

# Adaptive Scheduling Server for Power-Aware Real-Time Tasks

PEDRO MEJIA-ALVAREZ

CINVESTAV-IPN, Sección de Computación

and

EUGENE LEVNER

Holon Academic Institute of Technology

and

DANIEL MOSSÉ

University of Pittsburgh

---

In this paper we propose a novel scheduling framework for a dynamic real-time environment with energy constraints. This framework dynamically adjusts the CPU voltage/frequency so that no task in the system misses its deadline and the total energy savings of the system are maximized. In this paper we consider only realistic, discrete-level speeds.

Each task in the system consumes a certain amount of energy, which depends on a speed chosen for execution. The process of selecting speeds for execution while maximizing the energy savings of the system requires the exploration of a large number of combinations, which is too time consuming to be computed on-line. Thus, we propose an integrated heuristic methodology which executes an optimization procedure in a low computation time. This scheme allows the scheduler to handle power-aware real-time tasks with low cost while maximizing the use of the available resources and without jeopardizing the temporal constraints of the system. Simulation results show that our heuristic methodology is able to generate power-aware scheduling solutions with near-optimal performance.

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms: Algorithms

Additional Key Words and Phrases: Heuristics, Real-Time Scheduling, Variable Voltage Scheduling

---

---

P. Mejia-Alvarez, CINVESTAV-IPN, Sección de Computación, AV. IPN. 2508, México. DF. pmejia@cs.cinvestav.mx,

Eugene Levner, Department of Computer Science, Holon Academic Institute of Technology, 52 Golomb ST, Holon 58102, Israel. levner@hait.ac.il,

Daniel Mossé, Computer Science Department, University of Pittsburgh, Pittsburgh, PA 15260. mosse@cs.pitt.edu. This author was supported in part by DARPA PARTS (Power-Aware Real-Time Systems) project under Contract F33615-00-C-1736 and through NSF-ITR medium grant.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 2003 ACM 0000-0000/2003/0000-0001 \$5.00

## 1. INTRODUCTION

Power management is increasingly becoming a design factor in portable and hand-held computing/communication systems. Energy minimization is critically important for devices such as laptop computers, PCS telephones, PDAs and other mobile and embedded computing systems simply because it leads to extended battery lifetime. Furthermore, the need for power-efficient designs is not solely associated with portable computing systems. Power dissipation has become a design constraint in virtually every type of computing system including desktop computers, network routers and switches, set-top entertainment systems and the most performance-hungry computer servers.

The problem of reducing and managing energy consumption was addressed in the last decade with a multi-dimensional effort by the introduction of engineering components and devices that consume less power, low power techniques involving VLSI/IC designs, algorithm and compiler transformations, and by the design of computer architectures and software with power as their primary criterion for performance. Recently, hardware and software manufacturers have introduced standards such as the ACPI (Advanced Configuration and Power Interface) [Intel-ACPI 2003] for energy management of laptops, desktops and servers that allow several modes of operation, several sleep levels, and the ability to turn off some parts of the computer (e.g., the disk) after a preset period of inactivity. More recently, *variable voltage scheduling* (VVS) has been proposed as an alternative to energy management. In VVS the voltage and frequency (hence, the CPU speed) is adjusted dynamically. Because the power consumption is linearly dependent on the time and quadratically dependent on voltage, reducing the frequency and voltage of operations (we call this *reducing the speed*) will cause them to consume less energy, but take longer to complete.

In this paper, the VVS problem in real-time systems is to assign appropriate speeds (from a set of discrete speeds) to a set of dynamically arriving and departing periodic tasks, such that no task misses its predefined deadline while the total energy savings in the system is maximized. The identification of feasible options that maximize our optimality criteria (expressed as the total energy savings of the system) requires the exploration of a large combinatorial space of solutions. This optimization problem is formulated as a multiple-choice knapsack problem (MCKP) with binary variables [S. Martello 1990].

In order to cope with the high computation costs of the dynamic real-time environment, we have developed a low-cost power-aware scheduling scheme. Our Power-Optimized Real-Time Scheduling (PORTS) Server consists of four stages: (a) an *acceptance test* for deciding if and when dynamically arriving tasks can be accepted in the system, (b) a *reduction procedure* which transforms the original multiple-choice knapsack optimization problem into a standard knapsack problem, (c) a *greedy heuristic algorithm* used to solve the transformed optimization problem, and (d) a *restoration algorithm* which restores the solution of the original problem from the transformed problem. The PORTS methodology provides a novel approach to real-time scheduling which yields a near-optimal solution for the problem of selecting speeds of execution of all tasks in the system. The solution developed satisfies the condition of maximizing the energy savings of the system while guar-

anteeing the deadlines of all tasks in the system. The performance of the PORTS Server and its heuristic algorithms will be compared with the performance of known algorithms.

The rest of this paper is organized as follows. In Section 2 related models and previous work are reviewed. In Section 3, the system and energy models used are defined. In Section 4, the power-optimized scheduling is formulated as an optimization problem. In Section 5, the Power-Optimized Real-Time Scheduling (PORTS) Server is described and in Section 6, we use an example to simulate the execution of our algorithms, and compare its performance against known algorithms. In Section 7 we provide a theoretical justification of our method. In this section we prove that our method produces a 2-approximate solution to our optimization problem. In Section 8, simulation results are presented to show the performance of the PORTS Server. Finally, Section 9 presents concluding remarks.

## 2. RELATED WORK

In this paper we address the issue of reducing power consumption of processors through dynamically changing the speed (i.e., voltage and frequency) of a processor. This technique can be classified as *static* and *dynamic*. Static techniques, such as static scheduling, compilation for low power [D. Mossé 2000] and synthesis of systems-on-a-chip [I. Hong 1998c], are applied at design time. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving dynamically arriving real-time tasks or variable workloads.

Static (or off-line) scheduling methods to reduce power consumption in real-time systems were proposed in [F. Yao 1995; T. Ishihara 1998; I. Hong 1998a]. These approaches addressed aperiodic tasks. Heuristics for on-line scheduling of aperiodic tasks while not hurting the feasibility of off-line periodic requests were proposed in [I. Hong 1998b]. Non-preemptive power-aware scheduling is investigated in [I. Hong 1998a]. Recent work on VVS includes the exploitation of idle intervals in the context of the Rate Monotonic and Earliest Deadline First (EDF) scheduling frameworks [Y. Shin ; C. M. Krishna 2000; H. Aydin 2001b; J.R. Lorch 2001]. Following the same VVS technique, other work [A. Dudani 2002; P. Pillai 2001] considers the fact that some real-time tasks do not always consume their worst-case execution times, and have the ability to dynamically reclaim unused computation time to obtain additional energy savings.

Most of the above research work on VVS assumes that all tasks have identical power functions. Using an alternate assumption, efficient power-aware scheduling solutions are provided where each real-time task may have different power consumption characteristics [H. Aydin 2001a; F. Gruian 2001]. A recent survey of such methods, considering continuous voltage has shown the behavior of many different VVS algorithms [D. Shin 2002].

Although systems which are able to operate on an almost *continuous* voltage spectrum are rapidly becoming a reality thanks to advances in power-supply electronics [T.D. Burd 2000], it is a fact nowadays that most of the microprocessors that support dynamic voltage scaling use a few *discrete* voltage levels. Some examples of processors that support discrete voltage scaling are: (a) the Crusoe processor [transmeta 2003] (200–700 MHz in 33 MHz steps, 1.1–1.6V); (b) the ARM7D pro-

cessor [Intel ] (20MHz or 33MHz, 5V or 3.3V); (c) the Intel StrongARM SA1100 processor [Intel ] (59–221 MHz in 14.7 MHz Steps) [arm 2003]; and (d) the Intel XScale (150MHz–1GHz) [Intel-Xscale 2003]. Contrary to the *continuous voltage* assumption of most of the recent research, discussed above, in this paper we consider only realistic, discrete-level speeds.

### 3. SYSTEM AND ENERGY MODELS

We consider a set  $\mathbf{T} = \{T_1, \dots, T_n\}$  of  $n$  periodic preemptive real-time tasks running on one processor. Tasks are independent (i.e., they share no other resources besides the CPU) and have no precedence constraints. Each task  $T_i$  arrives in the system at time  $a_i$ . We consider dynamic systems in which an admission control procedure is executed to admit or reject tasks that arrive in the system. We assume that there is a contingency plan to take care of rejected tasks.

The Earliest Deadline First (EDF) [C.L. Liu 1973] scheduling policy will be considered. The *life-time* of each task  $T_i$  consists of a fixed number of instances  $r_i$ , that is, after the execution of  $r_i$  instances, the task leaves the system. Each task  $T_i$  has associated a Period  $P_i$ , which represents the minimum inter-arrival time of consecutive *instances* of the task. In our model, we assume that  $P_i$  is equal to the relative deadline of the task. Further, once a task passes the admission control, it is guaranteed to meet the deadlines of its  $r_i$  instances.

