Multi-version Scheduling in Rechargeable Energy-aware Real-time Systems

Cosmin Rusu, Rami Melhem, *Fellow, IEEE,* Daniel Mossé, *Member, IEEE*

***Abstract*— In the context of battery-powered real-time systems, three constraints need to be addressed: energy, deadlines and task rewards. Many future real-time systems will count on different software versions, each with different rewards, time and energy requirements, to achieve a variety of QoS-aware tradeoffs. We first propose a solution that allows the device to run the most valuable task versions while still meeting all deadlines and without depleting a given energy budget. Assuming that the energy budget can be replenished by using a rechargeable battery, we also propose (i) a static solution that maximizes the system value assuming a worst-case scenario (i.e., worst-case battery recharging/discharging conditions, as well as worst-case task execution times); and (ii) a dynamic scheme that takes advantage of the extra energy in the system when worst-case scenarios do not happen. Three dynamic policies are shown to make better use of the recharging energy while improving the system value.**

***Index Terms*— power-aware, real-time, scheduling, energy, rechargeable, multi-version**

I. Introduction

POWER management is a critical design factor for embedded systems that rely on their own power source (battery). It would appear as though the lifetime of the device is ultimately dependent on battery storage capacity, but devices may scavenge the existing energy in the environment. An example of such a device is the NASA/JPL Mars rover, which relies on both a non-rechargeable battery source and a solar panel [2]. In our work, we assume that the battery is rechargeable. During rechargeable periods (e.g., daytime for devices with solar panels) real-time tasks are executed at the same time the battery is recharging, while when the system cannot recharge, it relies entirely on the battery energy acquired during the recharging period.

The periods in which recharging is possible may be limited and must be used efficiently. The technique we use in this paper for reducing the energy consumption is voltage and frequency scaling, through which we can achieve cubic savings in power at the expense of linear performance loss. Clearly, this technique is not applicable to all kinds of systems. The performance loss from scaling down the voltage and frequency (leading to more time to compute results) can increase the energy consumption in other components of the system, resulting in an overall increase in the energy consumption. Our proposed schemes are most beneficial for systems where the processing unit is an important energy consumer.

On the other hand, the schemes we propose are not limited to voltage scaling: *our goal is not to minimize energy but to run the most important/valued applications given the timing and energy constraints*. If voltage scaling does not save energy, then our solution would still correctly select the most important tasks (although they would run at their highest frequency). Moreover, using our scheme we can determine the lifetime of the system (assuming worst-case conditions) and whether the system is stable (i.e., at all times there is energy left in the battery).

A large number of energy-constrained embedded real-time systems operate in a cyclic basis, with a set of applications that must execute within a frame whose execution is to be repeated. Examples of such applications are real-time communication and imaging in satellites. An extension of the frame-based task model is a periodic task model with individual deadlines for each task. Our solution is intended for both models. In addition, each task is assumed to have multiple versions, each with different time and energy requirements. Each version has a value or reward associated with it, a measure of task's importance. We assume that versions that require more energy and execution time return more accurate results. An example of versions of applications is when the satellite images being transmitted can be of different sizes or resolution, yielding different rewards but requiring different amounts of energy and time for transmission.

The rest of the paper is organized as follows: We first present related work, followed by a brief background on rechargeability. Task and recharging models are explained in Section II. We present an algorithm for energy/aware task selection in Section III. Based on this algorithm, the solution is completed in Section IV with a static analysis and three dynamic policies, followed by experimental results. We conclude the paper in Section V.

A. Related Work

The variable voltage scheduling (VVS) framework has recently become a major research area. In the context of real-time systems, VVS schemes target minimizing the energy consumption while still meeting the deadlines. Yao et al. [3] provided a static off-line scheduling algorithm assuming aperiodic tasks and worst-case execution times (WCET). Periodic tasks with identical periods and upper bounds on the voltage change rate are investigated in [4]. Systems with two (discrete) voltage levels and periodic hard real-time tasks are analyzed in [5]. An optimal static solution for periodic task sets with different power characteristics is given in [6]. Slack management techniques are explored in [7–10]. In this work we analyze the case of discrete voltage/frequency levels.

Reward-based scheduling was explored in the context of IC (Imprecise Computation) and IRIS (Increased Reward with

This work has been supported by the Defense Advanced Research Projects Agency through the PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736.

A preliminary version was presented at ECRTS'03 [1]

Increased Service) models. In the IC model [11], real-time tasks consist of mandatory and optional parts and a reward function is associated with the length of the optional part. The IRIS model [12], [13] makes no separation between mandatory and optional parts. Typical reward functions are assumed to be linear or concave in the number of cycles allotted to the tasks, targeting applications such as image and speech processing or multimedia. An optimal algorithm for concave reward functions and periodic tasks was presented in [14]. The case for discrete reward functions (or step reward functions) with no reward for partial execution was shown to be NP-hard [11]. In the QRAM model (QoS-based resource allocation model) [15], [16] reward functions are in terms of utilization of resources. A solution was proposed for one resource with multiple QoS dimensions [15] and a particular audio-conferencing application with two resources and one QoS dimension was analyzed in [16]. One work recently published combined the three constraints [17], but voltage/frequency scaling was not considered.

Multi-version programming has been extensively explored in the context of fault tolerance. In this work, multiple versions allow quality of service tradeoffs. An example of version programming comes from satellite-based signal processing [18]. Four different algorithms with running times ranging from microseconds to milliseconds and energy consumptions from microJoules to Joules provide different levels of accuracy. Another example is Automated Target Recognition (ATR), where task values, running times and energy requirements are roughly proportional with the number of targets detected [19]. Task versions can result from different algorithms, as well as from the same application with different input arguments, such as encoding/decoding at different rates, low/high quality compression schemes, low/high resolution image processing, etc. For real-time tasks, QoS tradeoffs can also be achieved through modifying the invocation frequency (period) of applications, as suggested in [17], [20]. This effectively results in task versions where the reward of a task is a function of its invocation frequency.

Rechargeable systems remain generally unexplored. A solution for the Mars Pathfinder rover that makes best use of the available recharging energy is presented in [2]. For frame-based systems with a rechargeable battery, a solution is presented in [21] that schedules tasks in such a way that the wasted recharging energy is minimized and the battery level is at all times within some acceptable limits (no task rewards were considered).

