

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

Cosmin Rusu, Rami Melhem, Daniel Mossé
Computer Science Department, University of Pittsburgh
Pittsburgh, PA 15260
(*rusu,melhem,mosse*)@cs.pitt.edu

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 propose a solution that allows the device to run the most valuable task versions while still meeting all deadlines and without depleting the energy. Assuming that the battery is rechargeable, we also propose (a) a static solution that maximizes the system value assuming a worst-case scenario (i.e., worst-case task execution times); and (b) 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.

1 Introduction

Power management is a critical design factor for embedded systems that rely on their own power source (battery). Thus, it would appear as though the lifetime of the device is ultimately depending on the battery storage capacity. However, this is not necessarily the case. One solution is to replenish the battery by scavenging the existing energy in the environment, such as the NASA/JPL Mars rover, which relies on both a non-rechargeable battery source and a solar panel [14]. In our work we also assume that the battery is rechargeable. During the day real-time tasks are executed at the same time the battery is recharging. At night, the system relies entirely on the battery energy acquired during the day.

The rechargeable energy is 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 (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. In this paper we concentrate on systems where the processing unit is an important energy consumer.

However, even for such complex systems our scheme that attempts to increase the value of the system, has three contributions. First, 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). This provides information to the designer that either there is plenty of energy in the system or that voltage scaling is not beneficial. Second, the lifetime of the system assuming worst-case conditions can be computed. The answer can be that the system is stable (i.e., at all times there is energy left in the battery). Third, and most important, our work is 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.*

A large number of 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.

The rest of the paper is organized as follows: We first present related work. Task and recharging models are explained in Section 2. We present an algorithm for energy/aware task selection in Section 3. Based on this algorithm, the solution is completed in Section 4 with a static

*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.

analysis and three dynamic policies, followed by experimental results. We conclude the paper in Section 5.

1.1 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. [21] 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 [9]. Systems with two (discrete) voltage levels and periodic hard real-time tasks are analyzed in [12]. An optimal static solution for periodic task sets with different power characteristics is given in [2]. Slack management techniques are explored in [3, 7, 15, 19]. 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 Increased Service) models. In the IC model [13], 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 [5, 11] 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 [4]. The case for discrete reward functions (or step reward functions) with no reward for partial execution was shown to be NP-hard [13]. In the QRAM model (QoS-based resource allocation model) [16, 17] reward functions are in terms of utilization of resources. A solution was proposed for one resource with multiple QoS dimensions [16] and a particular audio-conferencing application with two resources and one QoS dimension was analyzed in [17]. One work recently published combined the three constraints [10], 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 [20]. 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 [8]. Task versions can result from different algorithms, as well as from the same ap-

plication with different input arguments, such as encoding/decoding at different rates, low/high quality compression schemes, low/high resolution image processing, etc.

Rechargeable systems remain generally unexplored. A solution for the Mars Pathfinder rover that makes best use of the available recharging energy is presented in [14]. For frame-based systems with a rechargeable battery, a solution is presented in [1] 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).

In [18], a solution was proposed for frame-based embedded systems that maximizes the system value, while still being aware of time and energy constraints. The task model in [18] is optional single-version (i.e., one version for each task and each task is not necessarily selected in the final schedule). We extend the work in [18] in four directions: (a) *periodic tasks*, (b) *multiple versions of a task*, (c) *mandatory/optional tasks*, and (d) *rechargeable batteries*.

2 Task Set and Recharging Models

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.

2.1 Multiple-version Task Models

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), $\{f_1, f_2, \dots, f_M\}$. 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, 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 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 $\min(M, N)$. We assume the overhead of $\min(M, N)$ speed changes is negligible¹ compared to the deadline D , or that it was already subtracted from D (similar for the energy cost overhead). 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.

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_{ij}^k and e_{ij}^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.

2.2 Recharging model

We assume a periodic recharging model in which a relatively long recharging time T_r (hundreds or thousands of frames / hyperperiods) is followed by a relatively long discharging time T_d on which the system runs entirely on the battery energy accumulated during recharging. The number of frames dispatched during the recharging time (also referred to as recharging frames) is $N_r = \frac{T_r}{D}$ and the number of frames dispatched while running entirely on the battery (discharging frames) is $N_d = \frac{T_d}{D}$. For the periodic task model, $N_r = \frac{T_r}{T_{LCM}}$ and $N_d = \frac{T_d}{T_{LCM}}$.