Examples of event-driven real-time systems exhibiting this behavior include: (1) video-on-demand systems, where media streams are generated aperiodically; each stream contains a fixed number of periodic instances which are transmitted over the network, and (2) digital signal processing, where each task processes source data that often arrives in a *bursty* fashion. These types of systems can be found in communication satellites, which have an extreme need for extending power management (for instance, a 10% increase in battery life for a satellite with a 5-year estimated lifetime would make the satellite functional –and profitable– for an extra 1/2 year).

We denote by  $C_i$  the number of processor cycles required by  $T_i$  in the worst-case. We assume that  $C_i$  processor cycles<sup>1</sup> are executed on each instance of  $T_i$ . Under a constant speed  $f_i$  (given in cycles per second), the execution time of the task is  $t_i = \frac{C_i}{f_i}$ . A schedule of periodic tasks is *feasible* if each task  $T_i$  is assigned at least  $C_i$  CPU cycles before its deadline at every instance. The *utilization* of a task is the processor load that it demands for execution:  $U_i = \frac{t_i}{P_i}$  (or  $\frac{C_i}{f_i \cdot P_i}$ ). According to EDF, a set of tasks is feasible (no tasks misses its deadline) if  $\sum U_i \leq 100\%$ . Although we assume EDF in this paper, the scheme can be easily extended to fixed-priority schedulers [C.L. Liu 1973; M. Joseph 1986].

We assume that the CPU speed can be changed at *discrete* levels between a minimum speed  $f_{min}$  (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed  $f_{max}$ . In our formulation,  $f_{ij}$  denotes the speed of execution of an instance of task  $T_i$  when it executes at speed  $j$ , and  $U_{ij}$  denotes the utilization of task  $T_i$  executing at speed  $j$ . We assume that the speed remains the same during the execution of a single instance.

<sup>1</sup>Contrary to our assumptions, the work in [H. Aydin 2001b; A. Dudani 2002; P. Pillai 2001] consider that some real-time tasks not always execute their worst-case processor cycles.

The energy  $E$ , measured in Joules (J) consumed by a computer over  $t$  seconds is equal to the integral of the power consumption, measured in Watts (W). The dominant source of power consumption in digital CMOS circuits is the dynamic power dissipation ( $P_d$ ), characterized by [T.D. Burd 2000; A.P. Chandrakasan 1992],

$$P_d = C_a N_{sw} \cdot V_{dd}^2 \cdot f \quad (1)$$

where  $C_a$  is the *output capacitance*,  $N_{sw}$  is the number of switches per clock, and  $f$  is the processor clock frequency, that is the processor speed.  $P_d$  is a strictly increasing *convex* function of the supply voltage  $V_{dd}$ , but its exact form depends on the technology [I. Hong 1998c].

Processor speed is almost linearly related to the supply voltage [T.D. Burd 2000; A.P. Chandrakasan 1992]:

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

where  $k$  is a constant, and  $V_t$  is the threshold voltage (i.e., the minimum voltage that can be supplied to the processor allowing full and correct functionality). Since power varies linearly with the clock speed and the square of the voltage, adjusting both can produce cubic power reductions, at least in theory. Since the time needed to execute task  $T_i$  is  $t_i = \frac{C_i}{f}$ , the energy consumption of the task executing for an interval of time  $I$ , is  $E = P_d \cdot I$ .

While applying voltage-clock scaling under EDF scheduling, we make the following two additional assumptions. First, the time overhead associated with voltage switching is negligible, that is, the voltage change overhead can be incorporated in the workload of each task. According to [transmeta 2003] the time overhead associated with voltage switching in the Transmeta Crusoe microprocessor is less than 20 microseconds per step. The worst-case scenario of a full swing from 1.1 V to 1.6 V takes 280 microseconds. Second, different tasks have different power consumptions, that is, the power dissipation is dependent on the nature of the running software of each task in the system. For example, a task may use more memory or the floating point unit than other tasks, or it may be executed on specialized processors (e.g., DSPs, micro-controllers, or FPGAs).

#### 4. FORMULATION OF THE PROBLEM

In a real-time system with energy constraints, the scheduler should be able to guarantee the timing constraints of all tasks in the system and to select the speed of execution of each task such that the energy consumption of the system is minimized, or equivalently, that the energy savings of the system is maximized. Obviously, energy savings are minimum when all tasks execute at their maximal speeds, therefore, in order to maximize energy savings, the minimum possible speed for each task has to be determined such that no task misses its deadline.

The problem we address can be formulated as follows. Each time a new task  $T_i$  arrives at or leaves the system, determine the speed of execution for each task in the system such that no task misses its deadline and the energy savings of the system is maximized. Note that a solution to this problem must be computed each

time a new task arrives at or leaves the system; thus, the solution must have low computational complexity.

#### 4.1 The Optimization Problem

We define  $N_i$ , as the set of speeds<sup>2</sup> for each task  $T_i$ . Each level of speed  $j \in N_i$  has an Energy Saving computed by

$$S_{ij} = (E_{i1} - E_{ij}) \quad (3)$$

where  $E_{ij}$  is the energy consumed by task  $T_i$  executing at speed  $j$  ( $j = 1$  means maximum speed). Note that,  $S_{i1} = E_{i1} - E_{i1} = 0$ ; that is executing at maximum speed yields no energy savings. It is assumed that the items (speeds)  $j \in N_i$  are arranged in non-decreasing order, so that  $S_{i1}$  and  $U_{i1}$  are the items with the smallest values.

Each task  $T_i$  in the system accrues an accumulated energy savings  $S_i^k$  upon executing a number of instances during the interval of time between different tasks arrivals  $a_k$  and  $a_{k+1}$ .  $S^k$  denotes the amount of energy savings accrued by all the tasks in the system during  $a_{k+1} - a_k$ , that is,  $S^k = \sum_{i=1}^n S_i^k$ .

The aim of this *optimization problem* is to find a speed level  $j \in N_i$  for each task  $T_i$ , such that the sum of energy savings for all tasks is maximized without having the utilization sum to exceed the capacity of the system<sup>3</sup> (i.e., 100%).

That is, Problem  $P_0$  is defined as follows.

$$\begin{aligned} \text{maximize} \quad & Z_0 = \sum_{i=1}^n \sum_{j \in N_i} S_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^n \sum_{j \in N_i} U_{ij} x_{ij} \leq 100\% \\ & \sum_{j \in N_i} x_{ij} = 1, \quad i = 1, \dots, n \\ & x_{ij} = \begin{cases} 1 & \text{if speed } j \in N_i \text{ for task } T_i \text{ is chosen} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

By achieving the optimality criteria, whenever a new task arrives at or leaves the system, we intend to maximize the accumulated energy savings  $S^k$  obtained after scheduling the entire set of tasks for the complete duration of the schedule.

We have formulated the power saving problem as a Multiple-Choice Knapsack Problem (MCKP) with 0-1 variables [S. Martello 1990]. However, the MCKP is known to be NP-hard [S. Martello 1990] which implies that there is no general fast (polynomial-time) exact method for its solution. From a practical point of view this means that exact methods, such as dynamic programming [S. Martello 1990], Lagrange multipliers [H. Aydin 2001a], mixed-integer linear programming [V. Swaminathan 2001] and enumeration schemes [I. Hong 1998b], do not satisfy realistic temporal requirements for solving any reasonable size MCKP.

<sup>2</sup>In  $N_i$  there may (or may not) be a different set (or different number) of speed levels than in  $N_k$  ( $i \neq k$ ).

<sup>3</sup>This bound is given by the EDF Scheduling Policy.

## 5. PORTS: POWER-OPTIMIZED REAL-TIME SCHEDULING SERVER

The Power-Optimized Real-time Scheduling (PORTS), is an extension of the Earliest Deadline First scheduling policy (EDF [C.L. Liu 1973]). The PORTS Server is capable of handling dynamic real-time tasks with power constraints, such that the energy savings of the system is maximized and the deadlines of the tasks are always guaranteed. In order to meet our optimality criteria, when new tasks arrive in the system, the PORTS Server adjusts the load of the system by controlling the speed of execution of the tasks.

The PORTS Server is activated whenever a task arrives at or leaves the system. The proposed method consists of five basic parts, or stages, as illustrated in Figure 1, and described in detail in the following subsections.

