Elastic DVS Management in Processors With Discrete Voltage/Frequency Modes

Mauro Marinoni and Giorgio Buttazzo, *Senior Member, IEEE*

***Abstract*—Applying classical dynamic voltage scaling (DVS) techniques to real-time systems running on processors with discrete voltage/frequency modes causes a waste of computational resources. In fact, whenever the ideal speed level computed by the DVS algorithm is not available in the system, to guarantee the feasibility of the task set, the processor speed must be set to the nearest level greater than the optimal one, thus underutilizing the system. Whenever the task set allows a certain degree of flexibility in specifying timing constraints, rate adaptation techniques can be adopted to balance performance (which is a function of task rates) versus energy consumption (which is a function of the processor speed).**

In this paper, we propose a new method that combines discrete DVS management with elastic scheduling to fully exploit the available computational resources. Depending on the application requirements, the algorithm can be set to improve performance or reduce energy consumption, so enhancing the flexibility of the system. A reclaiming mechanism is also used to take advantage of early completions. To make the proposed approach usable in real-world applications, the task model is enhanced to consider some of the real CPU characteristics, such as discrete voltage/frequency levels, switching overhead, task execution times nonlinear with the frequency, and tasks with different power consumption. Implementation issues and experimental results for the proposed algorithm are also discussed.

***Index Terms*—Dynamic voltage scaling (DVS), energy-aware scheduling, real-time computing.**

I. Introduction

In battery-powered real-time systems, reducing energy consumption through dynamic voltage scaling (DVS) techniques may create overload conditions that can jeopardize the schedulability of the task set. Hence, the issue of reducing energy consumption must be considered in conjunction with the one of meeting timing constraints. Moreover, in current processors, the voltage level cannot be varied continuously, but only a limited number of voltage/frequency operating modes are usually available, causing the processor to run at a speed selectable within a discrete range. In these conditions, the speed selected by the power manager will likely be different than the ideal one that could minimize some cost function; thus, either timing constraints are not met or energy is not minimized.

Manuscript received July 24, 2006; revised October 25, 2006; accepted November 20, 2006. This work was supported in part by the Italian Ministry of University Research under Contract 2004095094 (COFIN04). Paper no. TII-06-07-0070.R1.

M. Marinoni is with the University of Pavia, Pavia, Italy (e-mail: mauro.marinoni@unipv.it).

G. Buttazzo is with the Scuola Superiore Sant'Anna of Pisa, Pisa, Italy (e-mail: giorgio.buttazzo@sssup.it).

Digital Object Identifier 10.1109/TII.2006.890494

Whenever a control application has hard real-time requirements (and the task set is feasible at the highest speed), energy and timing constraints can be met by setting the processor speed to the nearest level greater than the optimal one. In this way, however, if the difference between the ideal and the selected speed is not small, the processor becomes underutilized, so wasting computational resources. A better solution could be to increase task rates and improve control performance whenever possible. On the other hand, if reducing energy is more important and the application allows a certain degree of flexibility in specifying timing constraints, the processor speed could be set to the next lower level (just below the ideal speed), and the resulting overload could be prevented by slightly increasing the task periods.

In this paper, we present a novel DVS management algorithm that integrates energy-aware with elastic scheduling to cope with processors with a limited number of operating modes. To avoid wasting processing time due to speed quantization, we consider a more flexible task model, in which tasks can operate within a given range of periods, with different performance. The algorithm allows the application to select energy-oriented, performance-oriented, and user-defined strategies. The energy-oriented strategy sacrifices performance in favor of energy saving by selecting the lowest speed that guarantees the task set with the largest possible periods. The performance-oriented strategy selects the lowest speed that guarantees the task set with the smallest periods. Whenever the selected (discrete) speed level leaves some free processor bandwidth, elastic scheduling is invoked to reduce task periods to fully utilize the processor and increase the control performance. Finally, the user-defined strategy allows selecting any processor speed within the feasible range and adjusts task rates to fully utilize the processor. Speed selection can be done either manually or automatically, using the best solution found by the power manager according to a cost function provided by the user.

An online reclaiming mechanism is integrated in the algorithm to exploit the unused computation time resulting from early completions of the jobs. To better consider the effects of the hardware architecture on task execution times, we use an enhanced execution time model [16], [23] that splits the code in two parts: one that varies with speed and one that is speed independent. This enables a more precise representation of the application code, allowing the user to distinguish, for example, between code for pure computations and code accessing peripheral devices.

The proposed algorithm has been implemented in the Shark real-time operating system [9] as a new scheduling module, and experimental results have been derived on an Athlon64 3000+ processor.

The rest of this paper is organized as follows: Section II presents some related work; Section III introduces the models used to describe the execution time and the energy consumption of a task; Section IV describes the integrated DVS-elastic algorithm; Section V introduces metrics to evaluate the performance of a control system as a function of the task periods; Section VI describes some experimental results; and Section VII states our conclusions and future work.

II. Related Work

The problem of minimizing energy consumption while guaranteeing real-time constraints has been widely addressed in the real-time literature. However, most of the achieved results were derived under simplified system models, where the processor, for example, can change its voltage and frequency within a continuous range. Unfortunately, most of the current commercial processors [2], [11] only provide a limited number of operating modes, each characterized by a given voltage, clock frequency, and power consumption. Adapting a continuous model to a discrete DVS system clearly causes a waste of computational resources, because, in order to guarantee the feasibility of the task set with a single speed level, the processor speed must be set to the nearest level greater than the optimal one [21].

Other authors focused on energy-aware scheduling for specific task models. The most investigated task model is the periodic one [3], [4], [13], [28], but energy-aware algorithms have also been proposed and analyzed for aperiodic tasks [25], sporadic tasks [22], and mixed task sets [26]. To deal with processors having discrete speed levels, some authors proposed to split tasks into parts and run each part at a different frequency [12], [15], [28].

Mejia-Alvarez *et al.* [19] proposed an approach where each task is assigned a different frequency; however, the processor model is simpler than the one used in this paper, and the frequency assignment problem is NP-hard; thus, it can be treated online only by a heuristic algorithm.

More recently, Bini *et al.* [6] presented a method for approximating any speed level with two given discrete values, which are properly switched as a pulse width modulation signal to obtain its average value. Schedulability analysis to guarantee the feasibility of real-time task sets was also presented.

In order to guarantee task timing constraints, most of the algorithms for hard real-time systems perform the analysis assuming that each task executes for its worst-case execution times (WCETs). This is usually a strong conservative hypothesis that may cause a waste of computational resources. To exploit the additional slack coming from early completions, some authors [5], [13] proposed to mix an offline approach based on WCETs with an online method, which is in charge of reclaiming the unused computation time for a further reduction of the CPU frequency.

Reclaiming algorithms can be distinguished in two categories, known as inter-task and intra-task methods. Inter-task algorithms decide the working frequency on a task-by-task basis, while intra-task algorithms may adjust the frequency, even within a single task [20]. Some algorithms mixing intra-task and inter-task approaches have also been proposed [13].

The use of elastic scheduling has already been proposed for improving DVS management [16], but the method is based on WCET estimations and is only applicable offline; thus, all the unused computation time leads to a waste of energy. Moreover, the algorithm is presented without a set of tests to experimentally validate the approach.

In this paper, an online reclamation algorithm works in combination with an offline method to achieve a further reduction of the processor speed whenever a job completes earlier than expected. A set of experimental results are presented to show various aspects of the algorithm and describe the effects of some user-defined parameters.

III. Models

This section introduces the models adopted in this paper to represent task execution times and power consumption. Moreover, the elastic model is also briefly recalled for the sake of completeness. To simplify the comparison between processors with different frequency ranges $[f_{min}, f_{max}]$, all the quantities of interest (power, computation times, etc.) will be expressed as a function of speed, defined as the normalized frequency $s = f/f_{max}$. Hence, the validity range for the normalized speed is $[s_{min}, s_{max}]$, where $s_{min} = f_{min}/f_{max}$ and $s_{max} = 1$.

If more voltage levels can be used for a given frequency, the rule adopted in this paper is to select the minimum voltage level compatible with the frequency selected by the algorithm. This approach is in line with the selected power model and leads to a simple and fast implementation. It is a quite common solution adopted in most of the proposed algorithms, and it is also used in the CPUFreq driver for the Linux kernel.