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

The battery has a maximum capacity, denoted by E_{max} . Our goal is to have a stable system in which the battery energy is at all times within the limits $[E_{low}, E_{high}]$. From a theoretical point of view E_{low} can be zero and E_{high} can be the battery capacity E_{max} . However, it could be more practical to have a safe margin, such as $E_{low} = 5\% \cdot E_{max}$ and $E_{high} = 95\% \cdot E_{max}$, for which reason we prefer the $[E_{low}, E_{high}]$ notation. For example, it could be considered safe to have at all times at least E_{low} battery energy. Similarly, the recharging pattern when the battery charge is above E_{high} can be slightly unpredictable, while it is approximately linear otherwise [1].

Unlike [1], we make no assumptions about the recharging pattern, which can be linear or not. We only assume that the amount of recharging energy that can be accumulated during the recharging period T_r is known to be at least E_{rec} . All incoming energy when the battery is fully recharged is lost.

2.3 Problem Definition

Our goal is to determine for each frame / hyperperiod how much energy to allocate so that the system is stable (i.e., the battery energy can never be less than E_{low}) provided that at least E_{rec} energy is received every recharging period and provided that for all task versions the 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 that 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 / hyperperiod for multiple-version task sets with an energy budget. The total value of a frame / hyperperiod 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 / hyperperiod 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 that to:

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

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

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

$$v_i \in \{1, 2, \dots, V\} \quad (4)$$

$$s_i \in \{1, 2, \dots, M\} \quad (5)$$

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:

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

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

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

$$v_i \in \{1, 2, \dots, V\} \quad (9)$$

$$s_i \in \{1, 2, \dots, M\} \quad (10)$$

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. 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 $\frac{T_{LCM}}{D_i}$.

A solution for the problem defined by Equations (1)-(5) or (6)-(10) is presented 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 4. The problem was shown in [18] 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.

3 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 [18] for the case of single version frame-based task sets. The algorithm is presented next, followed by a quantitative evaluation.

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, provided that the energy budget is associated with T_{LCM} .

3.1 The MV-Pack Algorithm

The flowchart of the algorithm is presented in Figure 1. The three major components (add task, increase speed and in-

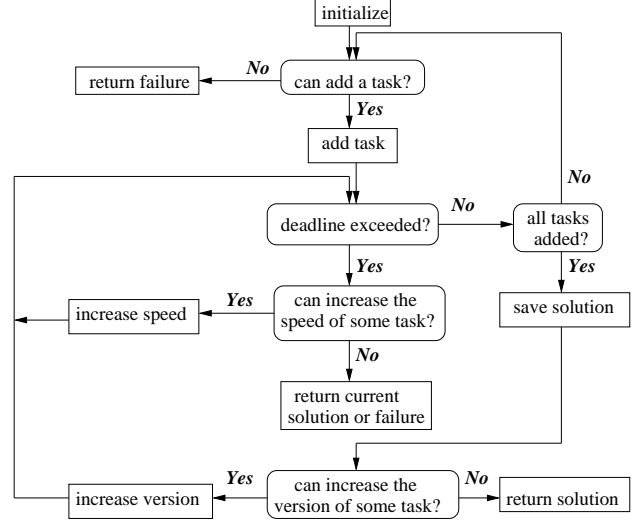


Figure 1: Flowchart of *MV-Pack*

crease version) are described next. We denote by *time* and *energy* 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 $\frac{r_i^1}{t_{i,1}^1 \cdot e_{i,1}^1}$.

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 ($energy + e_{i,1}^{v_i+1} - e_{i,1}^{v_i} \leq E$).
- 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 $\frac{r_{i,1}^{v_i+1}}{t_{i,1}^{v_i+1} \cdot e_{i,1}^{v_i+1}}$.

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 $(M - 1) \cdot V$ 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 and mandatory tasks Observe that in the problem definition (Equations(1)-(5)) all tasks are mandatory (i.e., must be included in the final schedule). However,

Table 1: 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

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.

3.2 Experimental Results

We simulated the Intel XScale architecture, with 5 speed levels. The running speeds and their corresponding voltages (from [6]) are shown in Table 1. Each task has $V = 4$ versions. 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 ($t_{i,j}^k = t_{i,1}^k \cdot \frac{f_1}{f_j}$).

