

Optimizing System Reward in Battery-Powered Spacecrafts and Rovers

Albert M. K. Cheng
Department of Computer Science

Yan Wang
Fuhua Zhou



Yan Wang and Albert M. K. Cheng

Abstract—Rechargeable batteries are used to operate many spacecrafts and autonomous rovers. Their operational periods are limited, however, by their battery supplies before the next recharge. How to use this battery-supplied energy efficiently is a critical issue. Most existing energy-conserving techniques are based on dynamic voltage scaling (DVS) and consider only timing or energy constraints. In a more realistic scenario, we should simultaneously consider three constraints: time, energy, and reward (quality-of-service). This project investigates two static methods (Greedy and Dynamic Programming) and an on-line method for selecting tasks to optimize system reward while meeting timing constraints and conserving energy. We use simulation experiments to compare the performance of these methods with existing techniques. We study three static methods to select tasks from the task sets: (1) Simplified REW-Pack, (2) Greedy, and (3) REW-Pack. We have compared the reward gained by these three methods. We found that our Greedy method often yields a larger total reward value than the gain with REW-Pack. The Greedy method, we find, is more efficient than REW-Pack.

THIS PROJECT IS TO DEVELOP AND IMPLEMENT EFFICIENT dynamic voltage scaling (DVS) algorithms in battery-operated spacecrafts and autonomous rovers^{1,2} in order to extend their battery life while satisfying timing constraints and optimizing system reward (quality-of-service or QoS). DVS technology has been incorporated by Transmeta,³ Intel,⁴ and AMD.⁵ The power consumption is defined as $P = kCV^2f$,⁶ where k is a constant, C is the total capacity of wires and transistor gates, V is the supplied power, and f is the clock frequency. Changing a processor's voltage supplied also changes its speed since $f \propto (V - V_t)^2/V$, where V_t is the threshold voltage, f is the clock frequency, and V is the voltage supplied.

Many DVS algorithms have been developed; we cite a few recent work⁶⁻⁹ here. Most papers focus on two constraints: time-energy or time-reward. Rusu et al¹⁰ is the only group who study three constraints and develop a scheme to run the most critical and valuable applications, without depleting the energy source while still meeting the deadlines. However, their approach does not perform well in overloaded systems.

Previous work on DVS has concentrated on general-purpose and soft real-time systems. However, in hard real-time systems such as spacecrafts and autonomous rovers, it is very difficult to guarantee that all tasks meet their deadlines since we often cannot predict their execution times accurately. Most

existing techniques assume that their systems can meet the demand of all tasks leading to a problem similar to the problem researchers confronted in the recent Spirit Mars Rover. In the real scenario, however, we may have to terminate some tasks if we cannot meet all their requirements.

Our objective is to develop and implement task selection techniques to obtain as much system reward (QoS) as possible. Our research points to two static and one on-line method that would reduce energy usage while meeting timing constraints and optimizing system reward. The first static method called Greedy is based on a solution to the well-known knapsack problem, while the second static method is based on dynamic programming. The on-line method is motivated by the fact that the actual workload of a set of tasks is often much less than their combined worst-case execution times (WCETs). We can therefore collect unused processor cycles and use them to execute additional tasks. This project develops these methods and studies their performance via simulation. In particular, we compare our static methods with REW-Pack,¹⁰ the only existing technique that deals with all three constraints (time, energy, and reward) but does not perform well for overloaded systems.

Methodology

We assume a frame-based task model. There are N periodic tasks: $T = \{T_1, T_2, \dots, T_N\}$. All task periods are identical and all task deadlines are equal to their periods. Each task has its own WCET and reward. 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: $\{f_1, f_2, \dots, f_m\}$. Each task can run at any of the available speed. The common deadline/period is denoted by D . 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 f is near-linearly related to the voltage: $f = k \times (V_{dd} - V_t)^2 / V_{dd}$, where k is constant, V_{dd} is the voltage supplied, and V_t is the threshold voltage. For simplicity, we assume power consumption $P = f^3$.⁸ Each task consumption $e_i = f_i^3 \cdot c_i / f_i = f_i^2 \cdot c_i$, where c_i is the WCET when the task runs at the maximal speed. When decreasing processor speed, we also reduce the supply voltage. This reduces processor power consumption cubically and task energy consumption quadratically at the expense of linearly increasing the execution time of the task.