A. Execution Time Model

Typically, task execution times are considered to be inversely proportional to the clock frequency and are modeled as $C_i(s) = C_{i_{max}}/s$, where $C_{i_{max}}$ is the task execution time at the maximum processor speed. Extensive experiments on real hardware, however, show that this assumption is not correct. A more accurate model is to split the execution time in two parts: one dependent on the CPU frequency and one independent. While the former part is due to the code that works with the processor or with the hardware running at the CPU frequency, the latter part comes from the code that uses hardware devices that are not affected by frequency changes. For example, the video output operates at the frequency of the PCI bus, so its execution time does not change with the CPU speed.

Let $C_{i_{max}}$ be the execution time evaluated at the maximum processor speed, and let ϕ_i be the percentage of code that deals with the frequency-dependent hardware. Then, the task execution time can be modeled as

$$C_i(s) = \frac{\phi_i C_{i_{max}}}{s} + (1 - \phi_i) C_{i_{max}}. \tag{1}$$

Unfortunately, classifying the code in the two parts described above is not easy, because the actual execution times depend on the architecture on which the task is running. For example, operations that rely on RAM memory are frequency-dependent

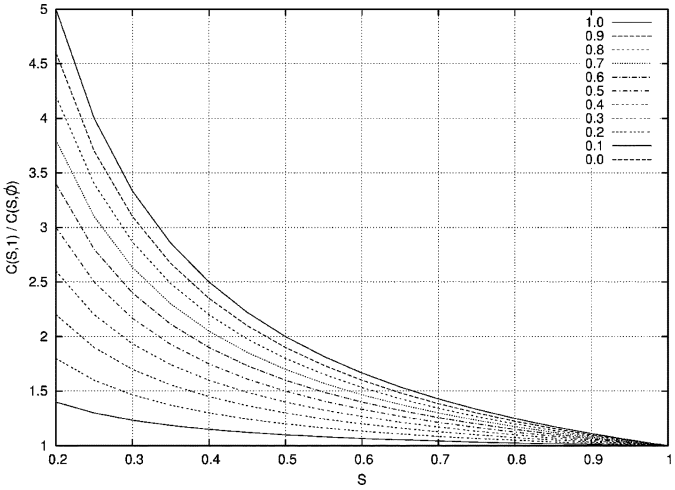


Fig. 1. C(s) as a function of the ϕ parameter.

if running on ARM processors and frequency-independent if running on $\times 86$ architectures.

The value of ϕ_i can be estimated experimentally by measuring the execution time of a task at the maximum frequency $(C_{i_{max}} = C_i(1))$ and at the minimum frequency $(C_{i_{min}} = C_i(s_{min}))$. In fact, by (1)

$$C_{i_{min}} = \frac{\phi_i C_{i_{max}}}{s_{min}} + (1 - \phi_i) C_{i_{max}}$$

thus, ϕ_i can be computed as

$$\phi_i = \frac{C_{i_{min}} - C_{i_{max}}}{C_{i_{max}}} \frac{s_{min}}{1 - s_{min}}. \tag{2}$$

It is worth noting that the simplified model that considers the execution time inversely proportional to the frequency (equivalent to the case $\phi = 1$) gets worse as the frequency decreases. For example, on a task with $\phi = 0.1$ running at speed 0.2, the computation time provided by the simplified model would be 3.57 times the real one.

Fig. 1 shows $C(s)$ as a function of the processor speed for different values of ϕ. It can be seen that the simplified model that considers the execution time inversely proportional to the frequency (equivalent to the case $\phi = 1$) gets worse as the frequency decreases.

B. Energy Consumption Model

In CMOS integrated circuits, the dominant component of power consumption is the dynamic power dissipation due to switching [10], which is given by

$$P = C_{eff} V_{dd}^2 f$$

where C_{eff} is the effective capacity involved in switching, V_{dd} is the supply voltage, and f is the clock frequency. The value of the capacity C_{eff} depends on two factors: the load capacity C being charged/discharged and the activity weight α, which is a measure of the actual switching activity. Thus, $C_{eff} = \alpha C$. Moreover, a voltage reduction causes an increase of the delays in the gates, according to the following formula:

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

where k is a constant, and V_t is the threshold voltage. Observing that the processor speed is directly proportional to the clock frequency f and inversely proportional to the gate delay, it turns out that the power consumption of a processor grows with the cube of its speed. The overall energy consumption of the system, however, also depends on other components of lower grade. Martin *et al.* [17], [18], [27] derived the following relation to describe the power consumption as a function of the speed:

$$P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0. \tag{3}$$

The K_3 term is a coefficient related to the consumption of those components that vary both voltage and frequency. The K_1 coefficient is related to the hardware components that can only vary the clock frequency, whereas K_0 represents the power consumed by the components that are not affected by the processor speed. Finally, the second-order term (K_2) describes the nonlinearities of dc-dc regulators in the range of the output voltage.

C. Elastic Task Model

In our framework, each task is considered as flexible as a spring, whose utilization can be modified by changing its period within a specified range. More specifically, each task is characterized by four parameters: a worst-case computation time C_i, which depends on the processor speed according to (1), a minimum period $T_{i_{min}}$ (considered as a nominal period), a maximum period $T_{i_{max}}$, and an elastic coefficient E_i. The elastic coefficient specifies the flexibility of the task to vary its utilization for adapting the system to a new feasible rate configuration: the greater E_i, the more elastic the task. Hence, we consider a set of n elastic tasks, where each task is indicated by

$$\tau_i \left(C_i, T_{i_{min}}, T_{i_{max}}, E_i\right).$$

In the following, T_i will denote the actual period of task τ_i, which is constrained to be in the range $[T_{i_{min}}, T_{i_{max}}]$. Moreover, $U_{i_{max}} = C_i/T_{i_{min}}$ and $U_{i_{min}} = C_i/T_{i_{max}}$ will denote the maximum and minimum utilization of τ_i, whereas $U_{max} = \sum_{i=1}^{n} U_{i_{max}}$ and $U_{min} = \sum_{i=1}^{n} U_{i_{min}}$ will denote the maximum and minimum utilization of the task set.

Note that both U_{max} and U_{min} depend on the processor speed; hence, any load variation due to a speed change is always subject to an *elastic* guarantee and is accepted only if there exists a feasible schedule in which all the periods are within their range. In our framework, tasks can be handled by any periodic scheduling algorithm with given utilization least upper bound (U_{lub}). Remember that for the earliest deadline first (EDF) algorithm, $U_{lub} = 1$, whereas for the rate monotonic algorithm, $U_{lub} = n(2^{1/n} - 1)$, where n is the number of tasks [14]. Hence, if $U_{max} \leq U_{lub}$, all tasks can be activated at the minimum period $T_{i_{min}}$; otherwise, the elastic

algorithm is used to adapt the tasks' periods to T_i such that $\sum(C_i/T_i) = U_d \leq U_{lub}$, where U_d is some desired utilization factor. It can be easily shown (see [7] for details) that a solution always exists if $U_{min} \leq U_d$.

As shown in [7], if Γ_f is the set of tasks that reached their maximum period (i.e., minimum utilization) and Γ_v is the set of tasks whose utilization can still be compressed, then to achieve a desired utilization $U_d < U_{max}$, each task has to be compressed up to the following utilization:

$$\forall \tau_i \in \Gamma_v \quad U_i = U_{i_{max}} - (U_{v_{max}} - U_d + U_f)\frac{E_i}{E_v} \tag{4}$$

where

$$U_{v_{max}} = \sum_{\tau_i \in \Gamma_v} U_{i_{max}} \tag{5}$$

$$U_f = \sum_{\tau_i \in \Gamma_f} U_{i_{min}} \tag{6}$$

$$E_v = \sum_{\tau_i \in \Gamma_v} E_i. \tag{7}$$

If there exist tasks for which $U_i < U_{i_{min}}$, then the period of those tasks has to be fixed at its maximum value $T_{i_{max}}$ (so that $U_i = U_{i_{min}}$), sets Γ_f and Γ_v must be updated (hence, U_f and E_v recomputed), and (4) applied again to the tasks in Γ_v. If there exists a feasible solution, that is, if $U_{min} \leq U_d$, the iterative process ends when each value computed by (4) is greater than or equal to its corresponding minimum $U_{i_{min}}$.