For the power consumption of a task version T_i^k at speed level j , we use the formula $P_{i,j}^k = a_i \cdot Voltage(j)^2 \frac{f_j}{f_M}$. 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 range $[0.8, 1.2]$. The energy requirement $e_{i,j}^k$ is then computed as $e_{i,j}^k = P_{i,j}^k \cdot t_{i,j}^k$, 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 deadline D and maximum energy E are generated by the formulas $D = \sum_{i=1}^N t_{i,s_i}^{v_i}$ and respectively $E =$

$\sum_{i=1}^N e_{i,s_i}^{v_i}$, where the indices for speed $s_i \in \{1, 2, \dots, M\}$ and for value $v_i \in \{1, 2, \dots, V\}$ are randomly picked for each task $i \in \{1, 2, \dots, N\}$. We denote by SR_{min} the minimum system reward that can be achieved for a given task set, $SR_{min} = \sum_{i=1}^N r_i^1$. Similarly, SR_{max} denotes the maximum reward that can be achieved, $SR_{max} = \sum_{i=1}^N r_i^V$. Observe that if each task i runs at the version v_i and the speed level s_i used to generate D and E , the energy and deadlines are not exceeded and the system reward is $SR_{gen} = \sum_{i=1}^N r_i^{v_i}$.

Since it is impractical to compute the optimal solution, we will compare the performance of *MV-Pack* with SR_{min} , SR_{max} and SR_{gen} . Figure 2a shows the comparison for task sets of 10 to 100 tasks, where SR_{gen} , SR_{max} and the reward returned by the algorithm are normalized to SR_{min} . Each point is the average of 1000 simulation runs. In all experiments, *MV-Pack* returned a system value higher than SR_{gen} and close to SR_{max} . Note that SR_{max} is an upper bound on the optimal solution, not the optimal solution itself. In most cases SR_{max} cannot be achieved without exceeding the deadline or energy restrictions. For most graph points *MV-Pack* used more than 99% of the available energy; the smallest value is 94%. Similarly, *MV-Pack* used at least 97% of the available time.

The system value can be improved even more with the following enhancement: if there is not enough energy to pack tasks within the deadline, the search continues (whereas *MV-Pack* would stop) by restoring the schedule just before attempting the last version increase. Furthermore, the task that caused the energy problem when increasing its version number has its version number frozen (that is, the higher version will never be considered again). The algorithm continues then as usual by selecting a task to increase its version from the remaining set of tasks. The upper bound on the running time becomes $O(MVN^2 \log N)$, although in practice the running time does not significantly increase. The enhanced *MV-Pack* returns slightly higher rewards in 14% of the simulations with 10 tasks and in 43% of the simulations with 100 tasks. However, the increase in the average normalized reward is almost unnoticeable. The actual running times of both algorithms were less than a millisecond for all experiments, on a 850 MHz Pentium III.

Unfortunately, the exponential nature of an optimal algorithm makes it impossible to compute the absolute error for high values of N . There is experimental evidence, however, that the absolute errors do not increase (rather, they actually decrease) as the number of tasks increases. For example, in Figure 2b, where we simulated single version task sets with 5 to 14 tasks under limited time and energy conditions, we can see this trend. In the figure, each point is the average error of 100 runs. The optimal solution was computed by exhaustive search and the average error is relative to the optimal system value. The worst performance is when there is

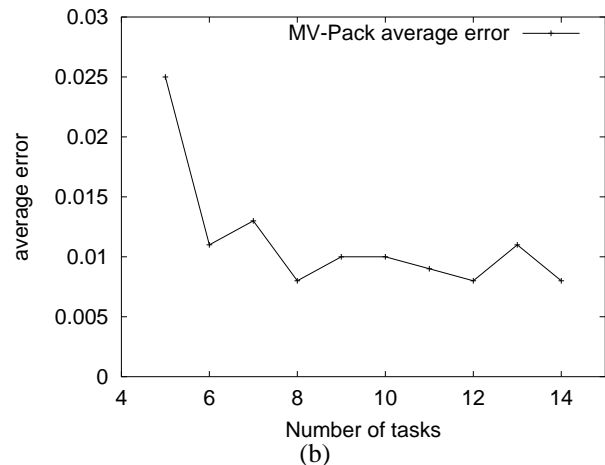
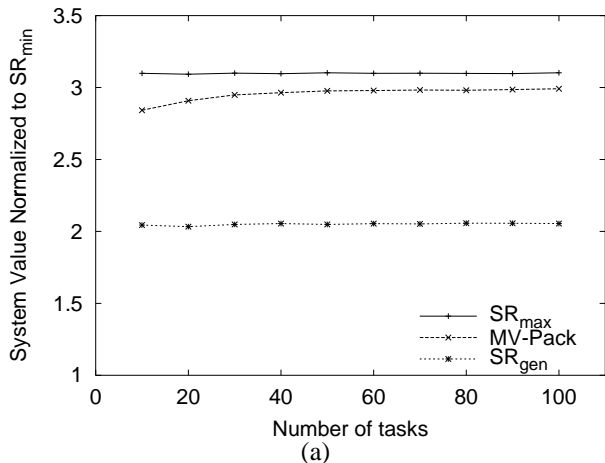