Associated with each task T_i there is a reward v_i . The system reward (which affects the quality-of-service or QoS) is defined as the sum of the task values for all tasks that are selected for execution. The objective is to find a subset of tasks $S \subseteq \{1, 2, \dots, N\}$ that maximizes the system value $\sum_{i \in S} v_i$. For all tasks $i \in S$, the speed level $f_i \in \{f_1, f_2, \dots, f_m\}$ must also

COMPUTER RESEARCH—Yan Wang, a Ph.D. student and research assistant in the Department of Computer Science, earned her B.S. in computer science at the Peking University of Technology.

be determined. There are two major constraints on the system:

- The timing constraint imposed by the global deadline D . Each task selected for execution must finish before this deadline D .
- The energy constraint imposed by the amount of energy available in the system is denoted by E_{max} . The energy consumed by the selected tasks cannot exceed E_{max} .

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

$$\sum_{i \in S} v_i \text{ subject to } \sum_{i \in S} \frac{c_i}{f_i} \leq D, \sum_{i \in S} e_i \leq E_{max}, S \subseteq \{1, 2, \dots, N\}, \text{ and } f_i \in \{f_1, f_2, \dots, f_m\}.$$

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

Results

The maximal reward value problem is similar to the 0/1 knapsack problem. We are given n objects and a knapsack. Object i has weight w_i and the knapsack has a capacity m . If we put the object i in the knapsack, then we set $x_i = 1$ and earn a profit of p_i is earned; otherwise, we set $x_i = 0$. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Formally, the problem can be stated as:

$$\text{maximize } \sum_{1 \leq i \leq n} p_i x_i \text{ subject to } \sum w_i x_i \leq m \text{ and } x_i = 0 \text{ or } 1, 1 \leq i \leq n.$$

Next, we need to convert our problem to a knapsack problem. As defined above, a periodic task set $T = \{T_1, T_2, T_3, \dots, T_n\}$, where each task has two parameters (c, v) , where c is the WCET, and v is the reward. We explore three methods:

A. Greedy Method: We have n inputs and attempt to obtain a subset that satisfies some constraints. Any subset that satisfies these constraints is called a feasible solution. We need to find a feasible solution that maximizes a given objective function.

B. Dynamic Programming Method: This is an algorithmic design method used when the solution to a problem can be viewed as the result of a sequence of decisions. A solution to the knapsack problem can be obtained by making a sequence of decisions on the variables x_1, x_2, \dots, x_n . A decision on variable x_i involves determining which of the values 0 or 1 is to be assigned to it. Let us assume that decisions on the x_i are made in the order x_n, x_{n-1}, \dots, x_1 . Following a decision on x_n , we may be in one of the two possible states: the capacity remaining is m and no profit has accrued or the capacity remaining is $m - w_n$ and a profit of p_n has accrued. It is clear that the remaining decisions x_{n-1}, \dots, x_1 must be optimal with respect to the problem state resulting from the decision on x_n . Otherwise, x_n, \dots, x_1 will not be optimal. Hence, the principle of optimality holds.

Let $f_j(y)$ be the value of an optimal solution to the 0/1 knapsack problem. Since the principle of optimality holds, we obtain $f_n(m) = \max\{f_{n-1}(m), f_{n-1}(m - w_n) + p_n\}$. For arbitrary $f_j(y)$, $i > 0$, the previous equation generalizes to $f_j(y) = \max\{f_{j-1}(y), f_{j-1}(y - w_j) + p_j\}$. The second equation can be solved for $f_n(m)$ by beginning with the knowledge $f_0(y) = 0$