D. Overall Task Model

To integrate the execution time model with the elastic one, each task will be denoted as follows:

$$\tau_i\,(C_{i_{max}}, \phi_i, T_{i_{min}}, T_{i_{max}}, E_i)$$

where the meaning of the parameters has been explained in the previous sections.

IV. Algorithm Description

The algorithm proposed in this paper combines DVS management with elastic scheduling to enhance performance or reduce energy consumption in systems with discrete operating modes. In the following, we assume the task set is feasible when the processor runs at the maximum speed and all tasks execute at their maximum period, that is,

$$U_{min}(s_{max}) \leq U_d \tag{8}$$

(where $U_d \leq U_{lub}$); otherwise, no feasible solution can be found, and the task set is rejected by the feasibility test. The U_d parameter allows the user to account for the overhead introduced by the kernel, which can be measured offline. A value $U_d = U_{lub}$ should never be used, since other internal kernel activities (e.g., the interrupt handlers for the network or other peripheral devices) could create critical transient overload conditions.

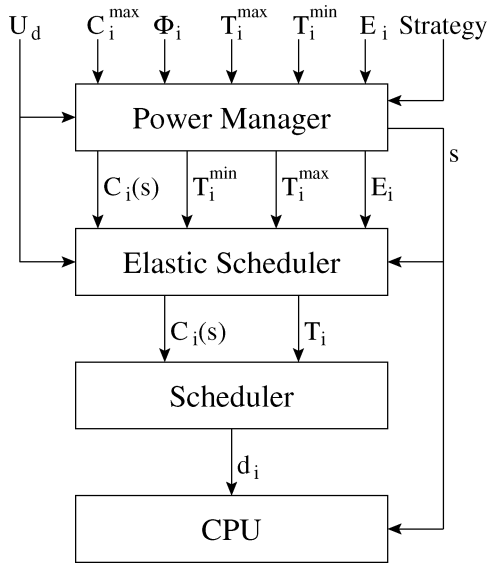


Fig. 2. Block diagram of the algorithm.

At the application level, the user can choose among three high level strategies.

- **Energy oriented**: energy consumption is minimized by selecting the lowest processor speed s_e that guarantees schedulability with the maximum periods; then, if $U_{min}(s_e) < U_d$ (note, strictly less), periods are reduced by the elastic algorithm to reach the desired utilization U_d, thus improving the control performance.
- **Performance oriented**: control performance is maximized by selecting the lowest processor speed s_p that provides full performance, that is, that guarantees schedulability with the minimum periods; if $s_p > s_{max}$, that is, if $U_{max}(s_{max}) > U_d$, then s_p is set to s_{max} and task periods are enlarged by the elastic algorithm to reach feasibility with the desired utilization U_d.
- **User mode**: this mode allows the user to manually select a speed level s_u included in the range $[s_e, s_p]$ defined by the two previous modes. If $U_{max}(s_u) > U_d$, periods are enlarged by the elastic algorithm to reach feasibility with the desired utilization U_d.

The algorithm consists of three hierarchical levels. At the top level, the power manager performs the acceptance test and computes the working speed s according to the selected strategy. At the medium level, the elastic scheduler computes the task periods and passes the task set to the system scheduler at the bottom level. We can see each level as a function that converts the input task model into a new one accepted at the lower level. The hierarchical structure of the algorithm is illustrated in Fig. 2. The power manager is invoked every time a new task enters/leaves the system or a new speed is selected by the application.

The algorithm starts by computing the ideal speed bounds (s_e^*, s_p^*) in the continuous speed domain; then, it selects the actual discrete speed limits (s_e, s_p), and finally, it invokes the

elastic algorithm to perform period adaptation. In addition, at runtime, a reclaiming mechanism is proposed to take advantage of early completions.

A. Computing the Frequency Bounds

The algorithm starts by computing a range of ideal speeds [s_e^*, s_p^*] in which the schedulability of the task set is guaranteed and there is no energy waste when the tasks run at their minimum periods.

In the energy-oriented strategy, assuming a single speed is used for the whole application, the minimum theoretical speed s_e^* (in a continuous range) is computed as the speed that minimizes energy consumption while guaranteeing the schedulability of the task set.

Considering the computation time model expressed in (1), the total processor utilization can also be expressed as a function of the processor speed as follows:

$$\begin{aligned} U(s) &= \sum_{i=1}^{n} \frac{C_i(s)}{T_i} \\ &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_i} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_i} \\ &= \frac{U_D}{s} + U_F \end{aligned} \tag{9}$$

where U_D is the processor utilization due to the frequency-dependent code, estimated at the maximum speed, whereas U_F is the one that is frequency-independent.

The minimum utilization (U_{min}) computed with the maximum periods can also be expressed as a function of speed as follows:

$$\begin{aligned} U_{min}(s) &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_{i_{max}}} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{max}}} \\ &= \frac{U_{D_{min}}}{s} + U_{F_{min}}. \end{aligned} \tag{10}$$

Imposing $U_{min}(s) = U_d$ (desired utilization), the related speed is given by

$$s_e^* = \frac{\sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{T_{i_{max}}}}{U_d - \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{max}}}} = \frac{U_{D_{min}}}{U_d - U_{F_{min}}}.$$

If s_e^* is out of the range [0,1], the task set is not feasible and it is rejected by the guarantee test.

In the performance-oriented strategy, if $U_{max}(s_{max}) > U_d$, the speed s_p^* that guarantees the best performance is clearly s_{max}. Otherwise, the best theoretical speed s_p^* to achieve full performance is computed as the minimum speed that guarantees schedulability with the nominal periods.

Considering that U_{max} can be expressed as

$$\begin{aligned} U_{max}(s) &= \sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{sT_{i_{min}}} + \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{min}}} \\ &= \frac{U_{D_{max}}}{s} + U_{F_{max}}. \end{aligned} \tag{11}$$

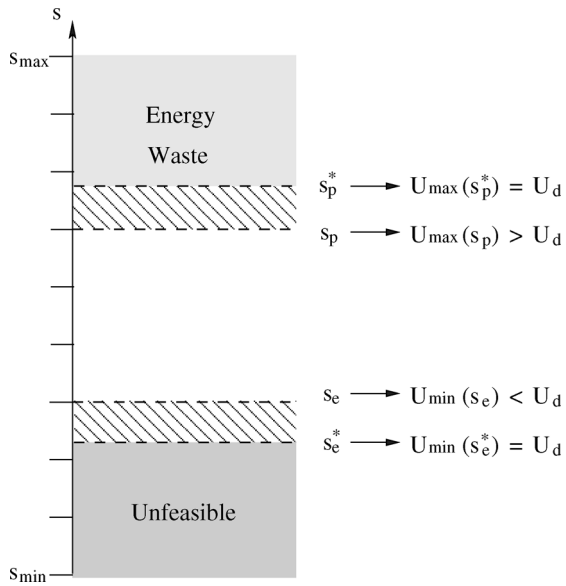


Fig. 3. Speed bounds computed by the algorithm for the energy-oriented and the performance-oriented mode.

Imposing $U_{max}(s) = U_d$, the best theoretical speed s_p^* is given by

$$s_p^* = \frac{\sum_{i=1}^{n} \frac{\phi_i C_{i_{max}}}{T_{i_{min}}}}{U_d - \sum_{i=1}^{n} \frac{(1-\phi_i)C_{i_{max}}}{T_{i_{min}}}} = \frac{U_{D_{max}}}{U_d - U_{F_{max}}}.$$

Hence, in general

$$s_p^* = \begin{cases} \frac{U_{D_{max}}}{U_d - U_{F_{max}}}, & \text{if } U_{max}(s_{max}) \le U_d \\ s_{max}, & \text{otherwise.} \end{cases}$$

B. Frequency Selection and Period Adjustment

Most commercial processors do not allow a continuous variation of voltage and frequency but only provide a limited number of operating modes, each characterized by specific values for supply voltage, frequency, and power consumption. Due to the discrete range of frequencies, it may not be possible to set the CPU speed at s_e^* or s_p^*. Hence, we set

$$s_e = \min_k \{s_k | s_k \ge s_e^*\} \tag{12}$$

$$s_p = \max_k \{s_k | s_k \le s_p^*\}. \tag{13}$$

Fig. 3 illustrates the speeds selected by the algorithm for the energy-oriented (s_e) and the performance-oriented (s_p) mode. Once the speeds s_e and s_p are computed and task set schedulability is guaranteed in the worst-case situation, there can be different strategies to select the operating speed as a function of the high level approach.