Figure 2: (a) Evaluation of *MV-Pack* for varying number of tasks (b) *MV-Pack* vs. optimal system value for small number of single-version tasks

little slack in the system combined with a reduced amount of energy. In this case even the optimal can select only a few tasks; if the algorithm does not pick exactly the same tasks as the optimal, the error is likely to increase.

In order to avoid the complexity of finding the optimal solution to evaluate our algorithms, we designed another experiment in which we constructed sets of tasks with known optimal solutions, and ran our algorithms against those task sets. The task sets were constructed as follows: the deadline D was set to $D = \sum_{i=1}^N t_{i,s_i}^V$ and the maximum energy E was set to $E = \sum_{i=1}^N e_{i,s_i}^V$, where $s_i \in \{1, 2, \dots, M\}$ was randomly generated for each task. Thus, if each task i runs at speed level s_i and version V , all tasks are schedulable and the optimal reward is simply SR_{max} . We ran 1000 simulations on task sets with 20 to 100 tasks and 4 versions for each task. We do not show a graph for the results, because the optimal solution was returned in all simulation runs.

4 Rechargeable RT Systems

The algorithm presented in the previous section determines which versions to select and at what speeds to run them so that 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 between frames. The same results apply to hyperperiods as well.

First, a *static analysis* is performed assuming a worst-case scenario, namely worst case execution times and energy requirements for task versions as well as minimum recharging energy E_{rec} . The analysis is necessary, as the system has to be provably stable (i.e., the battery energy is at all times higher than E_{low}) in all possible scenarios. If the system is determined to be stable, the static component is also responsible for distributing the available energy between frames, as well as scheduling inside each frame based on the *MV-Pack* algorithm.

The *dynamic component* deals with cases where extra energy appears in the system, due to tasks actual execution times and energy requirements being smaller than the worst-case or the recharging energy being larger than E_{rec} . Even if the worst-case scenario is true at all times, extra energy can slowly accumulate in time due to limitations of the static component (i.e., the available energy is not entirely used). 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 4.3.

4.1 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 between 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 (850MHz Pentium III).

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.

We choose to distribute the available energy equally between all recharging frames and equally between all discharging frames, for reasons described next. 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. 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 system is stable if and only if both the following conditions are satisfied:

- The recharging energy is enough to run all frames with their minimum energy requirement. That is, $E_{rec} \geq (N_r + N_d)\phi_E^1$ (N_r and N_d are the number of recharging and discharging frames). 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).
- If not all the energy can be stored, it means that at the end of the recharging period the battery is fully charged. Thus, the second condition is that starting with a battery fully charged all discharging frames can be executed without exhausting the energy before the recharging period starts. Thus, the second condition is: $E_{high} - E_{low} \geq N_d \cdot \phi_E^1$.

For a stable system, the actual schedules for the recharging and discharging frames are obtained as described next. We assume the system starts with a fully charged battery (E_{high}) and the first discharging frame. We require that each first discharging frame starts with a fully charged battery, in other words the system energy does not decrease across recharging periods. The schedules for the discharging and recharging frames are the solutions ϕ_S^i and ϕ_S^j that satisfy:

$$\text{maximize} \quad N_d \cdot \phi_R^i + N_r \cdot \phi_R^j \quad (11)$$

$$\text{subject to} \quad E_{high} - E_{low} \geq N_d \cdot \phi_E^i \quad (12)$$

$$E_{rec} \geq N_d \cdot \phi_E^i + N_r \cdot \phi_E^j \quad (13)$$

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.

