dynamic programming table by applying the above recursive function in (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 K-th frame. The following theorem shows that the quality of the derived solution $\vec{s}^\sharp$ of the above 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 $O(\frac{K^2 M}{\epsilon})$ time complexity and $O(\frac{K^2}{\epsilon})$ space complexity, and for any input instance with feasible assignment $\vec{s}$, $\frac{1}{1-\epsilon}\sum_{k=1}^{K} r_{s_k^\sharp} \geq \sum_{k=1}^{K} r_{s_k}$.*

C. 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



(a) Normalized reward in %. (b) Computational overhead.

Fig. 5. Comparison of the Greedy Algorithm, the Dynamic Programming Algorithm, and two approximated DP solutions with $\epsilon = 0.1$ and $\epsilon = 0.2$.

unconstrained system which $C = +\infty$. For a single horizon, we can now determine

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

where $\bar{s}^*$ is an assignment computed by Algorithm RD in Section B.1 or the dynamic algorithms in Section B.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.

D. 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 $r_j \in \{4, 5, 12, 13, 18\}$. The simulation begins with an initial energy $E_C(0) = E_\ell = 5000$ and a battery capacity $C = 10000$.

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 5(a), the average reward of the Greedy Algorithm is converging towards $\approx 63\%$ of the optimal reward value for the mentioned example application. Note, that this problem cannot be solved in acceptable time using a full search. On the other hand, Figure 5(a) also displays two approximation 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.

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 5(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 [22] or the BTnode [22] 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 an 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, e.g., sensor nodes.

V. Real-Time Scheduler

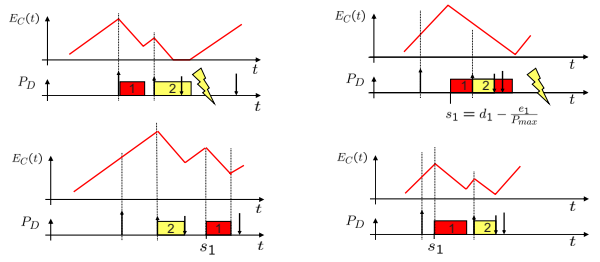
In this section we consider applications with real-time requirements where tasks are given by arrival times, deadlines, computation times as well as a certain amount of energy which is required to complete each task. We point out that greedy scheduling is not suitable if tasks are processed using regenerative 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, i.e., for time intervals smaller than T.

A. Task Model and Problem Definition

On an embedded system a resource, e.g., 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 s_i and finishing time f_i of task i are dependent on the scheduling strategy used: A task starting at time s_i will finish as soon as the required amount of energy e_i has been consumed by it. We can write $f_i = \min\{t : E_i(s_i, t) = e_i\}$.



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

Fig. 6. Why naïve approaches fail: A greedy scheduling discipline, like e.g. EDF scheduling, will lead to energy conflicts 6(a). On the other hand, scheduling tasks as late as possible will lead to timing conflicts 6(b).

The actual running time $(f_i - s_i)$ of a task i directly depends on the amount of power $P_i(t)$ which is driving the task during $s_i \leq t \leq f_i$. 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 [15].

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 tasks 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.

B. 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 6(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 6(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 paper outperforms a naïve, greedy approach like EDF in this situation. It will be showed in the next section how an optimal starting time s_i for a task i can be computed.

On the other hand, starting a task as late as possible (ALAP) seems to be a promising approach. The upper plots in Figure 6(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 $s_1 = d_1 - \frac{e_1}{P_{\max}}$. As illustrated, it may happen that shortly after s_1 an unexpected second task arrives. Assume that this unexpected Task 2 is nested in Task 1, i.e., 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 s_1 to earlier times and thereby reserving time for the unpredictable Task 2 (see lower plots Figure 6(b)). But starting earlier, we risk to "steal" energy that might be needed at later times.

C. Optimal Lazy Scheduling Algorithm LSA

From the examples in the previous section, we learned that it may be disadvantageous to pre-arrange 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 s_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 s_i must guarantee, that the processor could continuously use $P_{\max}$ in the interval $[s_i, d_i]$ and empty the energy storage $E_C(d_i) = 0$ exactly at time d_i. Before the optimal starting time s_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 s_i can be written as

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

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 Lazy Scheduling Algorithm LSA has originally been presented in [15]. 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 $s_i \leq t$. The currently running task is powered with $P_D(t) = P_{\max}$.
- **Rule 2**: If there is no waiting task i with $s_i \leq t$ 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, i.e., if $\hat{E} = E_S$ holds for all times t, we proofed 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 [15].

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 s_i must be performed once the scheduler selects the task with the earliest deadline. If the scheduler is not energy-constraint, i.e., if the available energy is more than the device can consume with power $P_{\max}$ within $[a_i, d_i]$, the starting time s_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 s_i. For $P_{\max} = \infty$, all starting times collapse to the respective deadlines, and LSA degenerates to an ALAP policy.

VI. Conclusions

The application scenarios investigated in this paper 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 paper. 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.

