

Static and Dynamic Methods to Improve Total Reward of Tasks in Battery-Powered Devices

Chien-Chih Chu and Albert M. K. Cheng
Real-Time Systems Laboratory
Department of Computer Science
University of Houston, TX 77204, USA
(ccc,cheng)@cs.uh.edu

Abstract

This paper assumes that we know the reward value of each task in a periodic task set before we select the tasks for execution. We study three static methods to select tasks from the task sets: (1)Simplified REW-Pack [18], (2)Greedy and (3)REW-Pack [18]. We compare the reward gained by these three methods. We found that our Greedy method often yields a larger total reward value than REW-Pack's. Besides, the Greedy method is more efficient than REW-Pack.¹

1. Introduction

The objective of this paper is to describe techniques to select tasks from a task set to obtain as much overall total reward value as possible. We study three static methods to solve this problem. The three static methods are Simplified REW-Pack[18], Greedy, and REW-Pack[18]. We found that our Greedy method is more effective than Simplified REW-Pack and REW-Pack.

2. Related Work

Many papers study dynamic voltage scaling and we focus on overloaded systems. Some applications such as robot control [9,10], speech processing [4], databases [3,5] and multimedia [2,7,11] have the characteristics that they can stop the execution of an application at anytime and obtain a partial or imprecise results. Its quality of service is often a linear or concave function of the service time. Therefore, sometimes we can sacrifice quality to meet time constraints. In [1], Aydin et al. construct a framework where each real-time task comprises of a mandatory and an optional part. The mandatory part must complete before the task's deadline, while a non-

decreasing reward function is associated with the execution of the optional part, which can be interrupted at any time. They develop an optimal schedule where mandatory parts complete in a timely manner and the weighted average reward is maximized.

Besides stopping a task to meet the timing constraints and get an imprecise result, some papers assume no partial reward. In [6], Koren et al. develop an algorithm that meets most of the deadlines but allows some instances to miss their deadlines.

Most previous papers focus on two constraints: time-energy or time-reward. However, in [8], Rusu et al. study three constraints together. They develop an optimal scheme that would allow the device to run the most critical and valuable applications, without depleting the energy source while still meeting the deadline. They assume that the processor has several levels of frequencies. They attempt to activate tasks at the lowest frequency as much as possible and then lift the processor speed to obtain some time slack and add more tasks. However, their simulation only studies the situation where energy limit is very low.

This paper extends Rusu's work by using different algorithms to solve the problem. Our algorithm is more suitable when the energy limit is higher, there is a larger set of processor frequencies, or there is a larger number of tasks. Furthermore, the Greedy method is more efficient than REW-Pack.

3. System Model

We assume a frame-based task model. There are N available periodic tasks in the system, all ready at time zero. The task set is denoted by $T = \{T_1, T_2, \dots, T_N\}$. All task periods are identical and all task deadlines are equal to their periods. Each task has its own WCET and reward value v_i . The tasks are to be executed on a variable voltage processor with the ability to dynamically adjust its frequency and voltage on application requests. There are M available frequencies: $\{f_1, f_2, \dots, f_m\}$. For example, $T_i = (3, 11)$ means that the task's WCET is 3 when the processor is running at its highest speed and the reward value is 11

¹ This material is supported in part by a grant from the Institute of Space Systems Operations.

if the task completes its execution. The common deadline/period is denoted by D . Besides, each frame has a total energy limit E_{\max} . A frame consists of a subset of tasks which are selected for execution. The execution of the frame is to be repeated. It is not a requirement that all tasks must be scheduled. However, a task cannot be selected more than once during a frame.

Processor frequency, represented by f , is almost linearly related to the supply voltage:

$$f = k \times \frac{(V_{dd} - V_t)^2}{V_{dd}}$$

where k is constant, V_{dd} is the

voltage supplied, and V_t is the threshold voltage. For simplicity, we assume power consumption $P = f^\beta$ [4]. Each task consumption $e_i = f^\beta_i * c_i / f_i = f_i^{\beta-1} * c_i$, where c_i is the WCET when the task runs at the maximal speed. When decreasing processor speed, we also reduce the supply voltage.

Thus, the problem is to find the subset S , the speed f_i , and v_i in order to

$$\text{maximize } \sum_{i \in S} v_i$$

$$\text{subject } \sum_{i \in S} \frac{c_i}{f_i} \leq D$$

$$\sum_{i \in S} e_i \leq E_{\max}$$

$$S \subseteq \{1, 2, \dots, N\}$$

$$f_i \in \{f_1, f_2, \dots, f_m\}$$

We attempt to retrieve as much reward value as possible in a time frame while not violating the total time and energy constraints.