Inside a frame, tasks are scheduled so that to minimize the number of speed changes. Recharging/discharging patterns play a secondary role within a frame, as will be argued next. During discharging, the feasibility conditions

give the guarantee that the battery energy will not be less than E_{low} . During recharging, simple schemes can ensure that the recharging energy is efficiently used. If at the start of a recharging frame the battery level is low (say below $\frac{E_{high}-E_{low}}{2}$), tasks with low power requirements are run first. Thus, inside a recharging frame the battery is first recharging and E_{low} cannot be reached. Similarly, if the battery level is high, task with high power requirements are run first in a recharging frame. The battery level is thus at all times within the acceptable limits $[E_{low}, E_{high}]$, except for the case when E_{rec} is too high to use all of the recharging energy. An optimal in-frame scheduler can be developed assuming tasks are preemptable and recharging and consumption patterns are known [1]. However, minimizing the number of preemptions is believed to be NP-hard [1]. In our case, due to the granularity considered (hundreds or thousands of recharging frames) the recharging rates and consumption patterns have a reduced importance and we will not further address such issues.

4.2 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, such as task computation and energy requirements being less than the estimated worst-case.

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 between frames.

Three dynamic policies are presented next. The first two schemes redistribute the energy between all remaining frames until the first discharging frame (at which moment, because this is a stable system, the battery level is E_{high}). Thus, the extra energy is not used in the current frame and the rescheduling overhead occurs only at frame boundaries. Also, the system reward can benefit most from this approach since reward increases less than linearly with the energy. A third dynamic policy 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.

Proportional In this scheme, upon the completion of each frame, the available energy is redistributed equally between all discharging frames and equally between all recharging 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 discharging frame k completes, aware of the current battery energy $E_{battery}$, a new schedule ϕ_S^i is selected for the remaining $N_d - k$ discharging frames and a new schedule ϕ_R^j is selected for the N_r recharging frames so that to maximize $(N_d - k)\phi_R^i + N_r \cdot \phi_S^j$ subject to $E_{battery} - E_{low} \geq (N_d - k)\phi_E^i$ (i.e., there is enough energy in the battery to run all remaining discharging frames) and $E_{battery} - (N_d - k)\phi_E^i + E_{rec} - N_r \cdot \phi_E^j \geq E_{high}$ (i.e., the battery is fully charged when the last recharging frame completes). Notice that for $k = 0$ and $E_{battery} = E_{high}$ Equations (11)-(13) are derived.

Similarly, when recharging frame k completes, the available energy is equally distributed between the remaining $N_r - k$ frames so that the battery energy is at least E_{high} at the completion of the last recharging frame.

Speculative The proportional scheme is too conservative as a worst-case scenario is assumed for all remaining frames. A better approach is to predict future energy consumption based on previous history and schedule tasks accordingly. For each frame we compute the ratio of its actual to worst-case energy consumption. The ratio for the next frame is then predicted as the average of such ratios for all frames in the history window (the last 10 frames were considered in the experiments). The amount of recharging energy for each recharging frame is similarly predicted.

While tasks are scheduled assuming predicted energy consumptions, a feasibility test is also performed to ensure that the system is stable even in worst-case conditions for all remaining frames.

Greedy The last scheme we propose greedily assigns all the available 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.

4.3 Experimental Results

Task sets with 10 to 100 tasks were generated as described in Section 3.2. We used the Intel XScale model and $V = 4$ versions for each task (all tasks are mandatory, i.e., must be selected in each frame). The static analysis was performed first by running the *MV-Pack* algorithm to generate the intermediate solutions ϕ_S^i .

The values for E_{low} , E_{high} and E_{rec} were then generated as described next. E_{high} and E_{low} were generated as 5% and respectively 95% of the battery maximum capacity E_{max} , computed so that the available energy during discharging (i.e., $E_{high} - E_{low}$) is in the range $[N_d \cdot \phi_E^1, N_d \cdot$

$\phi_E^s]$, 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 then generated in the range $[(N_d + N_r)\phi_E^1, (N_d + N_r)\phi_E^s]$. Equation (13) can be satisfied, while the recharging energy cannot support all tasks at their highest version. We, for our experiments, thus ensure that there is a solution, but not enough energy to run all the most valued task versions.

The static schedule was created as the solution to (11)-(13). The system was then simulated, starting with the first discharging frame and a fully charged (E_{high}) 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.