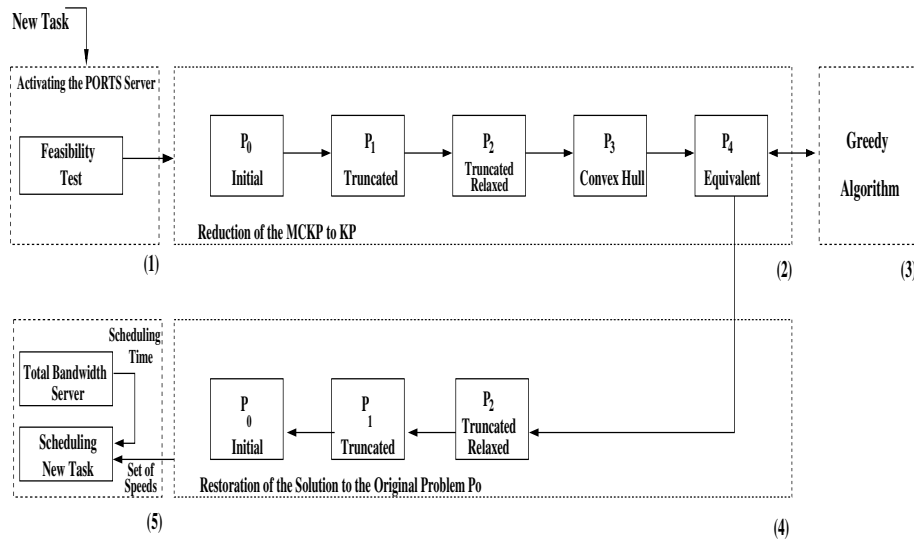


Fig. 1. Methodology for Handling Power-Aware Real-Time Tasks

The PORTS Server first executes a Feasibility Test (FT) to decide whether or not the new task can be accepted for execution in the system. If the new task is accepted, an *optimization procedure* is executed to calculate the speeds of execution of all tasks in the system.

This optimization procedure consists of three parts:

- (1) A *reduction algorithm*, which converts the original MCKP to a standard KP.
- (2) An *approximation algorithm* (e.g. Enhanced Greedy Algorithm) capable of finding an approximate solution to the reduced KP, and
- (3) A *restoration algorithm*, which re-constructs the solution of the MCKP from the solution of the standard KP.

The solution provided by the *optimization procedure* (speeds of execution of all tasks) is such that no task in the system misses its deadline and the energy savings of the system is maximized. After the *optimization procedure* is executed, the Total Bandwidth Server [G. Lipari 2000] is used to compute the start time of the new task.

Finally, with the start time of the new task computed and the solution provided by the optimization procedure (the set of speeds for execution), the PORTS Server will schedule the new task in the system.

### 5.1 Activating the PORTS Server and Feasibility Test

The two conditions for activating the PORTS Server and their procedures are:

**1. Task Arrival.** When a new task  $T_j$  arrives in the system, the feasibility test is executed.

The new task is rejected when, running all tasks (including  $T_j$ ) at their maximum speeds (minimum utilization), makes the system infeasible (some tasks miss their deadlines). Otherwise, the new task is accepted:

**Feasibility Test (FT):**

$$FT = \begin{cases} T_j \text{ is accepted} & \text{if } U_{min} = \sum_{i=1}^n U_{i1} \leq 100 \% \\ T_j \text{ is rejected} & \text{otherwise} \end{cases}$$

After a new task has been accepted in the system, the next problem is to choose the speed of execution of each task in the system. This problem is related to our optimization problem because by choosing a speed for the execution of task  $T_i$  we will obtain its corresponding energy savings achieved.

**2. Task Departure.** The PORTS Server is also activated when a task leaves the system. In this case, the optimization procedure is executed to satisfy the optimality criteria for the new set of tasks in the system. At task departure, the Feasibility Test does not need to be executed since the system utilization is decreased when a task terminates.

### 5.2 Reduction Scheme from MCKP to the Standard KP

Our approximation algorithm is based on the reduction of the MCKP to the equivalent KP using the *convex hull* concept [S. Martello 1990]. In order to reduce the MCKP, denoted by  $P_0$ , the following auxiliary problems will be used:

#### $P_1$ : The Truncated MCK Problem

Problem  $P_1$  is constructed from  $P_0$ , by extracting the lightest item (i.e., the item with the minimum  $U_{ij}$  value) from each set  $N_i$  and assuming that all these items are inserted into the knapsack. The sum of the lightest items from each set  $N_i$  is denoted by  $S_{min} = \sum_{i=1}^n S_{i1}$  and  $U_{min} = \sum_{i=1}^n U_{i1}$ . When formulating  $P_1$ , we have to consider that the new capacity of the system is now equal to  $(c - U_{min})$ , so we have to write  $\sum_{j \in N_i} x_{ij} \leq 1$  (instead of  $\sum_{j \in N_i} x_{ij} = 1$ ) because the lightest items are assumed to be already inserted into the knapsack. Therefore, some or even all sets  $N_i$  in Problem  $P_1$  may contain no items, that is, it may happen that  $\sum_{j \in N_i} x_{ij} = 0$  (no speed for task  $T_i$  is chosen) for the optimal solution of Problem  $P_1$ . Notice that  $c = 1.0$  (or  $c = 100\%$ ) denotes the maximum capacity of the system.

**Problem  $P_1$ :**

$$\begin{aligned} &\text{Maximize } Z_1 = \sum_{i=1}^n \sum_{j \in N_i} (S_{ij} - S_{i1}) x_{ij} \\ &\text{subject to } \sum_{i=1}^n \sum_{j \in N_i} (U_{ij} - U_{i1}) x_{ij} \leq (c - U_{min}), \\ &\sum_{j \in N_i} x_{ij} \leq 1, i = 1, \dots, n, \\ &x_{ij} = 0 \text{ or } 1, \text{ for } j \in N_i, i = 1, \dots, n. \end{aligned}$$

 **$P_2$ : The Truncated Relaxed MCK Problem**

Problem  $P_2$  is constructed from Problem  $P_1$  by allowing a relaxation on the variable integrality condition:  $0 \leq x_{ij} \leq 1$ , in other words,  $x_{ij}$  values are not necessarily 0 or 1. Let  $Z_2$  be the objective function of Problem  $P_2$ . The reason for introducing this problem is that its exact solution can be found in low computation time, which in turn, provides a good approximate solution to Problem  $P_1$  and hence a good approximate solution to  $P_0$  [P. Sinha 1978]. Problem  $P_2$  can be solved by the following  $P_3$  and  $P_4$  problems [Lawler 1979; P. Sinha 1978; S. Martello 1990; Pisinger 1995].

 **$P_3$ : The Relaxed MCK Problem on the Convex Hull**

Given  $P_2$ , a convex hull of items in each set  $N_i$  can be found [S. Martello 1990]. The elements constituting the convex hull will be called *P-undominated* and denoted by  $(R_{ij}, H_{ij})$  (this notion will be explained below in more detail).

Let us start by denoting the savings without the lightest item in Problem  $P_2$  by  $s_{ij} = S_{ij} - S_{i1}$  and analogously denote the utilization by  $u_{ij} = U_{ij} - U_{i1}$ .

The following Dominance Criterion (Sinha and Zoltners [P. Sinha 1978]) is applied to reduce the set of items in the convex hull.

**Dominance Criterion 1.** *If two items  $r$  and  $q$  in the same set  $N_i$  in Problem  $P_2$  satisfy  $s_{ir} \leq s_{iq}$  and  $u_{ir} \geq u_{iq}$  then item  $r$  is said to be **dominated** by item  $q$ . In every optimal solution of Problem  $P_3$ ,  $x_{ir} = 0$ , that is, the **dominated** items do not enter into the optimal solution.*

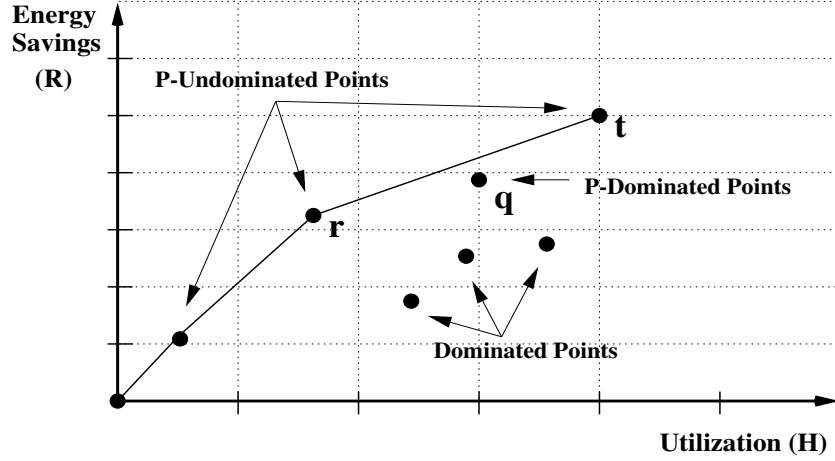
**Dominance Criterion 2.** *If some items  $r, q, t$  from the same set  $N_i$  are such that  $s_{ir} \leq s_{iq} \leq s_{it}, u_{ir} \leq u_{iq} \leq u_{it}$ , and*

$$\frac{(s_{iq} - s_{ir})}{(u_{iq} - u_{ir})} \leq \frac{(s_{it} - s_{iq})}{(u_{it} - u_{iq})} \quad (4)$$

*then  $x_{iq} = 0$  in every optimal solution of Problem  $P_2$ .*

The item  $q \in N_i$ , depicted in Figure 2, is called *P-dominated* [P. Sinha 1978]. In what follows, we exclude *P-dominated* items from each set  $N_i$  when solving the Relaxed MCK Problem  $P_3$  to optimality.