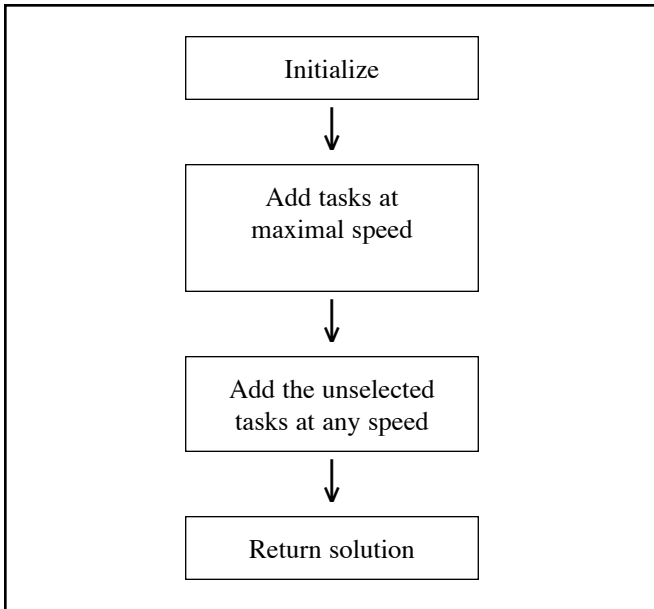


Figure 1. Flowchart of the Greedy Method

for all y and $f_i(y) = -\infty, y < 0$. Then f_1, f_2, \dots, f_n can be successively computed using this equation. Notice that $f_i(y)$ is an ascending step function; i.e., there is a finite number of y 's, $0 = y_1 < y_2 < \dots < y_k$, such that $f_i(y_1) < f_i(y_2) < \dots < f_i(y_k)$; $f_i(y) = -\infty, y < y_1$; $f_i(y) = f_i(y_1), y > y_k$; and $f_i(y) = f_i(y_j), y_j \leq y < y_{j+1}$. We need then to add the energy constraint.

C. On-Line Method: If the task execution time is in a normal distribution, there will be some time slack left due to the fact that the actual execution time of a task may be much shorter than its WCET. We can reuse this time slack to activate more tasks. There we define the best-case execution time (BCET) to be 10% to 100% of WCET. The mean of the distribution is $(\text{BCET} + \text{WCET})/2$. The standard deviation is $(\text{WCET} - \text{BCET})/6$. The sketch of the algorithm is:

1. Use the Dynamic Programming method to decide the selected task set.
2. Compute the time slack and energy slack.
 Time slack = time slack + $(\text{WCET} - \text{actual execution cycles})/f_i$.
 Energy slack = energy slack + $f_i^2 \times (\text{WCET} - \text{actual execution time})$.
3. Use the Greedy method to activate more tasks when we have enough time and energy slack.

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

We use simulations to evaluate our algorithms. 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 project 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 num-

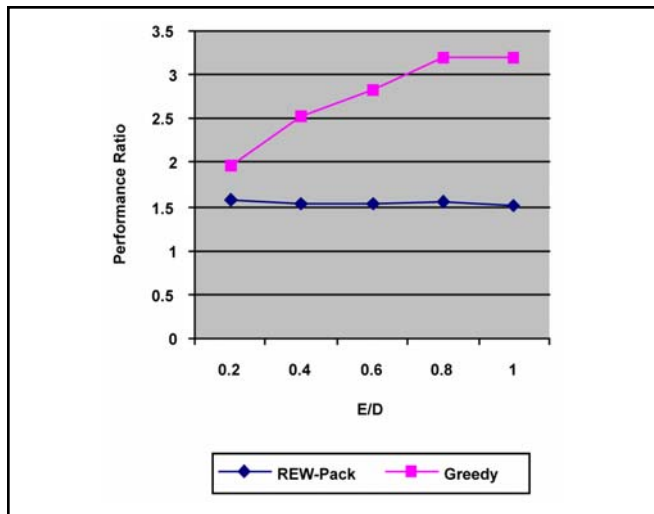


Figure 2. Energy Limit and the Reward Value

ber 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 Rusu, Melhem, and Mosse.¹⁰ The performance ratio shown in Fig. 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.

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

From Fig. 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. It is understandable that 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. Limited space does not afford discussion of the details here.

Discussion

In this project, we have developed a static method for scheduling an overloaded, battery-powered system. We compare our methods to a previous method and compare the performance of these four methods in many situations. We conclude that the Greedy method often has a better performance than REW-Pack, especially when the system has more energy limit, a number of processor frequencies, and a number of tasks. In a future project, 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.

Conclusions

This project implements power-saving methods and investigates their performance via simulation. In particular, we have compared our static methods with REW-Pack,¹⁰ the only existing technique that deals with all three constraints (time, energy, and reward) but does not perform well in overloaded systems. We believe that our algorithms are more suitable when the energy limit is higher, there is a larger set of processor frequencies, or when there are a larger number of tasks.