- **Energy-oriented**. If the objective is to minimize energy consumption, the actual speed is set to s_e. If $s_e > s_e^*$, then $U_{min}(s_e) < U_d$. Hence, to fully utilize the processor, task periods are reduced through elastic scheduling to bring the task set utilization at the desired level U_d.

- **Performance-oriented**. If the objective is to improve performance, the actual speed is set to s_p. Note that, if $U_{max}(s_p) \leq U_d$, all tasks can run at their nominal period and the elastic algorithm is not used; otherwise, task periods are expanded to reach the desired utilization U_d.
- **User mode**. Finally, if the user decides to select a specific speed $s_u \in [s_e, s_p]$ (among the available levels), the elastic method is invoked to reduce periods and reach the desired utilization U_d.

It is worth observing that, in the energy-oriented strategy, the elastic mechanism is always used to reduce periods to bring the processor utilization up to U_d, so improving the control performance whenever possible. Such an improvement is larger when the number of available speeds is small. Clearly, the values of computation times used in the elastic method are computed using the speed level set by the power manager or selected by the user.

C. Power Consumption and Elastic Coefficients

Another advantage of using the elastic approach in this context is that, if tasks have different power consumption, elastic coefficients can be set to reduce the energy of the tasks with higher power consumption. In fact, since the energy consumed by a task in a given interval is proportional to the number of jobs executed in that interval, elastic coefficients can be assigned so that tasks with higher power will be more compressed, that is, are subject to a larger period variation to decrease their energy consumption. To obtain this result, the elastic coefficient E_i can be set as

$$E_i \propto P_i(s)C_i(s) \tag{14}$$

where $P_i(s)$ is the power consumed by task τ_i, as defined in (3). Note, however, that since extracting energy consumption patterns from real applications is not a trivial task, a detailed analysis of such an approach requires a deeper investigation, which is out of the scope of this paper.

D. Online Reclamation of Unused Bandwidth

Real-time tasks are characterized by worst-case computation times, but most jobs usually run for much less than their worst-case value. Our DVS algorithm takes advantage of such a saving for a further reduction of the processor speed. The reclaiming algorithm has a tight interaction with the task scheduler, and its behavior depends on the particular scheduling policy adopted in the kernel [21].

When a task instance is released, determining its actual computational requirements is very hard (if not impossible), because a specific job could save a different percentage of frequency-dependent and frequency-independent code. Hence, a conservative worst-case assumption must be used. When the job completes, however, it is possible to compute the saved time by accounting the actual execution. Instead of wasting this time leaving the CPU idle, the processor can be slowed down to further reduce energy consumption. To do so, each time a job is released or completed, the computation of the working speed is performed using the actual status of the task set. For this purpose, a new variable e_i is added to the task model. It is the execution time actually consumed by the current job of task τ_i. Note that estimating such a variable requires a specific mechanism in the kernel capable of monitoring job execution times. In our experiments, computation time evaluations have been carried out using the job execution time (JET) monitor available in the Shark kernel [9].

TABLE I
JOB VARIABLE MANIPULATION

	Release	Completion	Preemption
e_i	0 or δ	$e_i + \delta$	$e_i + \sigma$

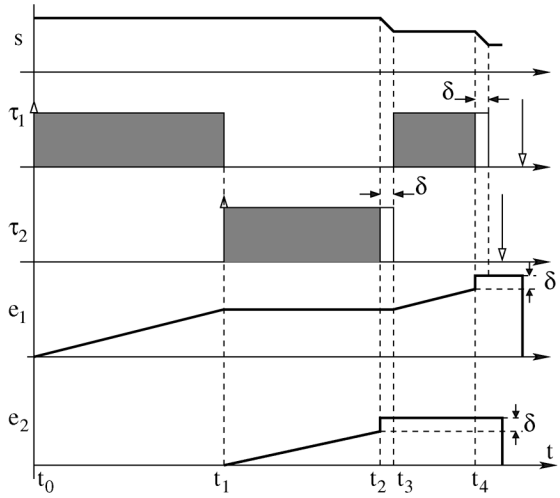


Fig. 4. Example for job variables updates.

This variable is updated upon the occurrence of the following events: job release, job completion, and job preemption.

- When a job is released, e_i is reset to zero. If the job activation triggers a speed switch, e_i is increased by δ, which is the overhead to switch the speed. Note that taking this overhead into account is extremely important since in some architectures, δ can be in the order of milliseconds.
- When a job completes, e_i is increased by δ. Such an increment at the end of the job is done because the e_i values are used to compute the actual speed from the current status of the task set, so they need to be consistent all the time.
- At a context switch, the variables are updated to account for the amount of used computation (σ).

The operations on the variable are summarized in Table I, whereas an example is presented in Fig. 4, which illustrates the behavior of two tasks running in the system.

In sequence, the different timelines show the processor speed, the task executions, and the e_i variable for tasks τ_1 and τ_2. At time t_0, the first task is released and starts executing, so e_1 starts growing while the task is running. At time t_1, a job of τ_2 is released, preempting τ_1. When τ_2 completes at time t_2, the speed is recomputed. The new minimum working speed is lower than the actual one, so the actual speed is changed and the switching time δ is accounted to τ_2. At time t_3, τ_1 resumes its execution that finishes earlier than expected at time t_4; hence, the unused computation time leads to a new speed reduction, and the switching time is accounted to e_1.

At job termination or activation, the reclaiming algorithm recomputes the working speed. Tasks periods are left unchanged to save runtime overhead and avoid extra jitter in task activations. If all tasks terminated the current job and the CPU has enough time to switch down the frequency (the next activation is at least ahead of δ), the chosen speed is s_{min}; otherwise, the minimum speed that guarantees the actual status of the task set is computed. The actual task utilization can be computed as

$$U_i(s) = \frac{1}{T_i}\left((1 - act(\tau_i))\, e_i + act(\tau_i)\left(C_{i_F} + \frac{C_{i_D}}{s}\right)\right)$$

where $act(\tau_i)$ is a function returning 0 if the current job of the task τ_i is completed or 1 otherwise.

So, the total utilization of the task set is expressed as

$$\begin{aligned} U(s) &= \sum_{i=1}^{n} U_i(s) \\ &= \sum_{i=1}^{n} \frac{1}{T_i}\left((1 - act(\tau_i))\, e_i + act(\tau_i)\left(C_{i_F} + \frac{C_{i_D}}{s}\right)\right) \\ &= \sum_{i=1}^{n} \frac{(1 - act(\tau_i))\, e_i}{T_i} + \sum_{i=1}^{n} \frac{act(\tau_i) C_{i_F}}{T_i} \\ &\quad + \frac{1}{s}\sum_{i=1}^{n} \frac{act(\tau_i) C_{i_D}}{T_i}. \end{aligned}$$

Imposing $U(s) = U_d$, the ideal dynamic speed s^*_{dyn} resulting from the reclaiming algorithm results to be

$$s^*_{dyn} = \frac{\sum_{i=1}^{n} \frac{act(\tau_i) C_{i_D}}{T_i}}{U_d - \sum_{i=1}^{n} \frac{(1 - act(\tau_i)) e_i}{T_i} - \sum_{i=1}^{n} \frac{(act(\tau_i) C_{i_F}}{T_i}}. \quad (15)$$

The chosen discrete frequency is the minimum one greater than or equal to s^*_{dyn}

$$s_{dyn} = \min_k \left\{ s_k | s_k \geq s^*_{dyn} \right\}. \quad (16)$$

If the reclaiming algorithm was triggered by the completion of a job τ_j and the proposed speed s_{dyn} is equal to the actual one, then the frequency switch time δ is subtracted from e_j.