4. Algorithms

The flowchart of the Greedy method algorithm is presented in Figure 1.

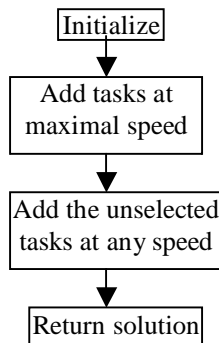


Figure 1: Flowchart of the Greedy method

The middle two components are described next in detail.

Add tasks at maximal speed: We attempt to schedule the tasks in descending order of their v/c while not violating the timing and energy constraints. We add them at the maximal frequency possible.

Add unselected tasks at any speed: We attempt to add the unselected tasks in descending order of their v/c at any speed while not violating the timing and energy constraint.

The Greedy algorithm follows.

Algorithm Greedy(a, n)

// $a[1:n]$ contains the n inputs.

```

{
(1)  solution := ∅; // Initialize the solution
(2)  selected[i] := false for i := 1 to n ;
(3)  for i := 1 to n do
(4)  {
(5)    x := Select(a);
(6)    if Feasible(solution, x) then
(7)    {
(8)      solution := Union(solution, x);
(9)      selected[i] := true;
(10)   }
(11) }
(12) for i := 1 to n do
(13) {
(14)  if (selected[i] = false)
(15)  {
(16)    if CouldAdd(a[i]) then
(17)    {
(18)      solution := Union(solution, x);
(19)      selected[i] := true;
(20)    }
(21)  }
(22) }
(23) return solution;
}
  
```

The function **Select** selects an input from $a[]$ and removes it. The selected input's value is assigned to x . **Feasible** is a Boolean-valued function that determines whether x can be included in the solution vector. The function **Union** combines x with the solution and updates the objective function. The function **Greedy** describes the essential way that a Greedy algorithm will look, once a particular problem is chosen and the function **Select**, **Feasible**, and **Union** are implemented.

In our maximal reward problem, we define these functions as follows:

Select:

We select the task with highest v_i/c_i .

Feasible:

We consider both time and energy constraints. We first add the task to run at the highest frequency possible while satisfying energy constraints and check if this task could meet the frame deadline.

Union:

If the task is feasible, we add it to our solution set.

CouldAdd:

We select the task with the highest v_i/c_i . Compute the minimal speed that could add the task while not violating the energy constraint and timing constraints. Then, we check if the task will miss its deadline. If it will not miss its deadline, then we add it to our solution.

4.1. Example

Assume that we have a task set $T = \{(3,11),(2,6),(3,10)\}$, frame deadline $D = 5$ and energy constraint $E_{max} = 5$. There are two frequencies $\{0.5,1\}$. We count value/WCET first and use VW to denote the set of value/WCET. $VW = \{11/3,3,10/3\}$. We select T_1 first due to its highest value/WCET value. We set T_1 at speed 1. Its energy cost is $1 \times 1 \times 1 \times 3 = 3$. T_3 cannot be added. We then add T_2 . The total energy cost is 5. The total reward value is 17.

5. Static Analysis

The worst-case time complexity of the REW-Pack algorithm is $O(MN \log N)$ [18]. In the Greedy method, we need to sort the tasks by v/c first, so the time complexity is $O(N \log N)$. Then, we need to calculate the corresponding speed it should use. Therefore, the worst case computation time is $O(N \log N \log M)$. The Greedy method is faster than the REW-Pack.

6. Simulation Experiments

We use simulations to evaluate our algorithms. The computation time is uniformly distributed between 1 and 100 clock cycles in the lowest frequencies. The value of each task is uniformly distributed between 1 and 100 units. We also assume in this paper that the shutdown energy cost is 0 and the switching overhead can be ignored. We repeat each experiment 100 times and count the average results of the total reward value. We compare three methods: simplified REW-Pack, REW-Pack, and Greedy. We use M to denote the number of frequencies. The total deadline of each time frame is D , which is the total time consumption of all tasks at their highest frequency. We denote the energy limit by E . The number of tasks is N .

We compared the two algorithms to a simplified version of REW-Pack mentioned in [18]. The

performance ratio shown in Figure 2 is defined as the system value returned by the algorithm (REW-Pack, Greedy) divided by the system value of the simplified REW-Pack.