B. Rechargeability background

This section presents a short introduction to rechargeable energy harvesting and storage (illustrated by a solar panel and respectively a rechargeable battery).

A solar cell (also known as PV cell) converts light into electricity through the photo-voltaic effect [22]. The current-voltage characteristics of a typical solar cell is shown in Figure 1. At short circuit the current is maximum (I_{sc}) but the power generated (the voltage multiplied with the current) is zero. Similarly, at open circuit the voltage is maximized

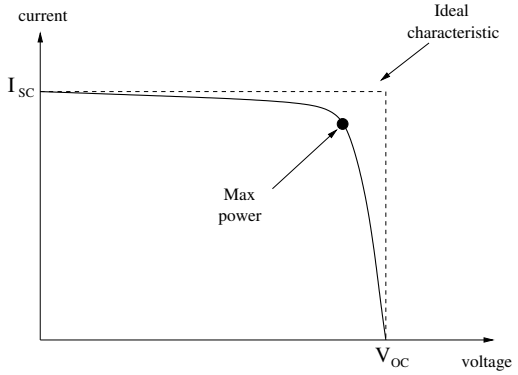


Fig. 1. Current-voltage characteristic of a solar cell

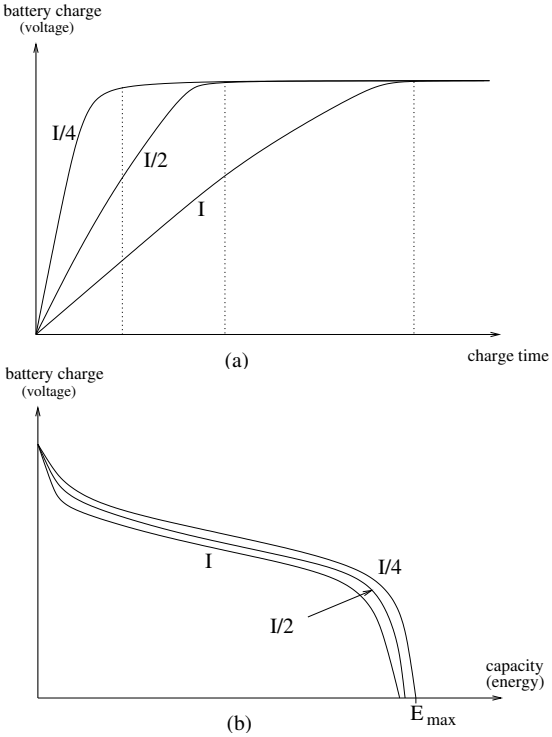


Fig. 2. (a) Charge characteristic (b) Discharge characteristic, as a function of charge/discharge current

(V_{oc}) but the current (and thus power) is zero. The optimal operating point (i.e., maximum power) is shown on the curve as P_{max}. Note that a solar cell cannot store energy by itself. The device attached to the cell will draw as much power as it needs; the remaining power (up to P_{max}) is simply wasted if not used.

A solar panel is obtained by connecting cells in series or parallel into PV arrays to obtain any desired voltage/current characteristic. Connecting two cells in series doubles the resulting V_{oc}; parallel connection doubles the resulting I_{sc}. For each cell I_{sc} depends on the intensity of light, while V_{oc} depends on other parameters (such as temperature).

A rechargeable battery has a nominal capacity (expressed in Amps-hour) corresponding to a maximum energy (expressed in Joules or Watts-hour). The charging characteristic for a typical lithium-ion rechargeable battery is shown in Figure 2(a). The charging time depends on the charge current (or power),

but also on other parameters (like temperature). Not all the power used to recharge the battery can be stored (for example, 1 W of charge for 1 hour results in less than 1 Watt-hour stored energy).

The discharging characteristic is shown in Figure 2(b). The nominal capacity is computed for a given constant discharging current and temperature. A variable discharging current results in a reduced effective capacity.

II. Task/Processor/Energy Models and Problem Definition

The frame-based and periodic task models with their characteristics and scheduling constraints are described first. We continue with a description of the recharging model. Finally, we define the problem and state our goals.

A. Multiple-version Task Model

Frame-based In this model all task periods are identical and all task deadlines are equal to their period. The common deadline/period (also known as frame length) is denoted by D. There are N available periodic tasks in the system, all ready at time zero. Tasks have multiple versions with different characteristics that will be described shortly. Exactly one version of each task is to be scheduled during a frame. Frames are issued periodically every D time units.

Periodic tasks There are N periodic tasks in the system, all ready at time zero. The deadline of the i^{th} task is denoted by D_i and the least common multiple of all task deadlines (also called hyperperiod) is denoted by T_{LCM}. As in the frame-based model, tasks have multiple versions with different characteristics. Exactly one version of each task is to be executed at every instance.

The remainder of this section applies to both task models, unless distinctively specified.

Variable voltage/frequency processor The tasks are to be executed on a variable voltage processor with the ability to dynamically adjust its frequency and voltage on application requests (we refer to a frequency/voltage change as a speed change). There are M available frequencies (clock rates or CPU speeds), $\{f_1, f_2, \ldots, f_M\}$. Each task can run at any of the available speeds and we say that a task runs at speed level k if the speed of the task is set to f_k. Since power functions are proportional with the frequency and with the square of the voltage [3], [4], and since energy is the product of power and time, the benefit of running at small frequencies is a reduced energy consumption for the processing unit, at the expense of increased execution time.

Speed change overhead We assume that the time and energy overhead of speed changes is negligible[1] compared to the deadline D, or that it was already subtracted from D. In the frame based model the number of speed changes that can occur during a frame is minimized by placing tasks that run at the same frequency next to each other. Thus, the maximum number of speed changes that can occur during a frame is $min(M, N)$. In the periodic tasks model, the order of execution is determined by the scheduling algorithm (such as EDF or RMS). We assume the worst-case number of speed changes (two for each instance in the case of EDF or RMS) has a negligible total time and energy requirement, or that it was already subtracted from the time/energy budget.

[1]We have measured $3\mu s$ speed changes in Crusoe chips.