V. Quality of Control

Note that, in control applications, the performance of periodic control tasks also depends on their activation rates. Hence, any scheduling approach manipulating task periods has to take into account the consequence of the imposed variations on the system performance. Increasing a task period causes a performance degradation, which is typically measured through a performance index $J(T)$ [8], [24]. Often, instead of using the performance index, many algorithms use the difference $\Delta J(T)$ between the index and the value of the performance index $J*$ of the optimal control. Many control systems belong to a class in which the function expressing the degradation is monotonically decreasing, convex, and can be approximated as

$$\Delta J(T_i) = \alpha_i e^{-\frac{\beta_i}{T_i}}$$

where the magnitude α_i and the decay rate β_i characterize the single task. The evaluation of the whole task set is computed as

$$\Delta J = \sum_{i=1}^{n} w_i \Delta J(T_i) = \sum_{i=1}^{n} w_i \alpha_i e^{-\frac{\beta_i}{T_i}}$$

where w_i are weights used to characterize the relative importance of the tasks.

To have a common scale for all task sets, the quality of control (QoC) index used in this paper is expressed as

$$QoC = \frac{\Delta J_{nom}}{\Delta J} \quad (17)$$

where ΔJ_{nom} is the value of the index calculated when tasks run at their nominal periods. A value of 1 means that all tasks are running with nominal periods.

VI. Experimental Results

To validate the proposed approach, the elastic-DVS algorithm has been implemented in the S.Ha.R.K. kernel [9] as an external scheduling module using EDF as a periodic task scheduler. Experiments were performed on an AMD Athlon64 3000+, whose clock can be set at four frequencies: 1000, 1800, 2000, and 2200 MHz, corresponding to the following normalized speeds: 0.4545, 0.8181, 0.9090, 1.

The first experiment is aimed at verifying the execution time model introduced in Section III, while the second one presents a qualitative comparison between the energy-aware and the performance-oriented strategy as a function of the workload. Then, a set of experiments are presented, each focusing on a different aspect of the proposed algorithm. The third experiment shows the advantages of using the elastic task model when working on a processor with discrete clock frequencies. The fourth experiment is used to test how the user-selected speed s_u affects the power consumption and the quality of control. The fifth experiment shows how the average power can be reduced by making elastic coefficients proportional to individual task power consumption. Finally, the last experiment illustrates the advantage of using the reclaiming mechanism.

All the tasks used in the last four experiments have the following characteristics.

- T_{min} was generated as a random variable uniformly distributed in the range [1, 100] milliseconds.
- T_{max} was generated as the product of T_{min} and a random number with Gaussian distribution, with mean equal to 4 and standard deviation equal to 2.
- C_{max} was computed in order to give each task the same fraction of the total utilization when running at the full speed with nominal period: $U_{max}(s_{max}) = U_{tot}/n$.
- ϕ was generated as a random value with Gaussian distribution (mean 0.5 and standard deviation 0.4).
- All the elastic coefficients were set equal to 1.

Finally, the desired utilization was set to $U_d = 0.9$ to take overheads into account, and all the weights in the quality of control index were set to 1 to simplify the comparison in the experimental results.

TABLE II
TASK EXECUTION TIMES (IN MILLISECONDS) AS A FUNCTION OF THE NORMALIZED SPEED

	speed			
	0.4545	0.8181	0.9090	1.0000
τ_1: Integer	2.796	1.554	1.399	1.271
τ_2: Float	2.752	1.529	1.376	1.251
τ_3: Text1	2.309	2.158	2.097	2.078
τ_4: Text2	2.727	1.794	1.678	1.582
τ_5: Graphics	2.506	1.839	1.756	1.687

TABLE III
ϕ VALUES AND ESTIMATION ERRORS

	ϕ	speed						
		0.4545		0.8181		0.9090		1.000
τ_1: Integer	1.0000	2.797	0.04%	1.554	0.00%	1.383	1.16%	1.271
τ_2: Float	1.0000	2.752	0.00%	1.529	0.00%	1.376	0.00%	1.251
τ_3: Text1	0.0926	2.309	0.00%	2.121	1.74%	2.097	0.00%	2.078
τ_4: Text2	0.6031	2.727	0.00%	1.794	0.00%	1.677	0.00%	1.582
τ_5: Graphics	0.4045	2.506	0.00%	1.838	0.05%	1.755	0.06%	1.687

A. Validating the Execution Time Model

This experiment was aimed at verifying the consistency of the execution model expressed by (1). To do that, we implemented a group of five periodic tasks with different characteristics (i.e., ϕ parameter), and the execution time of each task was estimated for all available speeds. The body of each task was composed as follows:

- τ_1: **Integer**: 7 000 000 operations on integer numbers;
- τ_2: **Float**: 90 000 floating point operations and trigonometric functions;
- τ_3: **Text1**: 700 000 integer operations mixed with the output of 9000 characters in text mode without screen scrolling;
- τ_4: **Text2**: same as τ_3 but with 2 100 000 integer operations and only 3000 characters;
- τ_5: **Graphics**: like τ_3 but with 1 500 000 integer operations and 150 characters printed in graphics mode.

Note that, although these tasks are not taken from a real control application, they represent a significant test case to evaluate the effectiveness of the execution time model proposed in this paper, because they include instructions with different balance of frequency-dependent and frequency-independent code, as shown by the Φ parameter reported in Table III.

Results are shown in Table II, which reports the mean execution times (in milliseconds), each obtained on 10 000 task activations.

Then, the ϕ_i value of each task was computed using (2), and measured values were compared against the theoretical values given by (1). Table III reports the ϕ value for each task and the computation time error of the measured value with respect to the theoretical one. It is worth observing that, even though ϕ changes from 0.0926 to 1, the relative error is less than 2%.

Fig. 5 shows the measured values of the execution times on the theoretical curves given by (1). It is worth observing that, using the simplified model of a fully frequency-dependent task ($C(s) = C_{i_{max}}/s$), the error would be much higher. For example, the theoretical execution time of task τ_3 (Text1) running at the lowest frequency would be 4572, while the real one is 2309 (98% smaller). Also notice that the measured values for tasks τ_1 and τ_2 both lie on the same curve corresponding to $\phi = 1$.

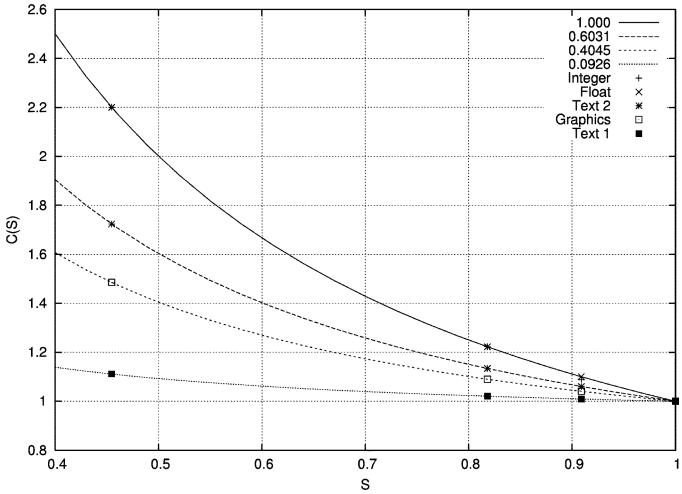


Fig. 5. Comparison between actual execution times and theoretical values.

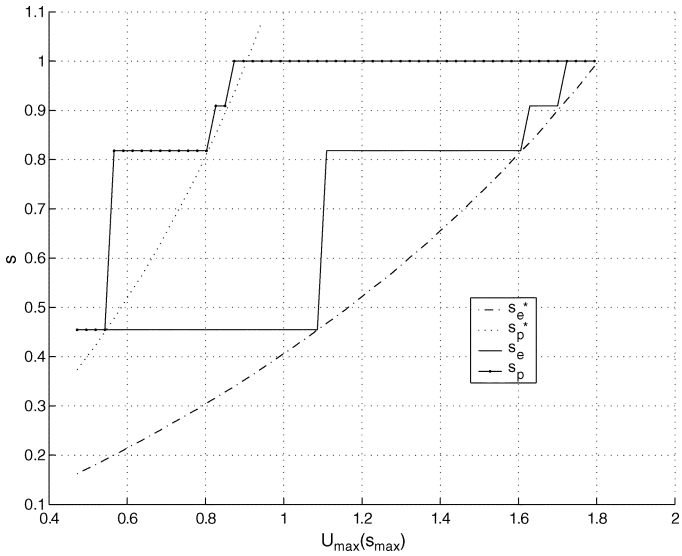


Fig. 6. Speed bounds computed by the algorithm as a function of the load for the energy-aware and the performance-oriented mode.