The items remaining after we excluded all the *P-dominated* items are called *P-undominated*. All the *P-undominated* items belonging to the same set  $N_i$ , if depicted as points in the two dimensional space, form the convex hull of the set  $N_i$  [S. Martello 1990], and denote the new set of P-undominated items  $\{(R_{ij}, H_{ij})\}$ , as illustrated in Figure 2. Note that R denotes energy savings and H denotes utilization.

Fig. 2. Problem  $P_3$ : Convex Hull.

The set of all  $P$ -undominated items may be found by examining all the items in each set  $N_i$  in an increasing order and according to Equation (4). Because of the ordering of the items, the upper convex hull can be found in  $O(m \log m)$  time [P. Sinha 1978], where  $m$  denotes the total number of items in the system,  $m = \sum_{i=1}^n |N_i|$ . The obtained Multiple-Knapsack Problem on the Upper Convex Hull is denoted as Problem  $P_3$ .

**Problem  $P_3$ :**

$$\begin{aligned} &\text{Maximize } Z_3 = \sum_{i=1}^n \sum_{j \in N_i} R_{ij} y_{ij} \\ &\text{subject to } \sum_{i=1}^n \sum_{j \in N_i} H_{ij} y_{ij} \leq (c - U_{min}), \\ &\quad \sum_{j \in N_i} y_{ij} \leq 1, i = 1, \dots, n, \\ &\quad 0 \leq y_{ij} \leq 1, \text{ for } j \in N_i, i = 1, \dots, n. \end{aligned}$$

As described in [P. Sinha 1978], some items belonging to the set  $N_i$  (i.e.,  $y_{ij} = 1$ ) can be included into the solution entirely; they are called *integer variables*. On the other hand, some items may exceed the constraint:  $\sum_{i=1}^n \sum_{j \in N_i} (H_{ij} y_{ij}) \leq (c - U_{min})$  and only part of it could be included into the solution. These items are called *fractional variables*.

**$P_4$ : The Equivalent Knapsack Problem (EKP)**

The Equivalent Knapsack Problem  $P_4$  is constructed from  $P_3$ . In each set  $N_i$ , *slices* or *increments* are defined as follows:

$$P_{ij} = (R_{ij} - R_{i,j-1}); \quad i = 1, \dots, n; j = 2, \dots, CH_i \quad (5)$$

$$W_{ij} = (H_{ij} - H_{i,j-1}); \quad i = 1, \dots, n; j = 2, \dots, CH_i \quad (6)$$

where  $CH_i$  is the number of the  $P$ -undominated items in the convex hull of set  $N_i$ .

When solving the (continuous) Problem  $P_3$ , we may now discard<sup>4</sup> the condition  $\sum_{j \in N_i} y_{ij} \leq 1, i = 1, \dots, n$ , and solve the problem of selecting *slices* in each set  $N_i$ .

**Problem  $P_4$  :**

$$\begin{aligned} &\text{Maximize } Z_4 = \sum_{i=1}^n \sum_{j \in N_i} P_{ij} z_{ij} \\ &\text{subject to } \sum_{i=1}^n \sum_{j \in N_i} (W_{ij} z_{ij}) \leq (c - U_{min}), \\ &0 \leq z_{ij} \leq 1, \text{ for } j \in N_i, i = 1, \dots, n. \end{aligned}$$

From the analysis of Problem  $P_4$  [P. Sinha 1978; Lawler 1979] it follows that, in all integer sets  $N_i$ : if some variable is equal 1 (e.g., the variable is chosen) then all preceding variables are also 1; if some variable is equal zero (e.g., the variable is not chosen) then all subsequent variables are also zeros. From this fact the following important properties of Problem  $P_4$  follow.

**Property 1:** *The combination of all items (slices) from set  $N_i$  ( $i = 1, \dots, n$ ) yielding an optimal solution to Problem  $P_4$ , corresponds to a specific item to be chosen from set  $N_i$  in Problem  $P_3$ ; namely, if  $k$  denotes this specific item, then  $\sum_{\{j \in N_i\}} (P_{ij} x_{ij}) = R_{ik}$  and  $\sum_{\{j \in N_i\}} (W_{ij} x_{ij}) = H_{ik}$ . In each set  $N_i$  all the slices are numbered in the decreasing order of their ratios,  $\frac{P_{ij}}{W_{ij}}$ .*

**Property 2:** *There should not be a gap in a set of slices corresponding to a solution in any set  $N_i$ .*

To exemplify Property 1, assume that  $P_{1,1} = 1642$ ,  $P_{1,2} = 2764$ , and  $P_{1,3} = 2074$  are the slices from set  $N_1$ . This optimal solution of Problem  $P_4$  correspond to  $R_{1,3} = 6480$  from Problem  $P_3$ . To exemplify Property 2, let us consider the set  $N_j$  containing the *slices*  $r, q$  and  $t$ . According to Property 2, the following solutions are *valid*:  $\{\}, \{r\}, \{r, q\}$  and  $\{r, q, t\}$ , while  $\{q\}, \{t\}, \{r, t\}$  and  $\{q, t\}$  are *invalid*. Note that,  $\{r, t\}$  is invalid because *slice*  $q$  is not included, causing a *gap* in the solution.

### 5.3 Enhanced Greedy Algorithm

In order to solve the Equivalent Knapsack Problem  $P_4$ , we may collect all *slices* from all sets  $N_i$  (following a decreasing order of their ratios,  $\frac{P_{ij}}{W_{ij}}$ ) as candidates for including them into a single set:  $PW$ . With all *slices* in the single set  $PW$ , now the problem becomes the standard knapsack problem.

The main idea of the Standard Greedy Algorithm (SGA) for solving the standard knapsack is to insert the *slices*,  $\{p_i, w_i\}$  (obtained from the single set  $PW$ ) inside the available capacity of the knapsack ( $c - U_{min}$ ) in order of decreasing ratio  $\frac{p_i}{w_i}$ , until the knapsack capacity is completely full, or until no more *slices* can be included. If the knapsack is filled exactly to its full capacity ( $c - U_{min}$ ) in the mentioned order, then this is the optimal solution. While inserting *slices* into the knapsack, one of them may not fit into the available capacity of the knapsack. This *slice* is called the *break-slice* [S. Martello 1990], and its corresponding set is called the *break-set*.

Contrary to the solution proposed by Pisinger [Pisinger 1995], our method does not consider *fractional items* to be part of the solution. Therefore, we will discard

<sup>4</sup>This condition is discarded because after Problem  $P_4$  we will include all items from all tasks into a single set in the Enhanced Greedy Algorithm discussed in the next Section.

the *break-slices*, and consequently (following Property 2) all remaining *slices* from the same *break-set*.

To the greedy scheme of [Pisinger 1995] we add the following two rules.

- Rule 1.** *When computing the solution of  $P_4$  take into account  $Z'_4 = \{(p_{max}), \hat{Z}_4\}$ , where  $p_{max} = \max\{p_i\}$  is the maximal energy saving item in the Truncated MCK Problem  $P_2$  and  $\hat{Z}_4 = p_1 + p_2 + \dots + p_{k-1}$  is the solution obtained by the Standard Greedy Algorithm (SGA).*
- Rule 2.** *After finding the break-slice, the remaining empty space is filled in by slices from the non-break sets in decreasing order of the ratios  $\frac{p_i}{w_i}$ .*

The SGA algorithm is executed until the first *break-slice* is found. The Enhanced Greedy Algorithm (EGA) algorithm, in contrast to SGA, does not stop when the first break-slice is found; it is executed for all remaining *slices* in the single set  $PW$ . According to Rule 2, *break-slices* are not considered to be part of the solution in the EGA algorithm. The SGA and EGA algorithms are illustrated in Figure 3.

---

```

1: Enhanced Greedy Algorithm: (EGA Algorithm)
2: input: a set of slices  $p_j$  and  $w_j$  from  $P_4$  ordered by the ratio  $\frac{p_i}{w_i}$ 
3:  $\hat{c} = (c - U_{min})$ ,  $\hat{n}$ : (number of items on  $P_4$ )
4: output:  $x_i$ ,  $(p^*, w^*)$ : (solution set);
5: begin
6:    $p^* = 0$ ;  $w^* = 0$ ;
7:   for  $j = 1$  to  $\hat{n}$  do
8:     if  $w_j > \hat{c}$  then
9:        $x_j = 0$ ; break-slice =  $j$ ;
10:      remove the remaining slices from the break set (Property 2)
11:      exit; (condition for SGA algorithm)
12:     else
13:        $x_j = 1$ ;  $\hat{c} = \hat{c} - w_j$ ;
14:        $p^* = p^* + p_j$ ;  $w^* = w^* + w_j$ 
15:   end;

```

---

Fig. 3. Greedy Algorithms: SGA, EGA

#### 5.4 Restoring the Solution from the EKP to the MCKP

An approximate solution to Problem  $P_4$  is obtained as follows:

- SGA Algorithm:  $Z'_4 = \max\{p_{max}, (p_1 + p_2 + \dots + p_{k-1})\}$
- EGA Algorithm:  $Z''_4 = \max\{p_{max}, (p_1 + p_2 + \dots + p_{k-1} + \alpha)\}$

The term  $\alpha$  is a possible increment caused by using Rule 2, that is, the profits of additional items from non-break sets.