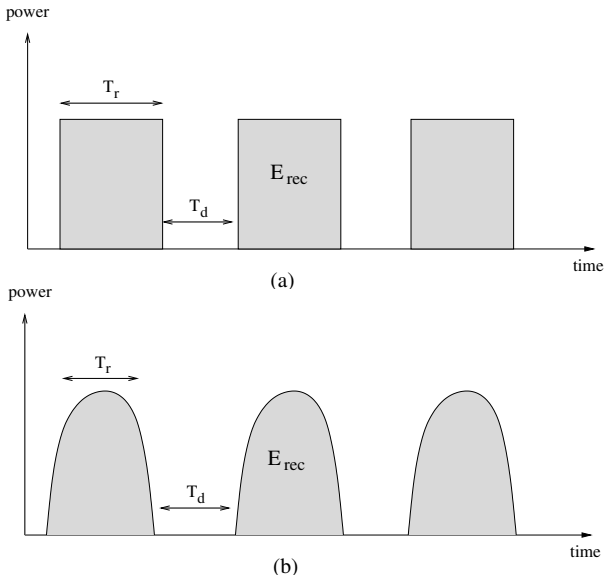


Fig. 3. (a) Constant power (b) Variable power

Task versions, rewards, time and energy Task versions are characterized by three parameters: time and energy requirements (different for each speed), and a version value (or reward - a measure of the importance/accuracy of each task version). The version k of task i at speed level j is denoted by T_{ij}^k. We assume that task worst-case execution times and energy requirements are known for all task versions and all speed levels. The worst-case execution time and energy requirement of version k of task i running at speed level j are denoted by $t_{i,j}^k$ and $e_{i,j}^k$ respectively. Associated with version k of task i there is a version value or reward, r_i^k. We assume that $r_i^k < r_i^{k+1}$, $t_{ij}^k < t_{ij}^{k+1}$ and $e_{ij}^k < e_{ij}^{k+1}$, that is, lower versions execute faster, require less energy, and produce less accurate/complete/valuable results. For simplicity, we assume the same number of versions V and speeds M for each task, although the algorithm proposed can handle different number of versions and speeds for each task.

B. Recharging model

The system we target consists of three components: a processing unit, an energy harvester (such as a solar panel) and a rechargeable battery. The processing unit includes all the components needed for processing real-time tasks, such as a DVS processor, memory and network, and we assume that the task energy values e_{ij}^k refer to the consumption in the entire system comprised by the processing unit. The harvested power can be either used by the processing unit or stored for future use by the third component (rechargeable battery).

The shape of the solar power that can be generated on a satellite orbiting the Earth is shown in Figure 3(a). The power is either constant (about 1350W/m^2) or zero if sunlight is

obstructed [22]. We will refer to the time when there is solar power as T_r (recharging time). T_d (discharging time) denotes the time when the system has to rely entirely on the battery (i.e., no solar power). The amount of energy generated during T_r (the area below the power curve) is denoted by E_{rec}. Note that E_{rec} is only a fraction of the solar power, as the efficiency of a solar cell is typically 10% to 20%. The solar power on Earth's surface varies with time due to atmosphere and clouds, as shown in Figure 3(b). For this scenario, E_{rec} will denote the worst-case amount of energy that can be generated during T_r.

We denote the maximum energy that can be stored in the rechargeable battery by E_{max}. As described in Section I-B, there is a loss of energy when recharging and discharging the battery. We'll use a parameter, $\alpha \in [0, 1]$, to denote the worst-case recharging loss. For example, for $\alpha = 0.9$ and $E_{max} = 9Wh$, $10Wh$ may be needed to fully charge the battery. A second parameter, $\beta \in [0, 1]$ denotes the worst-case discharging loss. Thus, if $\beta = 0.9$ and $E_{max} = 10Wh$, the actual energy is just $9Wh$ under a worst-case discharging scenario.

Our goal is to have a stable system in which the battery energy is at all times above a specified limit E_{min}. E_{min} can be higher than zero if it would be considered safe to have at all times some energy stored in the battery. For such a stable system we also determine how to most efficiently distribute the energy so as to maximize the reward (or value) acquired by the system. That is, we determine how much of the solar power should the processing unit draw, and how much should be stored for the discharging period T_d, so as to maximize the system reward.

C. Problem Definition

Our goal is to determine for each frame how much energy to allocate so that the system is stable (i.e., the battery energy can never be less than E_{min}), provided that E_{rec}, α and β, as well as task worst-case execution times and energy requirements are not underestimated. For a stable system we also determine what task versions v_i to select and at what speed levels s_i to run them so as to maximize the system value (the sum of values for all versions selected for execution in all discharging and recharging frames).

We first present the problem definition for maximizing the total value (reward) of a single frame for multiple-version task sets with a given fixed energy budget and then present the case of periodic tasks. The total value of a frame is defined as the sum of rewards for all task versions selected for execution. In this problem we assume that an energy budget E is associated with the frame and the goal is to maximize the total value without exceeding the available energy E.

Frame-based Formally, the problem is to determine for each task i its version v_i and speed level s_i, so as to:

$$\text{maximize} \quad \sum_{i=1}^{N} r_i^{v_i} \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{N} t_{i,s_i}^{v_i} \leq D \tag{2}$$

$$\sum_{i=1}^{N} e_{i,s_i}^{v_i} \leq E \tag{3}$$

$$v_i \in \{1, 2, \ldots, V\} \tag{4}$$

$$s_i \in \{1, 2, \ldots, M\} \tag{5}$$

Inequality (2) guarantees that the timing constraint is satisfied, and inequality (3) guarantees that the energy budget is not exceeded.

Periodic tasks For periodic tasks the problem is similar:

$$\text{maximize} \quad \sum_{i=1}^{N} r_i^{v_i} \frac{T_{LCM}}{D_i} \tag{6}$$

$$\text{subject to} \quad \sum_{i=1}^{N} \frac{t_{i,s_i}^{v_i}}{D_i} \leq 1 \tag{7}$$

$$\sum_{i=1}^{N} e_{i,s_i}^{v_i} \frac{T_{LCM}}{D_i} \leq E \tag{8}$$

$$v_i \in \{1, 2, \ldots, V\} \tag{9}$$

$$s_i \in \{1, 2, \ldots, M\} \tag{10}$$

The total reward of the hyperperiod is the sum of rewards for all task instances (6). Similarly, the energy consumption of all instances is accounted for in (8). The timing constraint in (7) assumes EDF scheduling. A different utilization formula can be used with different schedulers, such as RMS. Observe that problems (1)-(5) and (6)-(10) are equivalent (assuming EDF scheduling). The periodic task set is corresponding to a frame of length T_{LCM} in which the time, energy and value of each task T_i are multiplied with $\frac{T_{LCM}}{D_i}$.