B. Energy-Aware versus Performance-Oriented Algorithm

In this experiment, we tested the behavior of the DVS-elastic algorithm as a function of the workload. The load was generated using the same set of tasks described in Section VI-A and varied by scaling the execution times to increase the maximum utilization $(U_{max}(s_{max}))$ from 0.45 to 1.8. All elastic coefficients were set to 1 for simplicity, and the desired utilization was set to $U_d = 0.9$ to take overheads into account. Fig. 6 illustrates the speed levels s_e^*, s_p^*, s_e, and s_p computed by the algorithm as a function of the load under the energy-aware and the performance-oriented mode, respectively.

As is clear from the graphs, when the load is less than U_d, even the performance-oriented strategy is able to reduce energy consumption, by finding the minimum speed that can guarantee all the tasks at their minimum periods. On the other hand, the energy-aware strategy allows a reduction in terms of energy consumption up to an overload of about 70% (i.e., $\text{load} = 1.7$),

when the maximum overload that the classical elastic algorithm can manage is 80% (i.e., $\text{load} = 1.8$), for the specific task set.

A significant improvement achieved with the integrated algorithm can be seen in Fig. 6 for $U_{max}(s_{max}) = 1.1$. In fact, without elastic scheduling, the task set would not be feasible with all tasks running at their nominal periods (T_{min}), and the use of a pure energy-aware algorithm with discrete speeds and maximum periods would waste processor utilization, penalizing performance (since tasks would run at the lowest possible rate). In fact, for this value of $U_{max}(s_{max})$, the utilization factor with maximum periods is 0.62, which is a lot less than the desired value $U_d = 0.9$. Using the proposed approach, the desired utilization can be reached with a normalized speed $s = 0.8181$, and tasks can run with shorter periods computed by the elastic algorithm, so improving the application performance.

C. Effects of the Elastic Task Model

This experiment is aimed at evaluating the advantages of using the elastic task model with respect to an approach with fixed task periods. In a first simulation, the energy-oriented approach was applied on a set of elastic tasks with periods in $[T_{min}, T_{max}]$ and compared with the case in which all task periods were fixed and equal to T_{max}.

Fig. 7 shows the results in term of selected speed (upper graph) and quality of control (lower graph) when the total utilization of the task set at full speed $(U_{max}(s_{max}))$ varies from 0.8 to 2.8 with a step of 0.1. For each step, the value on the y-axis represents the mean obtained on 100 task sets of 50 tasks each. We can see that, although the two algorithms select the same speed, the quality of control of the performance-oriented strategy is always higher. As expected, for high workloads, elastic tasks tend to run with larger periods; hence, the QoC decreases toward the one of the fixed task set. On the other hand, when the maximum utilization of the task set is less than one, the QoC achievable by the elastic approach is significantly higher than that obtained by the fixed tasks.

In a second simulation, the performance-oriented strategy was applied on a set of elastic tasks with periods in $[T_{min}, T_{max}]$ and compared with the case in which all task periods were fixed and equal to T_{min}. As in the previous experiment, results were derived in terms of speed and quality of control and are illustrated in Fig. 8. Since the performance-oriented approach also considers energy issues, the quality of control index is less than 1, where the value of 1 corresponds by definition to the performance index of the tasks running with the minimum periods. When the two algorithms select the same speed, the quality reduction in the elastic task set (with respect to the fixed set) is relatively small. On the other hand, a higher decrease in the QoC is compensated by a larger speed reduction (i.e., energy saving). Also note that, when the system is overloaded, the task set with fixed periods may not be feasible, whereas the elastic application can always be executed with degraded performance.

Note that the QoC index decreases as the task set utilization grows because periods are enlarged. The spike at $U = 0.9$ is due to a speed switch from 0.4545 to 0.8181. Such a speed increase creates a slack, which is used by the algorithm to reduce periods, so increasing the QoC.

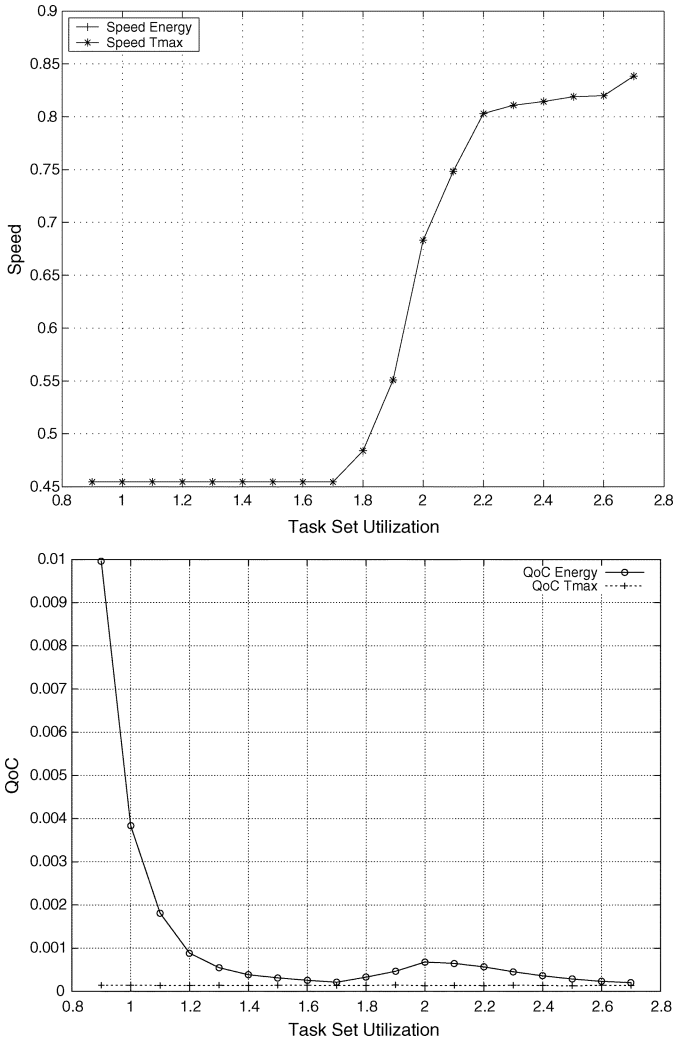


Fig. 7. Comparison of the energy-oriented approach on tasks with elastic and fixed (maximum) periods.

D. Impact of the User-Defined Strategy

Sometimes, the energy-oriented strategy could be too penalizing in terms of performance, and the performance-oriented one could be too energy consuming. In this cases, the processor speed can be manually selected by the user at a level suitable for the application aims. This experiment shows the effect of the user-defined strategy on the task set behavior. The experiment has been performed on a task set with maximum utilization of 0.9 and 1.1. For each utilization, 100 task sets with a random number of tasks in the range [20,100] have been generated. For every task set, the quality of service has been computed at each speed between s_e and s_p. The results are reported in Fig. 9. The experiment shows that a trade-off between performance and energy consumption is achievable by acting on the processor speed. If the selected value is within the computed bounds s_e and s_p, the feasibility of the schedule is always guaranteed. It turns out that the values of s_e computed by (12) result to be the same for the two tested utilizations, while (13) produces two different values for s_p. This happens because the performance-oriented approach is capable of reducing the power