References

- ¹Y. Liu and A. K. Mok, "An Integrated Approach for Applying Dynamic Voltage Scaling to Hard Real-Time Systems," *Proc.*, IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 27–30, 2003. 116-23.
- ²Transmeta Efficeon. Transmeta Corporation. Feb. 22, 2005 <<http://www.transmeta.com/efficeon/architecture.html>>.
- ³R. Washington, "On-Board Real-Time State and Fault Identification for Rovers," *Proc.*, IEEE International Conference on Robotics and Automation, San Francisco, CA, April 24–28, 2000. 1175-181.
- ⁴Intel XScale Microarchitecture. Intel Corporation. Feb. 22, 2005 <<http://developer.intel.com/design/intelxscale>>.
- ⁵C. Rusu, R. Melhem, and D. Mosse. "Maximizing the System Value while Satisfying Time and Energy Constraints," *Proc.*, IEEE Real-Time Systems Symposium, Austin, TX, Dec. 3–5, 2002. 246-55.
- ⁶M. Zu and A. M. K. Cheng, "Real-Time Scheduling of Hierarchical Reward-Based Tasks," *Proc.*, 9th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 27–30, 2003. 2-9
- ⁷M. Bus, T. Givargis, and N. D. Dutt, "Exploring Efficient Operating Points for Voltage Scaled Embedded Processor Cores," *Proc.*, IEEE-CS Real-Time Systems Symposium, Cancun, Mexico, Dec. 3–5, 2003. 275.
- ⁸AMD Power Play. Advanced Micro Devices, Inc. Feb. 22, 2005 <<http://www.amd.com/us-en/Processors/ProductInformation>>.
- ⁹H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc.*, 22nd IEEE Real-Time Systems Symposium, London, UK, Dec. 2–6, 2001. 95-105.
- ¹⁰S. Zilberstein and S. J. Russel. "Anytime Sensing, Planning and Action: A Practical Model for Robot Control," *Proc.*, 13th International Joint Conference on Artificial Intel-

ligence, Chambéry, France, Aug. 28–Sept. 3, 1993. 1402-07.

Publications

- Chen, H. and A. M. K. Cheng. “Applying Ant Colony Optimization to the Partitioned Scheduling Problem for Heterogeneous Processors,” *Proc.*, IEEE Real-Time and Embedded Technology and Applications Symposium WIP Session, San Francisco, CA, March 7–10, 2005 (*to appear*).
- Cheng, A. M. K. and R. Wang. “A New Scheduling Algorithm and a Compensation Strategy for Imprecise Computation,” *Proc.*, 28th Annual International IEEE-CS International Computer Software and Application Conference (COMP-SAC), Hong Kong, Sept. 2004.
- Chokhawala, J. J. and A. M. K. Cheng. “Optimizing Power Aware Routing in Mobile Ad Hoc Networks,” *Proc.*, WIP Session, *Proc.*, IEEE-CS Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004.
- Chu, C.-C. and A. M. K. Cheng. “Static and Dynamic Methods to Improve Total Reward of Tasks in Battery-Powered Devices,” *Proc.*, WIP Session, IEEE-CS Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004.
- Wang, Y. and A. M. K. Cheng. “A Dynamic-Mode DVS Algorithm under Dynamic Workloads,” *Proc.*, IEEE Real-Time and Embedded Technology and Applications Symposium WIP Session, San Francisco, CA, March 7–10, 2005.

Presentations

- Chen, H. and A. M. K. Cheng. “Applying Ant Colony Optimization to the Partitioned Scheduling Problem for Heterogeneous Processors,” IEEE Real-Time and Embedded Technology and Applications Symposium WIP Session, San Francisco, CA, March 7–10, 2005 (*to appear*).
- Cheng, A. M. K. and R. Wang. “A New Scheduling Algorithm and a Compensation Strategy for Imprecise Computation,” *Proc.*, 28th Annual International IEEE-CS International Computer Software and Application Conference (COMP-SAC), Hong Kong, Sept. 2004.
- Chokhawala, J. J. and A. M. K. Cheng. “Optimizing Power Aware Routing in Mobile Ad Hoc Networks,” *Proc.*, WIP Session, IEEE-CS Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004.
- Chu, C.-C. and A. M. K. Cheng. “Static and Dynamic Methods to Improve Total Reward of Tasks in Battery-Powered Devices,” *Proc.*, WIP Session, IEEE-CS Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada, May 2004.
- Wang, Y. and A. M. K. Cheng. “A Dynamic-Mode DVS Algorithm under Dynamic Workloads,” IEEE Real-Time and Embedded Technology and Applications Symposium WIP Session, San Francisco, CA, March 7–10, 2005 (*to appear*).

Funding and Proposals

- Cheng, A. M. K. “Adaptive Energy-Saving Scheduling Strategies for Dynamic Real-Time Workloads,” National Science Foundation, three years, \$315,842 (*submitted*).