Note that the problem formulation (6)-(10) assumes that the energy budget E is associated with T_{LCM}. If the energy budget is given for a time $t \neq T_{LCM}$, then $\lfloor \frac{t}{D_i} \rfloor$ must be used as the number of task invocations in Equation (6) and $\lceil \frac{t}{D_i} \rceil$ must be used in Equations (7) and (8).

A solution for the problem defined by Equations (1)-(5) or (6)-(10) is reviewed in the next section. In the context of rechargeable systems, we will show how to determine the energy budget E for each frame / hyperperiod in Section IV. The problem was shown in [23] to be NP-hard even for single-version task sets in frame-based systems. Therefore, we relax the maximization objective and look for solutions that approximate the optimal solution.

III. Energy/Value-Aware Task Selection

The algorithm *MV-Pack* that we propose to solve the problems described by Equations (1)-(5) and (6)-(10) is an extension of the *REW-Pack* algorithm proposed in [23] for the case of single version frame-based task sets. The *MV-Pack* algorithm, first presented in [1], is reviewed next.

Throughout the rest of the paper we will only be referring to frames and the problem described by Equations (1)-(5). As mentioned before, the two formulations (frame-based and periodic task sets) are equivalent.

The flowchart of the algorithm is presented in Figure 4. The three major components (add task, increase speed and increase version) are described next. We denote by $time$ and $energy$

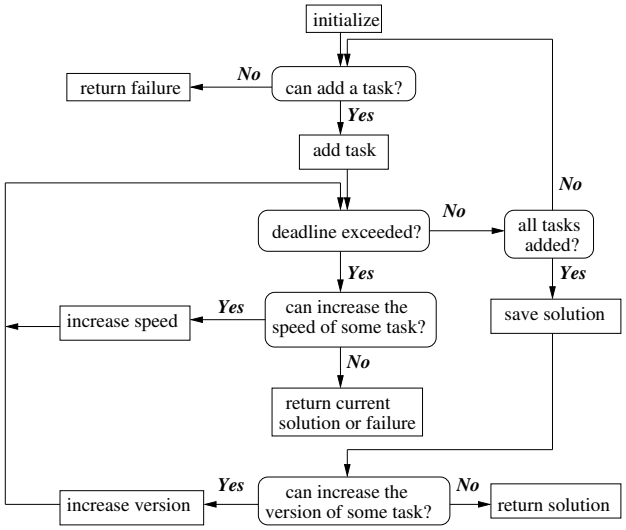


Fig. 4. Flowchart of *MV-Pack*

the total execution time and energy requirements of the current schedule.

Add a task When it is possible to add a new task, it is added always at the first (smallest) speed level and version (we assume that task versions are sorted by their reward – the first version has the smallest reward). The task to be added satisfies all of the following criteria:

- It was not considered before.
- The current schedule is feasible ($time \leq D$).
- By adding the task to the current schedule at the minimum speed the energy budget is not exceeded ($energy + e_{i,1}^1 \leq E$).
- Among all the tasks that satisfy the above criteria, select the one that has the largest ratio $\frac{r_i^1}{t_{i,1}^1 e_{i,1}^1}$.

The task added must have a good (large) reward, a reasonable (small) running time and a reasonable (small) energy consumption. Hence the metric used to decide which task is best to add is proportional to the reward and inversely proportional to the time and the energy required by the task. The task with the highest metric is considered the best. Observe that for each task, the smaller the speed, the larger the value of the metric, since energy increases more than linearly with the speed while time decreases approximately linearly and the task value remains the same regardless of the running speed. Thus, it is reasonable to start with the smallest speed (level 1) and later increase the task's speed, if possible. We experimented with different heuristics, such as adding tasks at the smallest speed that does not exceed the deadline, and they consistently return smaller system values than the present heuristic.

Increase the speed of a task If the deadline is exceeded, the algorithm *packs* tasks to make room for other not yet selected tasks, where *packing* means to increase the speed of one of the selected tasks, to the next higher speed level. The task chosen for a speed increase must satisfy the following:

- It must be selected in the current schedule.
- It is not running at the maximum speed ($s_i \neq M$).
- By increasing its speed to the next higher speed level the energy budget is still not exceeded ($energy + e_{i,s_i+1}^{v_i} - e_{i,s_i}^{v_i} \leq E$).
- Among all selected tasks it has the highest ratio $\frac{\Delta t}{\Delta E}$, where $\Delta t = t_{i,s_i}^{v_i} - t_{i,s_i+1}^{v_i}$ and $\Delta E = e_{i,s_i+1}^{v_i} - e_{i,s_i}^{v_i}$.

Packing reduces the total execution time and increases the energy consumption. The best candidates are considered the tasks that create a lot of room (time or slack) for the remaining tasks while not significantly increasing the energy consumption. Task values do not play any role here as the total reward is not changed by the packing operation, since the task version remains the same during packing.

Increase version of a task When all the tasks are selected in the schedule, a minimum reward solution is found, otherwise failure is returned. The third component of the algorithm (increase version) selects the task to move to its next higher version. The old version is removed from the schedule, while the new version is added at the minimum speed. The task i that is selected to move to the next higher reward version satisfies:

- It is not running at the highest version ($v_i < V$).
- By replacing the current version with the next higher version at the first speed level, the energy budget is not exceeded ($energy + e_{i,1}^{v_i+1} - e_{i,s_i}^{v_i} \leq E$).
- Among all the tasks that are not running at their highest version, the next version at minimum speed has the largest reward per unit time and energy. That is, we select task i that maximizes $\frac{r_i^{v_i+1}}{t_{i,1}^{v_i+1} e_{i,1}^{v_i+1}}$.

Note that by changing the version of a task, the deadline may be violated. If necessary, tasks are packed until either a feasible schedule with the new version is found or the energy is exceeded; in this latter case, the algorithm stops and the current solution (with lower version) is returned.

Complexity The complexity of *MV-Pack* can be analyzed as follows. Each task is added at most once and its version can be increased at most $V-1$ times. For each task we can increase its speed at most $(M-1)V$ times. With appropriate data structures (priority queues for example), determining which task to choose takes $\log N$ time for all functions (add task, increase speed and increase version). Thus, the complexity of the algorithm is $O(MVN \log N)$.