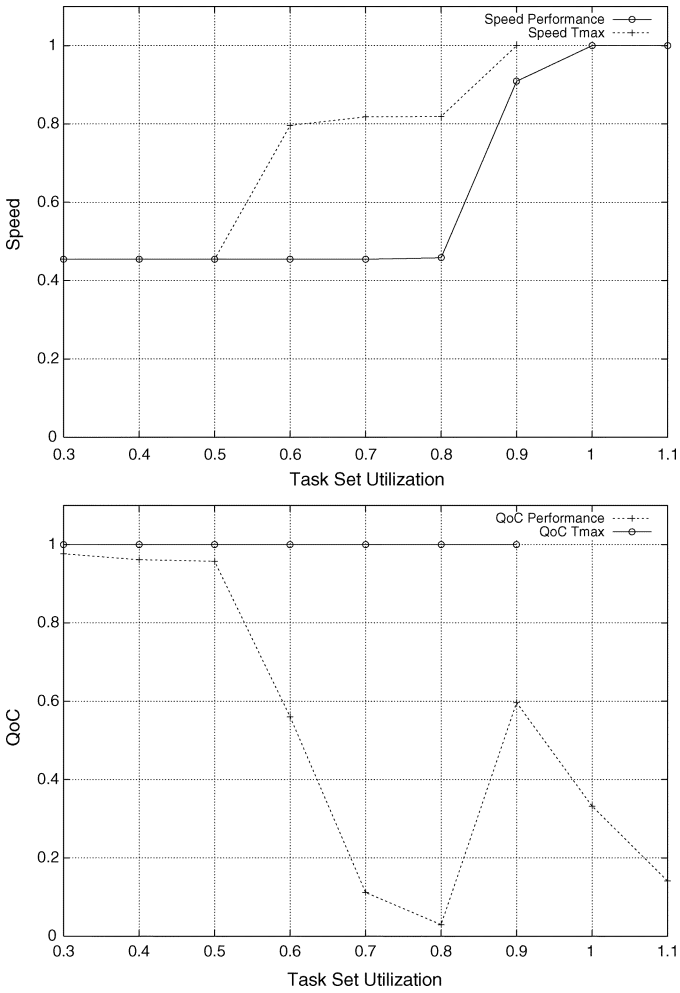


Fig. 8. Comparison of the performance-oriented approach on tasks with elastic and fixed (minimum) periods.

consumption if the task set utilization is smaller than the desired one. It is interesting to observe that, in applications where the task set is static (e.g., in OSEK-compliant systems [1]), the speed selection could be done automatically by a tool according to user-defined parameters, such as the minimum allowed QoC, the maximum available mean power, and so on.

E. Power Consumption and Elastic Coefficients

As mentioned in Section IV-C, assigning the elastic coefficients to express job energy consumption allows the user to privilege tasks with lower power demand. This experiment is aimed at showing how the elastic coefficients may affect the mean power consumption. To do so, the power-related assignment is compared with the one in which all elastic coefficients have the same value.

The power model is the one expressed in (3), where, for the sake of simplicity, K_2, K_1, and K_0 are set to 0, and only K_3 is managed. The maximum utilization of the task set at the maximum speed ($U_{max}(s_{max})$) is 1.1, and 100 task sets (of four tasks) are generated for each configuration. From a configuration to the next one, the value of K_3 is increased by a value equal to the task index ($K_3^i = K_3^i + i$). For each task, a displacement

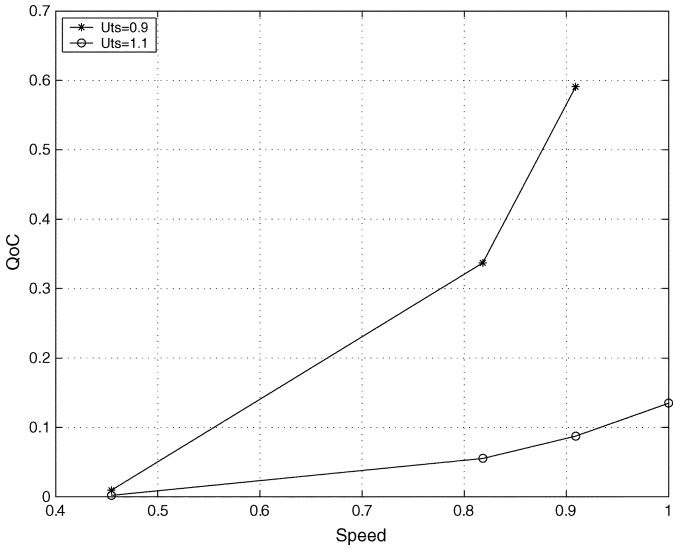


Fig. 9. QoC for the user-defined strategy.

factor is defined as the ratio between K_3^4 and K_3^1. The value of the elastic coefficient E_i for task τ_i is set equal to K_3^i. For each task set, the power consumed by tasks with fixed elastic coefficients and with power-related ones is computed as

$$P = K_3 s^3 \frac{C(s)}{T}.$$

Then, the mean is computed among all task sets for the same configuration and approach. For each coefficient configuration, the comparison between the two assignments is expressed as the ratio between the power-related approach and the fixed-coefficients approach.

Fig. 10(a) illustrates the dependency of the various parameters as a function of the displacement factor, whereas Fig. 10(b) reports the power saving achievable by the power-dependent elastic coefficients. As is clear from the graph, the power-related assignment of the elastic coefficients can produce a significant power saving in the application. As the displacement factor grows, the reduction produced by the power-related approach is more and more significant. In particular, a reduction of 0.8% is observed for a displacement factor equal to 1.2727, which increases up to 12.7%, when the displacement factor is equal to 4. Also note that implementing this features does not increase the runtime overhead of the acceptance test.

F. Online Reclaiming Mechanism

This final experiment shows the effect of the reclaiming mechanism on the power consumption. The energy-oriented strategy was used on a task set with $U_{max}(s_{max}) = 2.0$. To generate early completions, job execution times were set equal to rC_{max}, where r was varied from 0.1 to 1. For each value of r, 100 task sets were generated and the offline and the dynamic approaches were compared in terms of speed and power consumption. For the dynamic algorithm, speed and power consumption were computed as means over the hyperperiod. Results are reported in Fig. 11, which shows the average dynamic speed and the ratio between the mean power

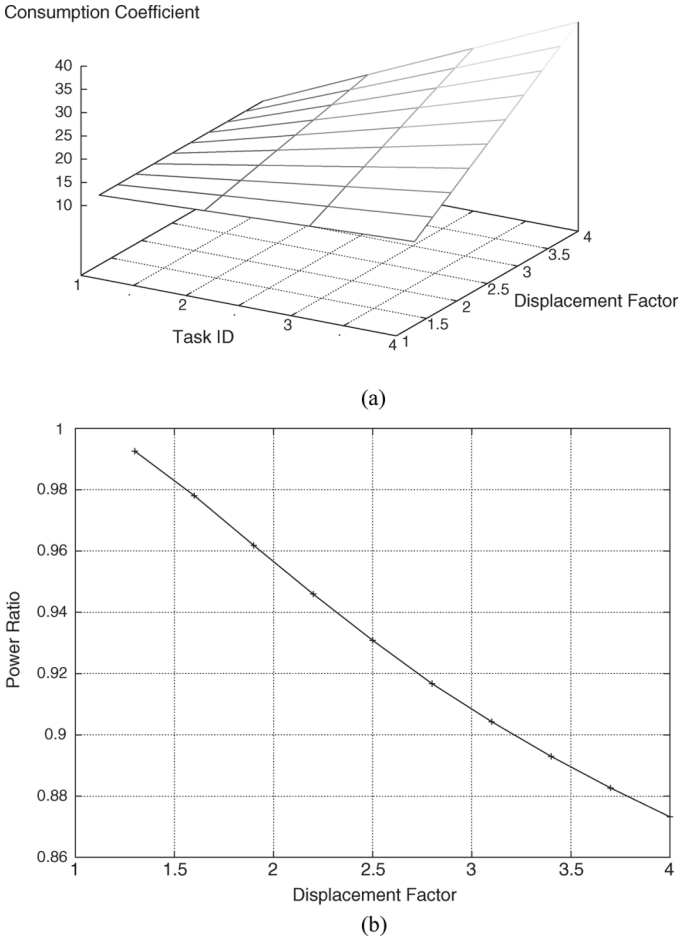


Fig. 10. (a) Task consumption constants. (b) Power consumption ratio.

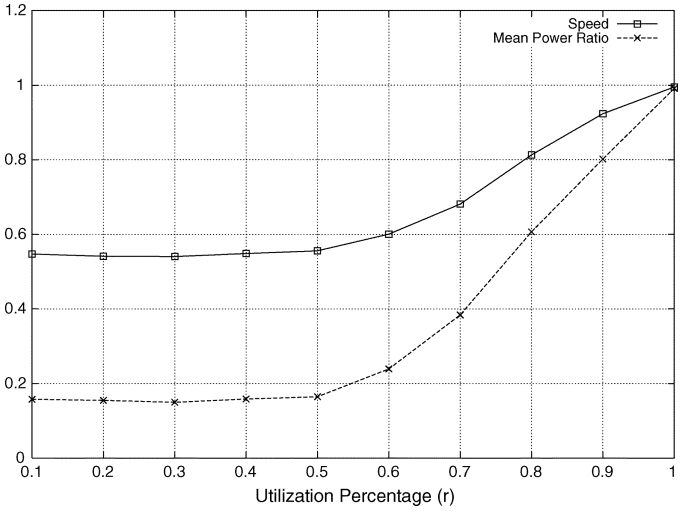


Fig. 11. Speed and power ratio.

in the dynamic approach and the mean power without using the reclaiming algorithm as a function of r.