For each recharging frame, the recharging energy was 0 to 20% higher than the average $\frac{E_{rec}}{N_r}$. For the speculative scheme, the ratio of actual-case to worst-case frame energy consumption was predicted as the average ratio of the last 10 frames. Similarly, the amount of recharging energy was predicted to be the average of the last 10 recharging frames. A comparison between the static and dynamic schemes is presented in Figures 3a and 3b. The frame reward and battery energy at the completion of each frame are shown. Each point in the graph is the average of 1000 experiments ($N = 50$ tasks and $N_r = N_d = 100$).

As seen in Figure 3b, 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. 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 3a, 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 87% 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

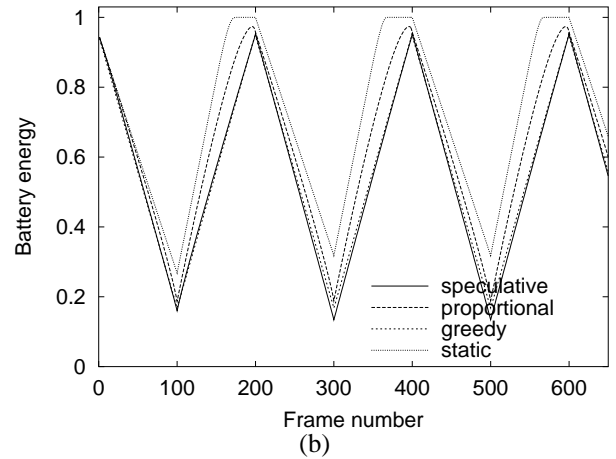
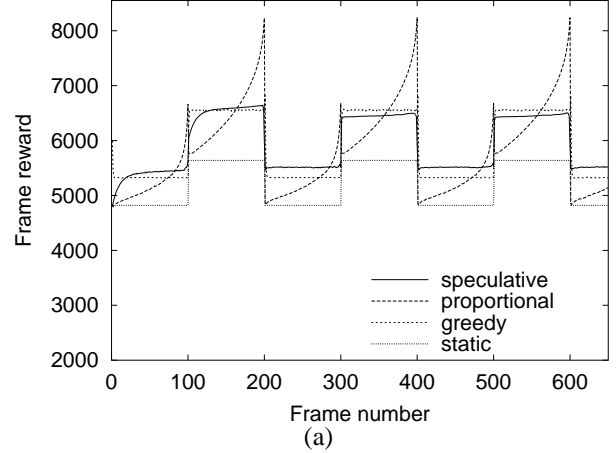


Figure 3: Average of 1000 simulations: (a) Frame rewards (b) Battery energy

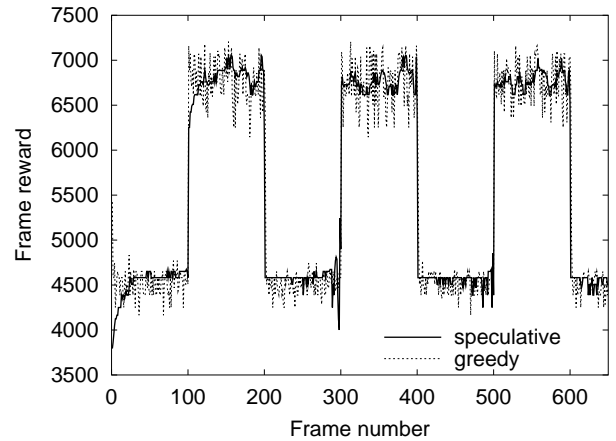


Figure 4: Greedy vs. speculative for a typical simulation

between the discharging and the recharging frames generally results in an improved total system value.

A comparison between the greedy and speculative schemes for a typical simulation (the first of the 1000 experiments) is shown in Figure 4. The greedy policy exhibits a lot of variation between adjacent frames, as the extra energy (which varies from frame to frame) is given only to the next frame. The speculative scheme is more stable as a result of equally distributing the available energy.

5 Conclusions

We presented the algorithm *MV-Pack* for the problem of maximizing the total value of a frame (frame-based task sets) or hyperperiod (periodic task sets) with time and energy constraints. Tasks are assumed to have multiple versions, each with different rewards, time and energy requirements. The algorithm selects the most important versions and determines their execution speeds given the constraints. In the context of rechargeable systems we proposed a static solution (based on *MV-Pack*) that determines how much energy to allocate to each frame so that 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 is too conservative, as it assumes a worst-case scenario for task execution times and recharging energy. 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). At the completion of each frame the dynamic policies redistribute the available energy between the remaining frames to avoid re-scheduling at every frame boundary. All dynamic solutions were shown to make better use of the available energy, resulting in a higher total system value. The overhead of redistributing the energy is typically in the range of microseconds to dozens of microseconds for each frame.