The approximate solution to the Problem  $P_0$  is defined as  $Z_4 + S_{min}$ . Recall that  $S_{min} = \sum_{i=1}^n S_{i1}$ , are the elements truncated in Problem  $P_1$ . From the definition of the *slice* (described in Equations 5 and 6) and Property 1, it follows that if several *slices* (for example  $s, r$  and  $t$ , in that order) belonging to the same set  $N_j$  are chosen to be part of the solution of the greedy algorithm, then the item corresponding to

the *slice*  $t$  is considered to be part of the solution of Problem  $P_0$ . On the other hand, if no *slice* is chosen from set  $N_j$  to be part of the solution, then the truncated item considered in Problem  $P_1$  ( $S_{j1}$  and  $U_{j1}$ ) is chosen to be part of the solution. The above criteria allow us to construct the corresponding items (speeds), from each set  $N_j$  of Problem  $P_4$  that are part of the solution of Problem  $P_0$ .

The Algorithm that describes the Optimization Procedure is illustrated in Figure 4. The solution from Problems  $P_1$ ,  $P_2$  and  $P_4$  can be obtained in  $O(m)$  time (lines 5 to 18 from Figure 4), while the EGA Algorithm obtains solutions in  $O(m \log m)$  time (lines 19 to 22 from Figure 4). The Reduction Algorithm (Problems  $P_1$ ,  $P_3$  and  $P_4$ ) is executed in lines 7 to 17. The Approximate Algorithm is executed in lines 19 and 20, and the Restoration Algorithm is executed in line 21.

It will be shown in Section 7 that the complete procedure provides a *2 approximate solution* to  $P_0$  in the worst case.

---

```

1: Optimization Procedure:
2: input: set of  $S_{ij}$  and  $U_{ij}$            (energy savings and utilization from problem  $P_0$ )
3:       $c$ :                               (size of the knapsack)
4: output:  $x_{ij}$ ,  $(p^*, u^*)$  :         (solution: vector, energy savings and utilization)
5: begin
6:      $S_{min} = \sum_{i=1}^n S_{i1}$ ;           (Savings and Utilization at maximum speed)
7:      $U_{min} = \sum_{i=1}^n U_{i1}$ ;
8:     for  $j = 1$  to  $n$  do
9:       begin
10:       $P_1$  :    $s_{ij} = S_{ij} - S_{i1}$ ;  $u_{ij} = U_{ij} - U_{i1}$ ;  $\hat{c} = (c - U_{min})$ ;
11:       $P_3$  :   for (all items in  $N_i$ )           (items  $r$ ,  $q$  and  $t$  belong to  $N_i$ )
12:              if ( $s_{ir} \leq s_{iq}$  and  $u_{ir} \geq u_{iq}$ ) then
13:                  remove item ' $r$ ' from set  $N_i$ ;           (Dominance Criterion 1)
14:                  if  $\left( \frac{s_{iq} - s_{ir}}{u_{iq} - u_{ir}} \leq \frac{s_{it} - s_{iq}}{u_{it} - u_{iq}} \right)$  then
15:                      remove item ' $q$ ' from set  $N_i$ ;           (Dominance Criterion 2)
16:                  for (all  $P$ -undominated items in  $N_i$ ) do
17:                       $R_{ij} = s_{ij}$ ;  $H_{ij} = u_{ij}$ ;           (Convex Hull with  $P$ -undominated items)
18:       $P_4$  :    $P_{ij} = (R_{ij} - R_{i,j-1})$ ;  $W_{ij} = (H_{ij} - H_{i,j-1})$ ;  $j = 2, \dots, CH_i$            (slices)
19:       end
20:       Insert all  $(P, W)$  items in order of  $\frac{P}{W}$  into array  $PW$ 
21:       Execute the Greedy Algorithm with  $PW$  as input.           (output:  $x_{ij}, p^*, w^*$ )
22:        $P = S_{min} + p^*$ ;  $W = U_{min} + w^*$ ;           (energy savings and utilization solution)
23:     end

```

---

Fig. 4. Optimization Procedure

## 5.5 Scheduling the New Task

After the optimization procedure is executed, the Total Bandwidth Server (TBS) [G. Lipari 2000] will calculate the start time of the new task. It is well known that TBS Server provides low response times for handling aperiodic tasks. As described in [G. Buttazzo 1999], the TBS Server assigns a deadline  $d_{TBS} = \max(t_a, d_{a-1}) + \frac{C_a}{U_s}$  to new aperiodic requests  $\tau_a$  that arrive at time  $t = t_a$ . In our framework,  $C_a$  denotes the computation time of the new task and  $U_s$  denotes the server utilization

factor.  $U_s$  is computed in our framework as  $U_s = 1 - \sum_i U_i$ . If the deadline of the new task obtained by the TBS is greater than the actual deadline of the new task,  $d_{TBS} > d_a$ , then the scheduling time of the new task is modified to  $t_a^* = d_{TBS} - P_a$ . Recall that  $P_a$  is the period of the new task. On the other hand, if  $d_a > d_{TBS}$  the new task is scheduled at its arrival time ( $t_a$ ). In any case, old tasks may be preempted by the new task. Finally, with the start time of the new task computed by the TBS server, and the solution provided by the optimization procedure (the set of speeds for execution), the PORTS Server will schedule the new task in the system following the EDF scheduling policy.

## 6. EXAMPLE

The purpose of this section is to use an example to simulate the execution of our algorithms and to compare their performance against several known algorithms.

- Dynamic Programming (DP)*: This algorithm was designed to solve to optimality the MCKP problem [S. Martello 1990].
- Maximum Speed (MS)*. In this algorithm, the processor is set to its maximum speed (in this case, maximum speed = 1.0). This algorithm is introduced with the purpose of finding the minimum energy savings achievable in the system, and for comparing its performance against our algorithms.
- Static Continuous (SC) and Static Discrete (SD) Algorithms*. In the SC algorithm [H. Aydin 2001a], the processor speed for all tasks is set to the system utilization (i.e.,  $f_i = \sum U_{i1}$ ). The static discrete solution uses the continuous solution (SC) and approximates its results. For example, if the speed levels of the processor are  $\{1.0, 0.75, 0.5, 0.25\}$ , and the utilization of the task set is 0.6, then the speeds of all tasks are set to 0.75.
- Optimal Continuous OP(c) and Optimal Discrete Algorithms OP(d)*. The continuous algorithm OP(c) [H. Aydin 2001a] considers tasks with different power characteristics and provides a continuous solution. The discrete solution OP(d) approximates the continuous solution using discrete levels of speed, in the same way that SD approximates SC.

Consider the real-time workload described in Table I. The value of  $c$  used in this example is 1000 (this value denotes 100% of the size of the knapsack). The workload described was computed assuming maximum speed levels for each task, obtaining a total load of  $\sum_i U_i = 60\%$ . In this example, a power consumption function  $P_d = k \cdot V^3$  is considered, where  $k$  is shown in Table I. Recall that energy consumption is computed by  $E = P_d * I$ , where  $I$  is the interval of time that some task is executing.

Table II shows the energy consumption  $E_{ij}$ , energy savings  $S_{ij}$  and utilization  $U_{ij}$  for the set of speeds of all tasks,  $N_i = \{1.0, 0.9, 0.7, 0.5, 0.3\}$ . Energy consumption measurements are simulated for a period of 32,000 time units, which is larger than the least common multiple of all the task periods  $P_i$ . For task  $T_1$ , energy consumption is computed as follows. Considering an interval of time  $I = 32000$ ,  $t_i = \frac{C_i}{f_i}$  (the maximum speed is 1.0), and  $P_1 = 1600$ , the number of instances of  $T_1$  in  $I$  is equal to  $\frac{32000}{1600} = 20$ . The energy consumption of task  $T_1$  is  $E_1 = k \cdot f_1^3 * \frac{C_1}{f_1} * 20 = 2 * 216 * 20 = 8640$ .