Optional tasks Observe that in the problem definition (Equations(1)-(5)) all tasks are mandatory (i.e., must be included in the final schedule). However, the *MV-Pack* algorithm can also handle a combination of mandatory and optional tasks, in which some (or all) tasks are not required to be present in the solution. In this case, the original task set is modified in the following way: for each optional task we artificially add a new version with zero reward and zero energy and time requirements. We call this added version the zero version. A task selected in the final schedule at its zero version is equivalent to a task not selected for execution.

Experimental Results The *MV-Pack* algorithm was evaluated in [1]. Task sets with up to $N = 100$ tasks were simulated, with $V = 4$ versions for each task and using the Intel XScale architecture [24] as the power model. The algorithm was shown to be within 3% of the optimal where the comparison was possible. The running times of the algorithm were less than a millisecond even for 100 tasks. The reader is referred

to [1] for further evaluation details.

IV. Rechargeable RT Systems

The algorithm presented in the previous section determines which versions to select and at what speeds to run them so as to maximize the total value of a frame / hyperperiod given an energy budget. It was shown that the problem formulations for frame-based and periodic tasks are equivalent. We complete the solution in this section by showing how to distribute the available energy among frames. The same results apply to hyperperiods as well.

In Section IV-A we present a *static analysis* assuming a worst-case scenario, namely worst case execution times and energy requirements for task versions, minimum generated power E_{rec}, combined with worst-case battery recharging (α) and discharging (β) characteristics. The analysis is necessary, as the system has to be provably stable (i.e., the battery energy is at all times higher than E_{min}) in all possible scenarios. If the system is determined to be stable, the static component is also responsible for distributing the available energy among frames, as well as scheduling inside each frame based on the *MV-Pack* algorithm.

In Section IV-B we present the *dynamic component* dealing with cases where extra energy appears in the system, since worst-case scenario assumptions rarely happen in practice. We propose three dynamic energy reclaiming schemes that are shown to improve the energy usage and the overall system value. Both static and dynamic components are based on the *MV-Pack* algorithm. A quantitative evaluation of the proposed solutions is presented in Section IV-C.

A. Static Analysis

We present necessary and sufficient conditions for the stability of the system. Based on these conditions we show how to distribute the available energy among frames, assuming a worst-case scenario.

The static analysis starts by running the *MV-Pack* algorithm, assuming infinite available energy. After each successful version increase (i.e., reward/value increase), the intermediate solution is saved (i.e., the speed and version for each task, as well as the total energy consumption and reward are stored). There can be at most NV successful version increases, thus the space and time complexity become $O(N^2V)$. In practice, running times are still under a millisecond even for 100 tasks (total running time in a Unix system with a 850MHz Pentium III CPU and 256MB of RAM).

The i^{th} intermediate solution schedule, energy and reward are denoted by ϕ_S^i, ϕ_E^i and ϕ_R^i respectively. If a solution has a smaller reward and a higher energy than some other solution, it is eliminated from the saved solutions. This case can happen for artificial scenarios, although we did not encounter it during simulations. Notice that saved intermediate solutions are ordered by their rewards/energy in increasing order. Also, note that even with infinite energy it may be not possible to run all the task at their highest version due to real-time constraints. If the frame deadline is D, the number of frames to be executed during the recharging period T_r is $N_r = \frac{T_r}{D}$. Similarly, N_d denotes the number of frames to be executed during the discharging period, $N_d = \frac{T_d}{D}$.

Task rewards are expected to be proportional to their execution times. Energy increases more than linearly with the speed, while time decreases approximately linearly. Thus, it is to be expected that the frame reward increases less than linearly with the frame available energy. In other words, having a fixed amount of energy to be distributed among several frames, an equal energy partition is expected to maximize the total reward of the frames. Thus, we choose to distribute the energy equally among frames (recharging frame allocation may be different from discharging frame allocation due to recharging/discharging characteristics and battery capacity limitation). While artificial cases can be constructed where an equal energy partition is not optimal from a reward view point, a complete analysis is NP-hard.

The following theorem gives necessary and sufficient conditions for a system to be stable (i.e., the battery energy is at all times above E_{min}).

Theorem 1: A system is stable if and only if:
(i) $E_{rec} \geq N_r \phi_E^1 + \frac{N_d \phi_E^1}{\alpha\beta}$
(ii) $E_{max} - E_{min} \geq \frac{N_d \phi_E^1}{\beta}$.

Proof: The generated energy during the recharging period T_r must be enough to run all the frames with their minimum energy requirement ϕ_E^1. During recharging, the processing unit will use at least $N_r \phi_E^1$ energy. Due to the discharging loss β, at least $\frac{N_d \phi_E^1}{\beta}$ has to be stored for use during the discharging period T_d. Considering also the recharging loss α, the first condition becomes: $E_{rec} \geq N_r \phi_E^1 + \frac{N_d \phi_E^1}{\alpha\beta}$. This condition is necessary but not sufficient, as it could be the case that not all of the recharging energy E_{rec} can be used (for example, due to battery capacity limitation).

The second condition enforces that a fully charged battery holds enough energy to execute all discharging frames at their minimum energy consumption, even in worst case discharging conditions β. Thus, the second condition is: $E_{max} - E_{min} \geq \frac{N_d \phi_E^1}{\beta}$. ■

For a stable system, the actual schedules for the recharging and discharging frames are obtained as follows. We assume the system starts with a discharged battery (E_{min}) and the first recharging frame. The schedules for the recharging and discharging frames are the solutions ϕ_S^i and respectively ϕ_S^j that satisfy:

$$\text{maximize} \quad N_r \phi_R^i + N_d \phi_R^j \tag{11}$$

$$\text{subject to} \quad E_{max} - E_{min} \geq \frac{N_d \phi_E^j}{\beta} \tag{12}$$

$$E_{rec} \geq N_r \phi_E^i + \frac{N_d \phi_E^j}{\alpha\beta} \tag{13}$$

Determining the optimal values for i and j has complexity $O(NV)$, as there are at most NV stored solutions. A solution always exists for a stable system.

During discharging, the feasibility conditions (12) and (13) give the guarantee that the battery energy will never be less

than E_{min}. During recharging, we assume the processing unit relies directly on the solar power, while the unused power is stored in the battery with a worst-case loss α. For the pattern in Figure 3(a), the assumption is natural. For the pattern in Figure 3(b), the assumption will restrict the recharging period T_r to start and end at a non-zero power level. This power level will also be an upper bound on the power the processing unit is allowed to draw during recharging. For future work we plan to consider the case when the processing unit power may exceed the generated power.