$M = 5$.

$N = 25$.

We change the ratio of E/D to observe its impact on the performance ratio.

Processor Frequency = $\{0.2,0.4,0.6,0.8,1.0\}$.

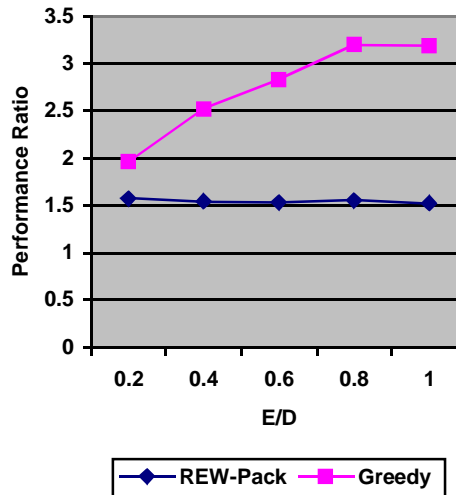


Figure 2. Energy limit and the reward value

6.1. Summary

From Figures 2, we can observe that the REW-Pack method almost always maintains its performance ratio between 1.5 and 2 regardless of the E/D value. However, the Greedy methods are always better than REW-Pack, especially when we have a larger energy limit. This is because when we have more abundant energy, we can schedule more tasks at the maximal frequency and compose a better execution task set using the Greedy method. However, REW-Pack always adds the task at its lowest frequency; in order to meet its deadline, it must drop some tasks from the selected task sets. In this situation, it cannot consider all the tasks at the same time and can only obtain a sub-optimal solution. When the E/D approaches 1, all the tasks can finish its execution at the highest frequency and do not violate the time and energy constraints. We also find that the Greedy method has a better performance when we have more tasks or more choices of processor frequencies. Due to the limited space, we don't discuss the detail here.

7. Conclusions and Future Work

In this paper, we develop a static method to schedule an overloaded, battery-powered system. We compare our methods to a previous method. We compare the performance of these four methods in many situations. Our conclusion is as follows: Greedy method often has a better performance than REW-Pack, especially when the system has more energy limit, number of processor frequencies, and number of tasks.

In future work, we plan to develop a combined method which is more suitable in more situations. We will also investigate the best scheduling choice for each system environment.

References

- [1] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez. Optimal Reward-Based Scheduling for Periodic Real-Time Tasks. Twentieth Real-Time Systems Symposium (RTSS'99).
- [2] M. Chen, G. Wei, H. Yu and X. Zhu. Scheduling Algorithm for Real-Time VBR Video Streams Using Weighted Switch Deficit Round Robin. LCN 2003: 289-290.
- [3] S. B. Davidson and A. Watters. Partial Computation in Real-Time Database Systems. IEEE 5th Workshop on Real-time Software and Operating Systems, May 1988.
- [4] W. Geng and J.W.S. Liu. An Extended Imprecise Computation Model for Time-Constrained Speech Processing and Generation. Proc. of the IEEE Workshop on Real-Time Applications, New York, NY, pp.76-80, May 1993.
- [5] J. Grass and S. Zilberstein. A Value-Driven System for Autonomous Information Gathering. Journal of Intelligent Information Systems, 14, pp. 5-27,2000.
- [6]G. Koren and D. Shasha. Skip-Over: Algorithms and complexity for Overloaded Systems that Allow Skips. Proceedings of the 16th Real-Time Systems Symposium, Pisa Italy, December 1995.
- [7]J. Nieh and M.S. Lam. SMART: A Processor Scheduler for Multimedia Applications. In Proc. of SOSP 15, December 1995.
- [8]C. Rusu, R. Melhem and D. Mosse. Maximizing the System Value while Satisfying Time and Energy Constraints. (RTSS'02) December 03 - 05, 2002 Austin, Texas.
- [9]R. Washington. On-Board Real-Time State and Fault Identification for Rovers. Proceedings of IEEE International Conference on Robotics and Automation(ICR 2000).
- [10]S. Zilberstein and S.J. Russel. Anytime Sensing, Planning and Action: A Practical Model for Robot Control. Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993.
- [11]M. Zu and A.M.K. Cheng. Real-Time Scheduling of Hierarchical Reward-Based Tasks. The 9th IEEE Real-Time and Embedded Technology and Applications Symposium May 27 - 30, 2003.