Of course, the single approaches presented in this paper can also be combined. There may be two hierarchically structured schedulers which control the application. In a first step, an Application Rate Controller or a Service Level Allocator decides which and how many tasks are executed on the long run. In a second step, a Real-Time Task Scheduler decides about the short-term task ordering. For example, the Application Rate Controller may invoke a set of periodic tasks with different periods and phases. In addition, certain external events may trigger the execution of sporadic tasks (e.g. communication tasks using the on board radio). In these situations, the Real-Time Task Scheduler will attempt to avoid deadline violation of time critical tasks. To realize this combination of approaches, of course, the presented methods have to be adapted accordingly.

For non real-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.

Acknowledgements

The presented 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.

References

[1] 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, March, 2007.

[2] T. A. Alenawy and H. Aydin. On energy-constrained real-time scheduling. In *EuroMicro Conference on Real-Time Systems (ECRTS'04)*, pages 165–174, 2004.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Alvarez. Optimal reward-based scheduling for periodic real-time tasks. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, pages 79–89, 1999.

[4] J. Beutel, M. Dyer, M. Hinz, L. Meier, and M. Ringwald. Next-generation prototyping of sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 291–292. ACM Press, New York, Nov. 2004.

[5] F. Borrelli, A. Bemporad, and M. Morari. A Geometric Algorithm for Multi-Parametric Linear Programming. *Journal of Optimization Theory and Applications*, 118(3):515–540, Sept. 2003.

[6] J.-J. Chen and T.-W. Kuo. Voltage-scaling scheduling for periodic real-time tasks in reward maximization. In *the 26th IEEE Real-Time Systems Symposium (RTSS)*, pages 345–355, 2005.

[7] M. Corporation. Tmote sky - ultra low power ieee 802.15.4 compliant wireless sensor module, datasheet. June, 2006.

[8] J. K. Dey, J. F. Kurose, and D. F. Towsley. On-line scheduling policies for a class of IRIS (increasing reward with increasing service) real-time tasks. *IEEE Transactions on Computers*, 45(7):802–813, 1996.

[9] T. Esram and P. Chapman. Comparison of photovoltaic array maximum power point tracking techniques. *Energy Conversion, IEEE Transaction on*, 2:439–339, 2007.

[10] J. Hsu, A. Kansal, J. Friedman, V. Raghunathan, and M. Srivastava. Energy harvesting support for sensor networks. In *SPOTS track at IPSN 2005*, 2005.

[11] X. Jiang, J. Polastre, and D. E. Culler. Perpetual environmentally powered sensor networks. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, pages 463–468, UCLA, Los Angeles, California, USA, April 25-27 2005.

[12] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava. Power management in energy harvesting sensor networks. *Transactions on Embedded Computing Systems*, 6(4):32, 2007.

[13] M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004.

[14] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, J. Yao, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, May 1991.

[15] C. Moser, D. Brunelli, L. Thiele, and L. Benini. Real-time scheduling for energy harvesting sensor nodes. In *Real-Time Systems*, volume 37, pages 233–260, Norwell, MA, USA, 2007. Kluwer Academic Publishers.

[16] C. Moser, J.-J. Chen, and L. Thiele. Reward maximization for embedded systems with renewable energies. *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA08)*, 0:247–256, 2008.

[17] C. Moser, J.-J. Chen, and L. Thiele. Optimal service level allocation in environmentally powered embedded systems. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1650–1657, New York, NY, USA, 2009. ACM.

[18] C. Moser, J.-J. Chen, and L. Thiele. Power management in energy harvesting embedded systems with discrete service levels. In *ISLPED '09: Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 413–418, New York, NY, USA, 2009. ACM.

[19] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive power management for environmentally powered systems. In *IEEE Transactions on Computers, to be published, 2009.*

[20] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Adaptive power management in energy harvesting systems. In *DATE '07: Proceedings of the Conference on Design, Automation and Test in Europe*, pages 773–778, NY, USA, 2007. ACM Press.

[21] C. Moser, L. Thiele, D. Brunelli, and L. Benini. Robust and Low Complexity Rate Control for Solar Powered Sensors. In *Design, Automation and Test in Europe (DATE 08)*, Munich, Germany, March 10-14 2008.

[22] J. Polastre, R. Szewczyk, and D. Culler. Telos: Enabling ultra-low power wireless research. In *4th International Conference on Information Processing in Sensor Networks (IPSN05)*, pages pp. 364–369, Piscataway, NJ, April 2005.

[23] C. Rusu, R. Melhem, and D. Mosse. Maximizing the system value while satisfying time and energy constraints. In *IEEE 23th Real-Time System Symposium*, pages 246–255, Dec. 2002.

[24] C. Rusu, R. Melhem, and D. Mossé. Multiversion scheduling in rechargeable energy-aware real-time systems. In *EuroMicro Conference on Real-Time Systems (ECRTS'03)*, pages 95–104, 2003.

[25] W.-K. Shih, J. W.-S. Liu, and J.-Y. Chung. Algorithms for scheduling imprecise computations with timing constraints. *SIAM J. Computing*, 20(3):537–552, June 1991.