B. Dynamic Energy Reclaiming

The static component is too conservative, as the system has to be stable even in worst-case conditions. The dynamic component handles cases when extra energy appears in the system. There are many ways to improve the system reward when worst-case scenarios do not happen. For example, whenever a task requires less energy than its worst-case, the remaining tasks inside its frame can benefit from the extra energy to improve their reward. However, this approach implies a considerable overhead as a new schedule needs to be constructed potentially every task completion. In terms of system reward the approach may also be inefficient, as it could be better to distribute the energy among frames.

Three dynamic policies are presented next. We assume that the battery charge can be examined with reasonable accuracy. By inspecting the battery charge at regular intervals, dynamic schemes will observe the deviation from the worst case scenario and redistribute the available energy among frames so as to maximize the system value. Frame boundaries provide such regular intervals for checking the battery level. Thus, the extra energy is not used in the current frame and the rescheduling overhead occurs only at frame boundaries.

The first two schemes (Proportional and Speculative) redistribute the energy among all remaining frames until the first recharging frame (at which moment, because this is a stable system, the battery level is known to be at least E_{min}). Also, the system reward can benefit most from this approach since reward increases less than linearly with the energy. A third dynamic policy (Greedy) uses the static schedule for frames, but gives all the extra energy to the next frame. Rescheduling decisions are still made only at frame boundaries.

a) Proportional: In this scheme, upon the completion of each frame, the available energy is redistributed equally among all recharging frames and equally among all discharging frames. A worst case scenario is assumed for the remaining frames and thus the system is guaranteed to be stable. However, the extra energy can now be used to improve the system reward while still guaranteeing its stability. When recharging frame k completes, aware of the current battery energy and the worst case remaining recharging energy, a new schedule ϕ_S^i is selected for the remaining $N_r - k$ recharging frames and a new schedule ϕ_S^j is selected for the N_d discharging frames so as to maximize $(N_r - k)\phi_R^i + N_d\phi_R^j$, while ensuring that the worst-case battery charge when the first discharging frame starts is enough to run all the discharging frames (i.e., is at least $E_{min} + \frac{N_d \phi_E^j}{\beta}$). Similarly, when discharging frame k completes, the available battery energy is equally distributed among the remaining $N_d - k$ frames so that the battery energy is at least E_{min} at the completion of the last discharging frame.

b) Speculative: The proportional scheme is too conservative as the worst-case scenario is assumed for all remaining frames. As has been shown in previous works [9], a better approach is to speculate about future energy consumption and schedule tasks accordingly, while ensuring that the system is stable even in worst-case conditions for all remaining frames. During discharging, the battery energy constantly decreases. At frame boundaries, the actual decrease in battery energy can be compared to the known worst-case. The ratio of actual consumption to worst-case consumption can be used to estimate consumption for future discharging frames. The ratio will be always less than 1, as the actual discharge loss is less than β and task energy consumptions will be less than their worst-case. The ratio for the next frame is then predicted as the average of such ratios for all frames in a history window.

During recharging, a similar ratio is computed at frame boundaries for estimating the energy accumulating in the battery.

c) Greedy: This scheme assigns all the available extra energy to the next frame with the constraint that enough energy is left to run the remaining frames according to the static schedule. Thus, the extra energy in the system is immediately used, unlike in the previous schemes.

The overhead of all dynamic schemes is $O(NV)$ at each frame completion. Simulation results presented in the next section quantitatively evaluate both the static and the dynamic components.

C. Experimental Results

Task sets with up to $N = 100$ tasks and $V = 4$ versions for each task were generated as described next. For each task, the execution time of the first version at minimum speed $t_{i,1}^1$ was randomly generated in the range $[10, 100]$. For the remaining versions, the running time at the first speed level was generated by the formula $t_{i,1}^k = t_{i,1}^{k-1} + \Delta_i^k$, where $\Delta_i^k \in [0.2 \cdot t_{i,1}^1, 1.2 \cdot t_{i,1}^1]$ was randomly generated for each task version. Next, $t_{i,j}^k$ was computed for all versions and all speed levels, inversely proportional with the speed ($t_{i,j}^k = t_{i,1}^k \frac{f_1}{f_j}$).

We simulated the Intel XScale architecture, with 5 speed levels. The running speeds and their corresponding voltages (from [24]) are shown in Table I. For the power consumption of a task version T_i^k at speed level j, we use the formula $P_{i,j}^k = a_i Voltage(j)^2 \frac{f_j}{f_M}$. Thus, the power is proportional with the normalized speed and the square of the voltage. a_i is an activity factor different for each task and identical for all versions of the same task, proportional with the dynamic switching caused by the task and randomly generated in the

TABLE I
INTEL XSCALE SPEED SETTINGS AND VOLTAGES

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8

range $[0.8, 1.2]$. The energy requirement $e_{i,j}^k$ is then computed as $e_{i,j}^k = P_{i,j}^k t_{i,j}^k$, that is the power multiplied with the time.

Task values of the first versions r_i^1 were generated randomly in the range $[10, 100]$. For the higher versions, task rewards were generated according to the formula $r_i^k = r_i^{k-1} + \delta_i^k$, where $\delta_i^k \in [0.2 \cdot r_i^1, 1.2 \cdot r_i^1]$ was randomly generated for each task version. Thus, observe that each version requires more time and more energy than the previous versions, but gives a higher reward; also, there is no assumption on the shape of the reward function (i.e., it is not necessarily convex, linear or concave). Experiments with different ranges for δ_i^k and Δ_i^k (such as $[10, 100]$), also with narrower or broader ranges for the activity factors a_i (such as $[0.2, 1.2]$) produced very similar results.

The frame deadline D and maximum energy E were generated so that there is not enough time and energy to run all highest versions at the highest frequency. The static analysis was then performed by running the *MV-Pack* algorithm to generate the intermediate solutions ϕ_S^i.

The values for E_{min}, E_{max} and E_{rec} were then generated as described next. E_{min} is 5% of the battery capacity E_{max}, which was generated so that the available energy during discharging (i.e., $E_{max} - E_{min}$) is at least $\frac{N_d \phi_E^1}{\beta}$ and less than $\frac{N_d \phi_E^s}{\beta}$, where s is the highest reward/energy intermediate solution (s is not always NV as deadlines can be missed). Thus, Equation (12) can be satisfied and not enough energy can be stored in the battery to run all tasks in all discharging frames at their highest version.