Future work will address the case of slack reclamation inside a frame, speed change overheads, as well as thresholds for triggering the redistribution of energy between frames. Finally, we plan to consider task dependencies as provided by a task graph.

References

- [1] A. Allavena and D. Mossé: Scheduling of Frame-based Embedded Systems with Rechargeable Batteries, *Workshop on Power Management for Real-Time and Embedded Systems (in conjunction with RTAS'01)*, 2001
- [2] H. Aydin, R. Melhem, D. Mossé, P.M. Alvarez: Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics, *Proceedings of the 13th Euromicro Conference on Real-Time Systems (ECRTS'01)*, Delft, Netherlands, June 2001
- [3] H. Aydin, R. Melhem, D. Mossé and P. M. Alvarez: Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems, *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01)*, 2001
- [4] H. Aydin, R. Melhem, D. Mossé, P. M. Alvarez: Optimal Reward-Based Scheduling for Periodic Real-Time Tasks, *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99)*, Phoenix, December 1999
- [5] 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* 45(7):802-813, July 1996
- [6] <http://developer.intel.com/design/intelxscale/benchmarks.htm>
- [7] F. Gruian: Hard Real-Time Scheduling Using Stochastic Data and DVS Processors, *Proceedings of International Symposium on Low Power Electronics and Design*, pp. 46-51, 2001
- [8] B. D. Guenther: Aided and Automatic Target Recognition Based upon Sensory Inputs from Image Forming Systems, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1004-1019, 1997
- [9] I. Hong, G. Qu, M. Potkonjak and M. Srivastava: Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors, *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS'98)*, Madrid, December 1998
- [10] D. Kang, S. P. Crago and J. Suh: A Fast Resource Synthesis Technique for Energy-Efficient Real-Time Systems, *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, December 2002
- [11] C. M. Krishna and K. G. Shin: Real-time Systems, *Mc Graw-Hill, New-York* 1997
- [12] C. M. Krishna and Y. H. Lee: Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems, *Proceedings of the 6th IEEE Real-Time Technology and Applications Symposium (RTAS'00)*, Washington D. C., May 2000
- [13] J. W.-S. Liu, K.-J. Lin, W.-K. Shih, A. C.-S. Yu, C. Chung, J. Yao, W. Zhao: Algorithms for scheduling imprecise computations, *IEEE Computer*, 24(5):58-68, May 1991
- [14] J. Liu, P. Chou, N. Bagherzadeh, F. Kurdahi: Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems, *Proceedings of the 38th Design Automation Conference (DAC'01)*, Las Vegas, NV, June 2001
- [15] D. Mossé, H. Aydin, B. Childers, R. Melhem: Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications, *Workshop on Compilers and Operating Systems for Low Power (COLP'00)*, Philadelphia, PA, October 2000
- [16] R. Rajkumar, C. Lee, J. P. Lehoczky, D. P. Siewiorek: A Resource Allocation Model for QoS Management, *Proceedings of 18th IEEE Real-Time Systems Symposium (RTSS'97)*, December 1997
- [17] R. Rajkumar, C. Lee, J. P. Lehoczky, D. P. Siewiorek: Practical Solutions for QoS-based Resource Allocation Problems, *Proceedings of 19th IEEE Real-Time Systems Symposium (RTSS'98)*, December 1998
- [18] C. Rusu, R. Melhem and D. Mossé: Maximizing the System Value while Satisfying Time and Energy Constraints, *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Austin, December 2002
- [19] D. Shin, J. Kim and S. Lee: Intra-Task Voltage Scheduling for Low-Energy Hard Real-Time Applications, *IEEE Design and Test of Computers*, 18(23):20-30, March 2001
- [20] P. M. Shriver, M. B. Gokhale, S. D. Briles, D. Kang, M. Cai, K. McCabe, S. P. Krago, J. Suh: A Power-Aware, Satellite-Based Parallel Signal Processing Scheme, *Power Aware Computing*, pp. 243 - 259, Kluwer Academic press, New York 2002
- [21] F. Yao, A. Demers and S. Shankar: A Scheduling Model for Reduced CPU Energy, *IEEE Annual Foundations of Computer Science*, pp. 374 - 382, 1995