Table I. Real-Time Workload and Parameter  $k$ 

Tasks	$C_i$	$P_i$	$U_i$	k
Task 1	216	1600	0.135	2
Task 2	228	2000	0.114	2
Task 3	300	2000	0.150	8
Task 4	1551	8000	0.194	4

Table II. Energy Consumption, Savings and Utilization

		Speed Levels				
		1.0	0.9	0.7	0.5	0.3
Task 1	$E_{ij}$	8640	6998	4234	2160	778
	$S_{ij}$	0	1642	4406	6480	7862
	$U_{ij}$	135	150	192	270	450
Task 2	$E_{ij}$	7296	5910	3575	1824	657
	$S_{ij}$	0	1386	3721	5472	6639
	$U_{ij}$	114	126	162	228	380
Task 3	$E_{ij}$	38400	31104	18816	9600	3456
	$S_{ij}$	0	7296	19584	28800	34944
	$U_{ij}$	150	166	214	300	500
Task 4	$E_{ij}$	24816	20101	12160	6204	2233
	$S_{ij}$	0	4715	12656	18612	22583
	$U_{ij}$	193	215	276	387	646

Table III. Result from Problems  $P_1$  and  $P_4$ 

	Problem $P_1$				Problem $P_4$			
	$S_{ij}$	$U_{ij}$	$P_{ij}$	$W_{ij}$	$P_{ij}$	$W_{ij}$	$P_{ij}$	$W_{ij}$
Task 1	1642	15	4406	57	6480	135	7862	315
Task 2	1386	12	3721	48	5472	114	6639	266
Task 3	7296	16	19584	64	28800	150	34944	350
Task 4	4715	22	12656	83	18612	194	22583	453

Table III show the results of the truncation procedure described in Problem  $P_1$ . This table is constructed with elements from Table II, removing the elements with minimum  $U_{ij}$  value. Note that, the elements  $S_{11}$  and  $U_{11}$  in Table III, Problem  $P_1$ , are computed by  $S_{11} = 1642 - 0 = 1642$  and  $U_{11} = 150 - 135 = 15$ . Note that 0 and 135 are the energy savings and utilization with minimum values from task  $T_1$  in Table II. Note that  $S_{min}$  and  $U_{min}$  values are,  $S_{min} = \sum_{i=1}^4 S_{i,1} = 0$  and  $U_{min} = \sum_{i=1}^4 U_{i,1} = 592$ . After building Table III all minimum items from each task are included into the knapsack. So, the remaining size of the knapsack is:  $\hat{c} = (1000 - U_{min}) = 408$ . The result of the slicing procedure of Problem  $P_4$  is shown in the right hand side of Table III. Following the slicing procedure of Problem  $P_4$ , it is easy to verify that  $P_{14} = S_{14} - S_{13} = 7862 - 6480 = 1382$  and  $W_{14} = U_{14} - U_{13} = 315 - 135 = 180$ . Finally, ordering the elements in Table III by decreasing *energy/utilization* ratio produces the array  $PW$ , which is shown in Table IV.

Following the execution of the Greedy Algorithms (see Table IV) it is easy to verify that their solution is,

Table IV. Array  $PW$ : Greedy and Enhanced Greedy Algorithms

$P_{ij}$	7296	12288	4715	7941	1386	1642	9216	2764	2335
$W_{ij}$	16	48	22	63	12	15	86	42	36
$\sum S$	7296	19584	24299	32240	33626	35268	44484	47248	49583
$\sum Load$	16	64	86	149	161	176	262	304	340
$\frac{Included}{Not-Included}$	I	I	I	I	I	I	I	I	I
$P_{ij}$	5956	6144	2074	1751					
$W_{ij}$	111	200	78	66					
$\sum S$				51334					
$\sum Load$				406					
$\frac{Included}{Not-Included}$	B-S	N-I	N-I	I					

SGA Algorithm:  $S = 49583$ , and  $U = 340$ .

EGA Algorithm:  $S = 51334$ , and  $U = 406$ .

Note that *slice* (5956,111) in  $PW$ , is the *break-slice* (B-S). According to our Restoring algorithm, the minimal elements from each set  $N_i$  that were truncated in Problem  $P_1$  must be added to the solution. Then  $S_{min}$  and  $U_{min}$ , must be added to the solution. Therefore, the solution to the MCKP Problem  $P_0$  is as follows.

SGA Algorithm:  $S = 49583$ ,  $U = 932$ , and solution vector = [3,3,4,3]

EGA Algorithm:  $S = 51334$ ,  $U = 998$ , and solution vector = [3,4,4,3]

In Table V, the energy consumption solution for several algorithms is shown, along with the solution vector, the run-time (measured in microseconds on a PII-233 MHz) and the percentage of energy savings normalized to the SD algorithm. Note that the OP(c) algorithm provides the best energy consumption results. However, because of the fact that we are using discrete levels of speed, its corresponding discrete solution (obtained by the OP(d) Algorithm), gives a performance lower than that of our Greedy Algorithms. Note that, the maximum speed algorithm achieves more than double (e.g., 104%) of energy savings than that of the SD algorithm. The solution vectors of algorithms OP(c) and SC are shown in Table V as  $[x, x, x, x]$  because they do not yield discrete levels of speed. In the SC algorithm the speed is set to the value of the utilization (0.6).

Note that for this example, the Convex Hull of Problem  $P_3$  will give no P-dominated elements. This result is obtained because the speeds for each task are the same on each  $N_i$  and because on each  $N_i$  there is a constant increase in the value of each item. Recall that in this example, the set of speeds  $N_i$  for all tasks are  $\{1.0, 0.9, 0.7, 0.5, 0.3\}$ . This arrangement on the set of speeds produces a Convex Hull with only P-Undominated points (see Figure 2) and may lead to a simplified algorithm without the Convex Hull Problem. In this special case, we may remove lines 10–16 from the Optimization Procedure in Figure 4.

According to the results obtained in this example, we can verify that our Greedy Algorithms can obtain results that are close to the optimal (discrete) solution obtained by the Dynamic Programming Algorithm.

Table V. Results for different algorithms

Algorithm	Energy	Solution-Vector	Run-Time	% Savings
DP	26337	[2,3,4,4]	3980	33 %
MS	79152	[1,1,1,1]	586	- 104 %
SC	28495	[x,x,x,x]	586	27 %
SD	38784	[3,3,3,3]	586	0 %
OP(c)	25711	[x,x,x,x]	1106	34 %
OP(d)	34668	[2,2,4,3]	709	11 %
SGA	29569	[3,3,4,3]	47	24 %
EGA	27818	[3,4,4,3]	58	29 %

## 7. ANALYSIS OF THE ALGORITHM

In this section we provide a theoretical justification of our method. Our aim here is to prove that our method produces an 2-approximate solution to problem  $P_0$ .

### 7.1 2-Approximate Solution to Problem $P_0$

To justify the next claims, let us first state the notion of *r-approximation* [S. Martello 1990]. An approximation algorithm used in a maximization problem is called *r-approximate* if for every instance  $I$  of the problem it delivers a solution  $x$  of value  $A_I(x)$  such that  $z_I^*/A_I(x) \leq r$ , where  $z_I^*$  is the optimal solution value for the instance  $I$ .

The value  $r$  is called the worst case error bound, or the guaranteed performance ratio; the solution  $x$  is called *r-approximate*.

LEMMA 7.1. *An optimal solution to Problem  $P_4$  yields an optimal solution to  $P_2$ .*

PROOF. To our knowledge, this fact was first established by Sinha and Zoltners [P. Sinha 1978] and D'Atri [D'Atri 1977]. Since the proof can be found in many books and papers (e.g., [R.D. Armstrong 1983; S. Martello 1990; Pisinger 1995]) it is omitted here.  $\square$

In what follows, it will be shown that our method produce a 2-approximate solution to  $P_0$ .

According to the definition of problem  $P_3$  we can conclude that the optimal solutions of problems  $P_2$  and  $P_3$  are the same.

Let  $S_{max}$  be the maximum value of energy savings on all sets  $N_i$  in  $P_2$ ,  $S_{max} = \max_{ij} (S_{ij} - S_{i1})$ , and let

$$\{(1, j_1^*), (2, j_2^*), \dots, (k, j_k^*)\} \quad (7)$$

be the  $k$  ( $k \leq n$ ) items chosen from the sets  $N_i$  ( $i=1, \dots, n$ ) that form the optimal solution to the problem  $P_3$ . For example,  $(k, j_k^*)$  denotes the chosen item,  $j_k^*$ , from set  $N_k$ . In what follows, we assume that all the items in each set  $N_i$  are numbered in the decreasing order of their ratio  $\frac{R_{ij}}{H_{ij}}$ .

LEMMA 7.2. *An adjustment to the optimal solution of Problem  $P_3$ ,  $\max (S_{max}, R_{(1, j_1^*)} + R_{(2, j_2^*)} + \dots + R_{(k-1, j_{k-1}^*)})$ , yields a 2-approximate solution to  $P_1$ .*

PROOF. This claim has been formulated, without a proof, by Lawler [Lawler 1979]. Aiming to make this paper self-contained, we present here a short proof.

Obviously, the variables  $y_{ij}$  corresponding to the first  $(k-1)$  items in Equation (7) are equal to 1 (they are chosen to be part of the solution), while the  $k^{th}$  item satisfies the condition in  $P_3$ :  $0 < y_{(k,j_k^*)} \leq 1$ . If  $y_{(k,j_k^*)} = 1$  then the relaxed solution is integer. The obtained solution is the optimal solution to  $P_2$ , and the claim is true. Assume now that  $y_{(k,j_k^*)} < 1$ . Obviously, the first  $(k-1)$  items in Equation (7) constitutes a feasible approximate solution to  $P_1$  as their total utilization is not greater than  $(c - U_{min})$ . That is,

$$R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)} \leq Z_1^* \quad (8)$$

where  $Z_1^*$  is the optimal solution to problem  $P_1$ .