Similarly, E_{rec} was generated to be at least $(\frac{N_d}{\alpha\beta} + N_r)\phi_E^1$ and less than $(\frac{N_d}{\alpha\beta} + N_r)\phi_E^s$. Equation (13) can be satisfied, while the recharging energy cannot support all tasks at their highest version. We thus ensure that there is a solution, but not enough energy to run all the most valued task versions. We also ensured that the processing unit power during recharging periods is less than the worst-case solar power $\frac{E_{rec}}{T_r}$. Note that when the system has a large amount of energy (e.g. large N_r or large E_{max} and E_{rec}), the problem is unrealistic and uninteresting. The solution is also trivial, namely runs all tasks at the highest energy/reward solution s.

The static schedule was created as the solution to (11)-(13). The system was then simulated, starting with the first recharging frame and a discharged (E_{min}) battery. The dynamic behavior is simulated as follows: with a probability of 50% tasks required their worst-case time and energy and with 50% probability their actual running time (and thus energy requirement) was between 50% to 100% of the worst-case. Thus, on average frames require 87.5% of their worst-case time and energy. We considered a worst-case α and β of 0.9. The actual α and β values were generated for each recharging/discharging frame in the range $[0.9, 1]$.

Note that the worst-case generated energy is $\frac{E_{rec}}{N_r}$ for each recharging frame, corresponding to the pattern in Figure 3(a). To simulate a deviation from the worst-case, we added an extra energy of up to 20% for each recharging frame (i.e., the generated energy was in the range $[\frac{E_{rec}}{N_r}, 1.2\frac{E_{rec}}{N_r}]$ for each recharging frame). To simulate the pattern in Figure 3(b), we simulated the deviation from the worst-case as a sinusoidal function with a maximum of 20% in the middle of the recharging period.

A comparison between the static and dynamic schemes for the recharging pattern in Figure 3(a) is presented in Figures 5(a) and 5(b), showing the frame reward and battery energy at the completion of each frame, respectively. Each point in the graph is the average of 1000 experiments ($N = 50$ tasks, $N_r = 50$, $N_d = 100$). The overhead of redistributing the energy was typically in the range of microseconds to dozens of microseconds for each frame.

As seen in Figure 5(b), the static scheme does not react to changes in the available energy and part of the recharging energy is wasted: the battery becomes fully charged before the recharging period ends (e.g., around frame 330). The dynamic schemes generally take advantage of all the recharging energy. In terms of frame rewards, all dynamic schemes outperform the static. Among the dynamic schemes the worst performance is that of the proportional, which is too conservative in assuming a worst-case scenario for the remaining frames. As a consequence, the available energy is too slowly redistributed, resulting in the pattern shown in Figure 5(a), with frame rewards slowly increasing and extra energy accumulating towards the end of recharging and discharging periods.

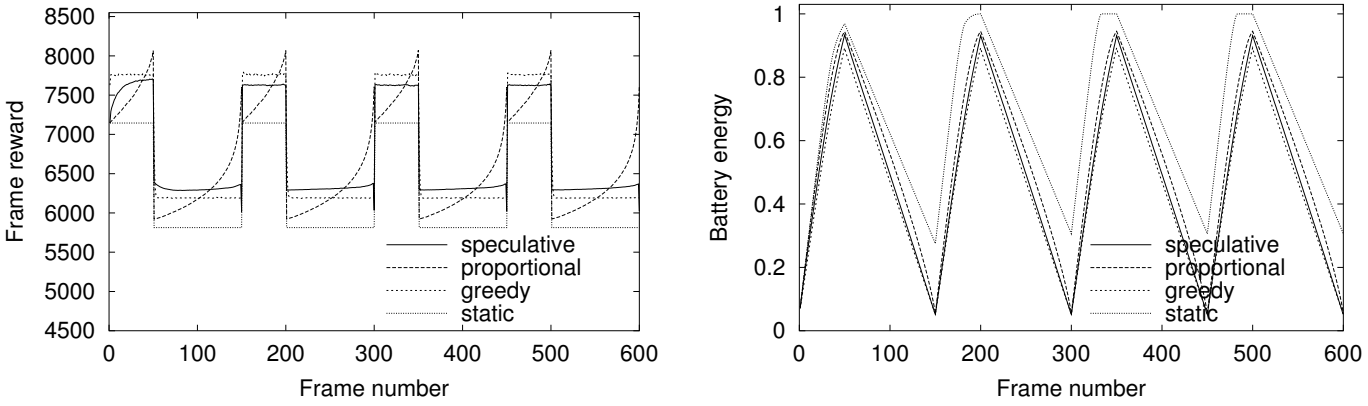
The speculative scheme returned higher total rewards than the greedy in 82% of the experiments. The greedy scheme ensures each frame has a reward higher than or equal to the static schedule reward. The speculative scheme only ensures that the remaining frames are feasible (i.e., minimum reward) and also speculates that tasks will not take their worst-case time and energy. For this reason it can be more aggressive and, on average, allocates more energy than the greedy policy to the discharging frames, and less energy to the recharging frames. Reducing the energy gap between the discharging and the recharging frames generally results in an improved total system value.

Figures 6(a) and 6(b) show simulation results for the pattern in Figure 3(b). The greedy scheme uses the extra energy immediately, with frame rewards following (on average) the sinusoidal shape of the extra energy during the recharging period. The speculative scheme choses instead to reserve some of the extra energy for the discharging period, resulting in a higher system value in 83% of the experiments.