Consider the item  $S_{max}$ . This single item also constitute a feasible approximate solution to  $P_3$  as its utilization is not greater than  $(c - U_{min})$ . That is

$$S_{max} \leq Z_1^* \quad (9)$$

Then we have:  $max(S_{max}, R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)}) \leq Z_1^*$ .

Next,

$$\begin{aligned} Z_1^* &\leq R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)} + R_{(k,j_k^*)} \leq \\ &R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)} + S_{max} \leq \\ &R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)} + S_{max} \leq \\ &2 \max(S_{max}, R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)}) \end{aligned} \quad (10)$$

The claim is proved.  $\square$

LEMMA 7.3. *A 2-approximate solution to Problem  $P_1$  yields a 2-approximate solution to the initial problem  $P_0$ .*

PROOF. Consider the following expression:

$$A = S_{min} + \max(S_{max}, R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)}) \quad (11)$$

Since  $S_{min} = \sum_{i=1}^n S_{i1}$  are the items truncated from  $P_1$ , it follows that the utilization  $U$  of the items in the vector  $\mathbf{x}$  corresponding to  $A$  does not exceed  $c$ , and, hence,  $A \leq Z_0^*$  (optimal solution of problem  $P_0$ ). On the other side, from Equation (10),

$$\begin{aligned} Z_0^* &= S_{min} + Z_1^* \leq \\ &S_{min} + 2\max(S_{max}, R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)}) \leq \\ &2S_{min} + 2\max(S_{max}, R_{(1,j_1^*)} + R_{(2,j_2^*)} + \dots + R_{((k-1),j_{(k-1)}^*)}) = 2A \end{aligned} \quad (12)$$

The claim is proved.  $\square$

## 8. SIMULATION EXPERIMENTS

The following simulation experiments have been designed to test the performance of the PORTS Server and its ability to achieve our optimality criteria using synthetic task sets. The goals in this simulation experiments are: (1) to measure the quality of the results over a large set of dynamic tasks that arrive and leave the system at arbitrary instants of time, and (2) to measure and compare the performance and run-time of our algorithms against known algorithms.

A simulator has been developed to illustrate the execution of a dynamic set of synthetic real-time tasks, energy consumption, and speeds of execution of the tasks. The algorithms used for comparison are: Optimal Dynamic Programming (DP) [S. Martello 1990], Static Discrete Algorithm (SD) [H. Aydin 2001a] and the Optimal Discrete Algorithm OP(d) [H. Aydin 2001a]. Each plot in the graphs represents the average of a set of 5000 task arrivals. The results shown in the graphs are compared with the SD Algorithm and the size of the knapsack used in the experiments is  $c = 1000$  (100% of the load).

Each task has a life-time ( $r_i$ ) that follows a uniform distribution between 30 and 200 instances. At the end of its life-time, the task leaves the system. The period  $P_i$  of each task follow a uniform distribution between 1000 and 16000 time units, such that the LCM of the periods is 32000.

The arrival time of task  $T_{i+1}$  is computed by  $a_{i+1} = \frac{P_{i+1} * r_{i+1}}{nt}$ , where  $nt$  is the actual number of tasks in the system at the time  $T_{i+1}$  was generated. For a given number of speeds, each speed level is computed proportionally between the maximum speed ( $f_{max} = 1.0$ ) and the minimum speed ( $f_{min} = 0.2$ ). For example, if there are 5 speed levels, the speed levels will be  $\{1.0, 0.8, 0.6, 0.4, 0.2\}$ . The utilization of task  $T_i$  under minimum speed,  $U_{min}$ , is chosen as a random variable with uniform distribution between 10% and 25%.  $C_i$  is computed by  $C_i = U_{min} \cdot f_{ij} \cdot P_i$ . For each speed, utilization  $U_{ij}$  is computed by  $U_{ij} = \frac{t_{ij}}{P_i}$ , and  $t_{ij} = \frac{C_i}{f_{ij}}$ .

The power functions for each task  $T_i$  used [C. M. Krishna 2000; Y. Shin ; V. Swaminathan 2001] are of the form  $k_i \cdot S_i^{x_i}$ , where  $k_i$  and  $x_i$  are random variables with uniform distributions between 2 and 10, 2 and 3 respectively. Then, the energy consumption for each task and each speed  $f_j$  is computed by  $E_{ij} = I \cdot (k_i \cdot f_j^{x_i} \frac{C_i}{f_j \cdot P_i})$ , where  $I$  is a fixed interval, given by  $I = LCM$ . The PORTS server executes at the speed of the *current* executing task, and the input to our Optimization Problem  $P_0$  is computed by Equation (3).

The performance of our algorithms is measured at each task arrival (and departure) according to the following metrics:

— **Percentage (%) of Energy Savings (%ES):** The solution obtained (in terms of Energy Consumption) by each algorithm for all tasks gives us the total energy consumption  $E_{tot} = \sum_{i=1}^n E_i$ . The solution provided by each algorithm is then compared with the solution obtained by algorithm SD, and the percentage of improvement is plotted in the graphs. The results shown in the graphs represent the average ( $\bar{X} = (\sum_{i=1}^{5000} \%ES) / 5000$ ) or *sample mean* of a set of 5000 experiments (each for every task arrival). On each result of %ES we also compute their corresponding confidence intervals (for the mean) assuming a confidence level of 95%.

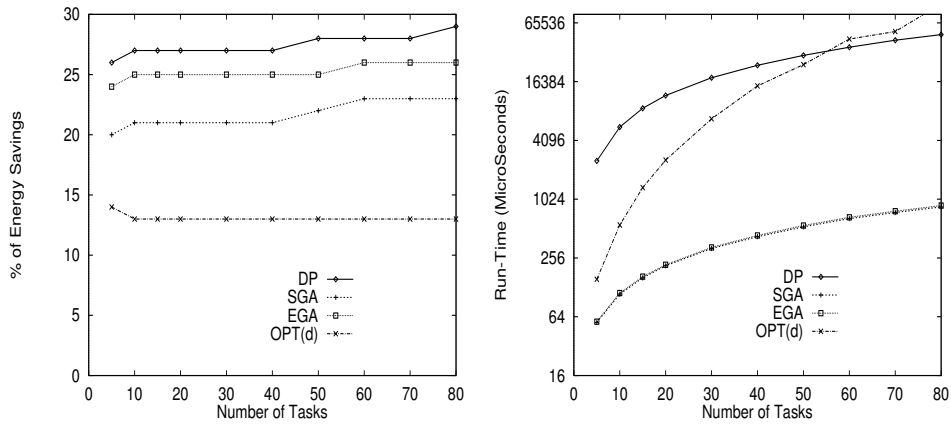


Fig. 5. % of Energy Savings and Run-Time (Microseconds)

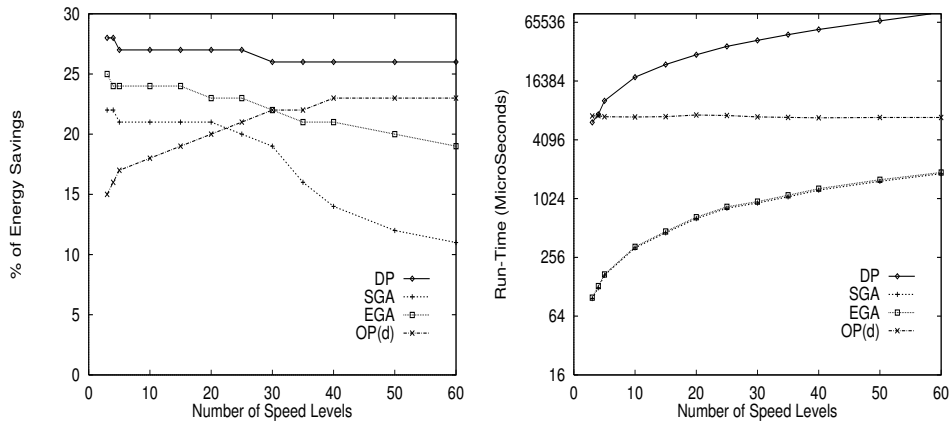


Fig. 6. % of Energy Savings and Run-Time (Microseconds)

- Run-Time:** This metrics denotes the execution time of each algorithm, which measures the physical time in microseconds, using a PC Intel 233 MHZ with 48MB of RAM and running on the Linux operating system. The function used for the measurements is `gettimeofday()`.
- Rejection Ratio (RR):** This metric denotes the percentage of tasks rejected from execution. A new task is rejected if it produces an overload. That is, a new task is rejected if the following condition is met  $\sum_i U_i > 100\%$ , while setting all tasks at maximum speed.

We demonstrate the performance of our algorithms with two simulation cases. The first case (Figure 5), considers 10 speed levels, and the number of tasks is varied from 5 to 80. In the second case (Figure 6), we show the influence of the granularity of the speed changing steps: the number of tasks is set to 30, and the speed level is varied from 3 to 60. The results obtained by algorithm EGA (shown

in Figure 5) vary from 89% to 96% of the DP Algorithm (we use DP as our *optimal* Algorithm), with % of energy savings ranging from 23% to 26%.

The SGA performs from 79% to 89% of optimal, with energy savings ranging from 19% to 22%. This results show an improvement of over 40% from the results obtained by the OP(d) algorithm. It is important to note that, although unrealistic due to the lack of discrete speed levels, the continuous OP(c) algorithm was also simulated to give us an upper bound on the energy savings; it yields between 24% and 28% energy savings, showing that our heuristics perform very well with respect to energy savings.

The results shown in Figure 5 (right side) indicate the low cost of the enhanced greedy algorithms. For the SGA and EGA algorithms the run-time varies from 56 to 853 microseconds. Note the large difference in run-time obtained by the EGA algorithms, when compared with the DP and the OP(d) algorithms: OP(d) varies from 155 to 102500 microseconds, and DP varies from 2529 to 49653 microseconds. The Rejection ratio (RR) in these simulations (Figure 5) is as follows. From 0 to 30 tasks, rejection ratio is equal to  $RR = 0\%$ . For 40 tasks  $RR = 7.5\%$ , which indicates that 3 out of 40 tasks, in average, are rejected from execution. For 50 tasks  $RR = 24\%$ , for 60 tasks  $RR = 35\%$ , for 70 tasks,  $RR = 42.8\%$ , and for 80 tasks,  $RR = 47.5\%$ . From the % of energy savings in Figure 5, the confidence intervals computed for 10 to 80 tasks are within the range  $[-7\% \text{ } \%ES, +7\% \text{ } \%ES]$ . Table VI shows the results of Figure 5 with 40 Tasks and their corresponding confidence intervals (assuming a confidence level of 95%).

The results shown in Figure 6 indicate how important is to consider an appropriate number of speed levels for achieving a high percentage of energy savings. As shown in Figure 6, under moderate, *realistic* number of speed levels (between 3 and 30), the EGA algorithm outperforms the OP(d) algorithm. However, for more than 30 speed levels OP(d) algorithm outperforms the EGA algorithm, because the system approaches a continuous voltage setting, in which OP(d) is close to the optimal OP(c).

The run-time computed (shown in Figure 6), indicate that the OP(d) algorithm has very little sensibility to the number of speed levels: the run-time of the OP(d) algorithm varied from 6900 to 7100 microseconds. In contrast, our Greedy Algorithms increased their run-time with higher number of speed levels, but still remained well under OP(d). For this experiments, the run-time of the Greedy Algorithms varied from 99 to 1800 microseconds. The DP algorithm is the most expensive, with one or two orders of magnitude higher run-time than the EGA and SGA algorithms. The Rejection ratio for this experiment was always 0% (because the number of tasks considered for the experiment shown in Figure 6 is 30) meaning that no tasks were rejected from execution on any single test.

Further tests were conducted (increasing the number of speed levels) to check when the EGA algorithm and the OP(d) algorithms have similar run-times; this happens when the number of speed levels is approaching 100.

The results obtained in our simulations indicate that the Enhanced Greedy Algorithms are a low cost and effective solutions for scheduling power-aware real-time tasks with discrete speeds.

Table VI. Confidence Intervals for 40 Tasks

	SGA	SGA	OP(d)	DP
% ES	21	25	13	27
CI	[20.1,21.8]	[24.1,25.3]	[12.6,13.3]	[25.6,27.9]

## 9. CONCLUSIONS

In this paper we proposed an adaptive power optimization method for a real-time application running on a variable speed processor with discrete speeds. The solution proposed is based on the use of a Power-Optimized Real-Time Scheduling (PORTS) Server which is comprised of two parts: (a) a feasibility test, for testing the admission of new dynamic tasks arriving in the system, and (b) an optimization procedure used for computing the levels of speed of each task in the system, such that energy savings of the system is maximized. The process of selecting levels of voltage/speed for each tasks while meeting the optimality criteria requires the exploration of a potentially large number of combinations, which is intractable to be done on-line. The PORTS Server finds near-optimal solutions at low cost by using approximate solutions to the knapsack problem. We have presented an analysis to demonstrate that our method produces a 2-approximate solution to the optimization problem.

Our simulation results show that our PORTS Server has low overhead, and most importantly generates near-optimal solutions for the scheduling of real-time systems running on variable speed processors.

We are currently extending the PORTS Server with algorithms for multiple processors and for real-time tasks with precedence and resource constraints.

## REFERENCES

- A. DUDANI, F. MUELLER, Y. Z. 2002. Energy-conserving feedback edf scheduling for embedded systems with real-time constraints. In *ACM SIGPLAN Joint Conference on Languages, Compilers and Tools for Embedded Systems (LCTES'02)*. ACM Press.
- A.P. CHANDRAKASAN, S. SHENG, R. B. 1992. Low-power cmos digital design. *IEEE J. of Solid-State Circuits* 27.
- ARM. 2003.
- C. M. KRISHNA, Y. H. L. 2000. Voltage clock scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proc. of the IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society Press.
- C.L. LIU, J. L. 1973. Scheduling algorithms for multiprogramming in hard real-time environments. *Journal of the ACM* 20, 1 (Jan).
- D. MOSSÉ, H. AYDIN, B. C. R. M. 2000. Compiler assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compiler and Operating Systems for Low Power (COLP'00)*.
- D. SHIN, W. KIM, J. J. J. K.-S. M. 2002. Simdvs: An integrated simulation environment for performance evaluation of dynamic voltage scaling algorithms. In *Proc. of the Workshop on Power-Aware Computer Systems (PACS'02)*.
- D'ATRI, G. 1977. The generalized knapsack problem. In *Communication at the Annual Meeting of CNR-GNIM, Rimini, Italy*.
- F. GRUIAN, K. K. 2001. Lenestask scheduling for low energy systems using variable supply voltage processors. In *Proc. Asia South Pacific - DAC Conference 2001*.
- F. YAO, A. DEMERS, S. S. 1995. A scheduling model for reduced cpu energy. In *Proc. of the IEEE Annual Foundations of Computer Science*.

- G. BUTTAZZO, F. S. 1999. Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments. *IEEE Transactions on Computers* 48, 10 (October).
- G. LIPARI, G. B. 2000. Schedulability analysis of periodic and aperiodic tasks with resource constraints. *J. of Systems Architecture* 46, 327.
- H. AYDIN, R. MELHEM, D. M. P. M.-A. 2001a. Determining optimal processor speeds for periodic real-time tasks with different power characteristics. In *Proc. of the IEEE EuroMicro Conference on Real-Time Systems*. IEEE Computer Society Press.
- H. AYDIN, R. MELHEM, D. M. P. M.-A. 2001b. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *IEEE Real-Time Systems Symposium*. IEEE Computer Society Press.
- I. HONG, D. KIROVSKI, G. Q. M. P.-M. S. 1998a. Power optimization of variable voltage core-based systems. In *Design Automation Conference*.
- I. HONG, M. POTKONJAK, M. B. S. 1998b. On-line scheduling of hard real-time tasks on variable voltage processor. In *Computer-Aided Design (ICCAD) '98*.
- I. HONG, G. QU, M. P. M. S. 1998c. Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proc. of 19th IEEE Real-Time Systems Symposium*. IEEE Computer Society Press.
- INTEL. *StrongARM SA-1100 microprocessor developer's manual*. INTEL.
- INTEL-ACPI. 2003. *ACPI Specification*. <http://developer.intel.com/technology/IAPC/tech>.
- INTEL-XSCALE. 2003. <http://developer.intel.com/design/xscale/>.
- J.R. LORCH, A. S. 2001. Improving dynamic voltage scaling algorithms with pace. In *Proc. of ACM SIGMETRICS Conference*. Cambridge, MA.
- LAWLER, E. 1979. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research* 4, 339.
- M. JOSEPH, P. P. 1986. Finding response times in a real-time system. *Computer Journal* 29, 390-395.
- P. PILLAI, K. G. S. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th. ACM Symposium on Operating System Principles (SOSP'01)*. ACM Press.
- P. SINHA, A. Z. 1978. The multiple choice knapsack problem. *J. of the Operations Research Soc. Japan* 21, 59.
- PISINGER, D. 1995. A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research* 83.
- R.D. AMSTRONG, D.S. KUNG, P. S. A. Z. 1983. A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software* 9, 2, 184-198.
- S. MARTELLO, P. T. 1990. *Knapsack Problems. Algorithms and Computer Implementations*. Wiley.
- T. ISHIHARA, H. Y. 1998. Voltage scheduling problem for dynamically varying voltage processors. In *Proc. Int'l Symposium on Low Power Electronics and Design*.
- T.D. BURD, T.A. PERING, A. S. R. B. 2000. A dynamic voltage scaled microprocessor system. *IEEE J. of Solid-State Circuits* 35, 11 (Nov.), 1571.
- TRANSMETA. 2003.
- V. SWAMINATHAN, K. C. 2001. Investigating the effect of voltage-switching on low-energy task scheduling in hard real-time systems. In *Proc. Asia South Pacific - DAC Conference*.
- Y. SHIN, K. C.