V. Conclusions

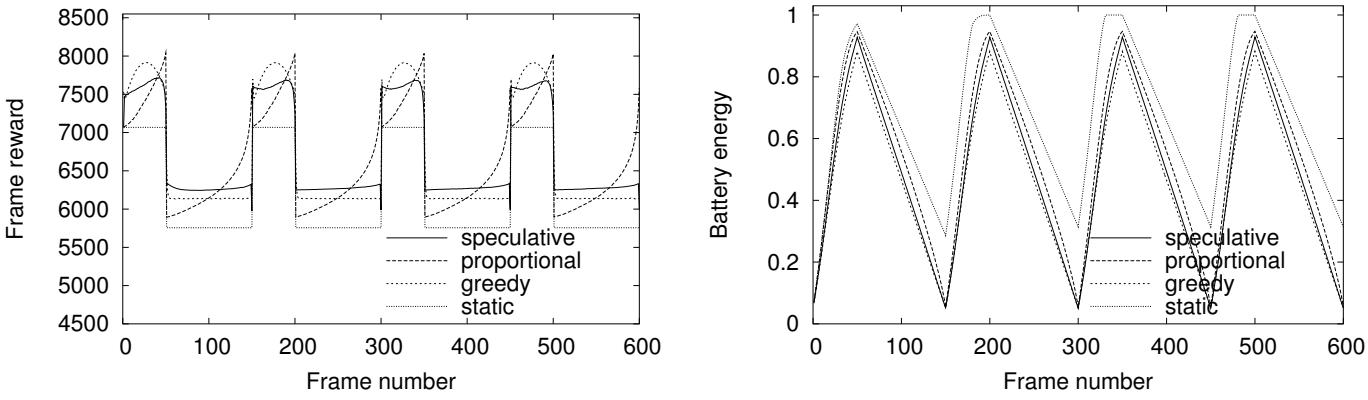
We presented an algorithm for the problem of maximizing the total value of a system executing periodic or frame-based real-time tasks with an allotted fixed energy budget. Tasks are assumed to have multiple versions, each with known rewards and worst-case requirements. The algorithm selects the most important versions and determines their execution speeds given the constraints. For a system with a rechargeable battery and a rechargeable energy source (such as a solar panel) we proposed a static solution that determines how to best distribute the energy so as to maximize the total value of the system.

The static analysis is necessary, as the system has to be stable (i.e., the battery is never exhausted) in all scenarios. However, the static solution by itself is too conservative, as



(a) (b)

Fig. 5. Average of 1000 simulations: (a) Frame rewards (b) Battery energy (for the pattern in Figure 3(a))



(a) (b)

Fig. 6. Average of 1000 simulations: (a) Frame rewards (b) Battery energy (for the pattern in Figure 3(b))

it assumes a worst-case scenario for task execution times and battery recharging/discharging. Three dynamic policies take advantage of the extra energy in the system (for example due to task actual execution times being less than their worst-case, or energy loss due to battery recharging/discharging characteristics being less than the worst-case loss). At the completion of each frame, the dynamic policies observe the actual battery charge and redistribute the available energy among the remaining frames. All dynamic solutions were shown to make better use of the available energy, resulting in a higher total system value.

Future work will address the case of slack reclamation inside a frame, speed change overheads, and will incorporate recharging patterns into the proposed policies.

REFERENCES

[1] C. Rusu, R. Melhem, and D. Mossé, "Multi-version scheduling in rechargeable energy-aware real-time systems," in *Proceedings of the 15^{th} Euromicro Conference on Real-Time Systems (ECRTS'03)*, Porto, July 2003.

[2] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," in *Proceedings of the 38^{th} Design Automation Conference (DAC'01)*, Las Vegas, NV, June 2001.

[3] F. Yao, A. Demers, and S. Shankar, "A scheduling model for reduced cpu energy," *IEEE Annual Foundations of Computer Science*, pp. 374–382, 1995.

[4] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors," in *Proceedings of the 19^{th} IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Dec. 1998.

[5] C. M. Krishna and Y. H. Lee, "Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems," in *Proceedings of the 6^{th} IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D. C., May 2000.

[6] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Proceedings of the 13^{th} Euromicro Conference on Real-Time Systems (ECRTS'01)*, Delft, Netherlands, June 2001.

[7] ——, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proceedings of the 22^{nd} IEEE Real-Time Systems Symposium (RTSS'01)*, 2001.

[8] F. Gruian, "Hard real-time scheduling using stochastic data and dvs processors," in *Proceedings of International Symposium on Low Power Electronics and Design*, 2001, pp. 46–51.

[9] D. Mossé, H. Aydin, B. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," in *Workshop on Compilers and Operating Systems for Low Power (COLP'00)*, Philadelphia, PA, Oct. 2000.

[10] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design and Test of Computers*, vol. 18, pp. 20–30, Mar. 2001.

[11] 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*, vol. 24, pp. 58–68, May 1991.

[12] J. K. Dey, J. Kurose, and D. Towsley, "On-line scheduling policies for a class of iris (increasing reward with increasing service) real-time tasks," *IEEE Transactions on Computers*, vol. 45, pp. 802–813, July 1996.

[13] C. M. Krishna and K. G. Shin, *Real-time Systems*. New York: Mc Graw-Hill, 1997.

[14] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, "Optimal reward-based scheduling for periodic real-time tasks," in *Proceedings of the 20^{th} IEEE Real-Time Systems Symposium (RTSS'99)*, Phoenix, Dec. 1999.

[15] R. Rajkumar, C. Lee, J. P. Lehoczky, and D. P. Siewiorek, "A resource allocation model for qos management," in *Proceedings of 18^{th} IEEE Real-Time Systems Symposium (RTSS'97)*, Dec. 1997.

[16] ——, "Practical solutions for qos-based resource allocation problems," in *Proceedings of 19^{th} IEEE Real-Time Systems Symposium (RTSS'98)*, Dec. 1998.

[17] D. Kang, S. P. Crago, and J. Suh, "A fast resource synthesis technique for energy-efficient real-time systems," in *Proceedings of the 23^{rd} IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, Dec. 2002.

[18] P. M. Shriver, M. B. Gokhale, S. D. Briles, D. Kang, M. Cai, K. McCabe, S. P. Krago, and J. Suh, "A power-aware, satellite-based parallel signal processing scheme," in *Power Aware Computing*. New York: Kluwer Academic Press, 2002.

[19] B. D. Guenther, "Aided and automatic target recognition based upon sensory inputs from image forming systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, pp. 1004–1019, Sept. 1997.

[20] G. C. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in *Proceedings of the 19^{th} IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, Dec. 1998.

[21] A. Allavena and D. Mossé, "Scheduling of frame-based embedded systems with rechargeable batteries," in *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS'01)*, 2001.

[22] J. L. Stone. Photovoltaics: Unlimited electrical energy from the sun. [Online]. Available: http://www.nrel.gov/research/pv/docs/pvpaper.html

[23] C. Rusu, R. Melhem, and D. Mossé, "Maximizing the system value while satisfying time and energy constraints," in *Proceedings of the 23^{rd} IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, Dec. 2002.

[24] [Online]. Available: http://developer.intel.com/design/intelxscale/benchmarks.htm