When $r = 1$, the ratio is almost 1 because tasks run for their worst-case execution time, as supposed in the offline algorithm. The real value of the ratio is less than 1 because sporadic idle times allow the reclaiming algorithm to set the speed to s_{min}. The mean power ratio decreases with r and reaches a stable level for values of r smaller than 0.5. This is explained with the limitation imposed by the value of the minimum discrete speed (0.4545), which does not allow the algorithm to produce the desired reduction.

VII. Conclusions

In this paper, we presented an integrated approach that combines DVS techniques with elastic scheduling to balance control performance with energy consumption in embedded systems running on architectures with a limited number of operating modes. An enhanced task execution time model was used to consider some real architecture characteristics, such as the access to peripherals, whose execution is not scalable with the clock frequency.

Experimental results on an AMD Athlon64 3000+ with four operating modes showed the validity of the proposed execution model and illustrated the advantage of the integrated approach, both for maximizing performance and minimizing energy consumption. The execution model showed that a more precise (less pessimistic) estimation can be achieved for the task computation time, so allowing a better exploitation of the computational resources. We also shown the advantage of the proposed approach in terms of flexibility for the application developer, who can balance energy versus performance, by selecting the system behavior from a high performance configuration (at a cost of high power consumption) to a low-energy setting (corresponding to a degraded performance level). Experiments have also shown that by integrating the power consumption characteristics of individual tasks into the elastic coefficients leads to a larger energy saving, without extra overhead costs. Finally, simulation experiments illustrated the effectiveness of a reclaiming mechanism that takes advantage of early completions to perform a further reduction of the processor speed.

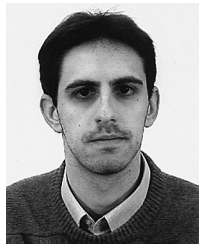
As a future development, we plan to use the proposed approach on a different platform that allows obtaining direct measurements of the real power consumptions, so that a full set of tests can be carried out to precisely compare our method with other related work proposed in the literature.

We also plan to apply the proposed approach to wireless mobile networks, for prolonging battery lifetime in a team of mobile robot units that need to operate under stringent performance constraints.

References

[1] *OSEK/VDX Standard.* [Online]. Available: http://www.osek-vdx.org.
[2] AMD Corporation, PowerNow! Technology, Dec, 2000. [Online]. Available: http://www.amd.com.
[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia Alvarez, "Determining optimal processor speeds for periodic real-time tasks with different power characteristics," in *Proc. 13th Euromicro Conf. Real-Time Systems*, Delft, The Netherlands, Jun. 2001.
[4] ——, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *Proc. 22th IEEE Real-Time Systems Symp.*, London, U.K., Dec. 2001.
[5] ——, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.
[6] E. Bini, G. C. Buttazzo, and G. Lipari, "Speed modulation in energy-aware real-time systems," in *Proc. 17th IEEE Euromicro Conf. Real-Time Systems*, Palma de Mallorca, Balearic Islands, Spain, Jul. 2005.
[7] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, Mar. 2002.

[8] G. C. Buttazzo, M. Velasco, P. Marti, and G. Fohler, "Managing quality-of-control performance under overload conditions," in *Proc. 16th IEEE Euromicro Conf. Real-Time System*, Catania, Italy, Jul. 2004.
[9] P. Gai, L. Abeni, M. Giorgi, and G. C. Buttazzo, "A new kernel approach for modular real-time system development," in *Proc. 13th IEEE Euromicro Conf. Real-Time System*, Delft, The Netherlands, Jun. 2001.
[10] I. Hong, G. Qu, M. Potkonjak, and M. B. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processor," in *Proc. 19th IEEE Real-Time Systems Symp.*, Madrid, Spain, Dec. 1998, pp. 178–187.
[11] Intel Corporation, Intel XScale Technology, Nov. 2001. [Online]. Available: http://www.developer.intel.com/design/intelxscale/.
[12] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage," in *Proc. Int. Symp. Low Power Electronics Design*, Monterey, CA, Aug. 1998, pp. 197–202.
[13] W. Kim, D. Shin, H. S. Yun, J. Kim, and S. L. Min, "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems," in *Proc. 8th IEEE Real-Time Embedded Technology Applications Symp.*, San Jose, CA, Sep. 2002, pp. 219–228.
[14] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. Assoc. Comput. Mach.*, vol. 20, no. 1, pp. 40–61, Jan. 1973.
[15] Z. Lu, Y. Zhang, M. Stan, J. Lach, and K. Skadron, "Procrastinating voltage scheduling with discrete frequency sets," in *Proc. Design Automation Test Europe (DATE)*, Munich, Germany, Mar. 2006.
[16] M. Marinoni and G. C. Buttazzo, "Adaptive DVS management through elastic scheduling," in *Proc. 10th IEEE Conf. Emerging Technologies Factory Automation*, Catania, Italy, Sep. 2005.
[17] T. Martin, "Balancing batteries, power and performance: System issues in CPU speed-setting for mobile computing," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1999.
[18] T. Martin and D. Siewiorek, "Non-ideal battery and main memory effects on CPU speed-setting for low power," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 9, no. 1, pp. 29–34, Jan. 2001.
[19] P. Mejia Alvarez, E. Levner, and D. Mossé, "Adaptive scheduling server for power-aware real-time tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 3, no. 2, pp. 284–306, May 2004.
[20] R. Melhem, N. AbouGhazaleh, H. Aydin, and D. Mossé, "Power management points in power-aware real-time systems," in *Power Aware Computing*, R. Graybill and R. Melhem, Eds. New York: Plenum/Kluwer, 2002.
[21] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 18th ACM Symp. Operating System Principles*, Banff, AB, Canada, Oct. 2001, pp. 89–102.
[22] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *Proc. 24th IEEE Real-Time Systems Symp.*, Cancun, Mexico, Dec. 2003, pp. 52–62.
[23] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg, "Fast: frequency-aware static timing analysis," in *Proc. 24th IEEE Real-Time Systems Symp.*, Cancun, Mexico, Dec. 2003, pp. 40–51.
[24] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control systems," in *Proc. 17th IEEE Real-Time Systems Symp.*, Washington, DC, Dec. 1996, pp. 13–21.
[25] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadrin, "Power-aware qos management in web servers," in *Proc. 24th IEEE Real-Time Systems Symp.*, Cancun, Mexico, Dec. 2003.
[26] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," in *Proc. Conf. Asia South Pacific Design Automation: Electronic Design Solution Fair*, Yokohama, Japan, Jan. 2004, pp. 653–658.
[27] J. Wang, B. Ravindran, and T. Martin, "A power aware best-effort real-time task scheduling algorithm," in *Proc. IEEE Workshop Software Technologies Future Embedded Systems, IEEE Int. Symp. Object-Oriented Real-Time Distributed Computing*, May 2003.
[28] Y. Zhu and F. Mueller, "Feedback edf scheduling of real-time tasks exploiting dynamic voltage scaling," *Real-Time Syst. J.*, vol. 3, no. 1, pp. 33–63, Dec. 2005.



Mauro Marinoni received the degree in computer engineering (with a thesis on real-time control for legged robots) in 2003 from the University of Pavia, Pavia, Italy, where he is currently pursuing the Ph.D. degree in computer engineering.

During the last two years, he has investigated real-time scheduling algorithms and kernel mechanisms with resource and energy constraints and collaborated in a research project involving the development of an inertial platform for autonomous flight. His main research interests include real-time operating systems, scheduling algorithms, energy-aware computing, and mobile robotics.

Giorgio Buttazzo (SM'05) received the degree in electronic engineering from the University of Pisa, Pisa, Italy, in 1985, the master degree in computer science from the University of Pennsylvania, Philadelphia, in 1987, and the Ph.D. degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa in 1991.

He is a Full Professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. From 1987 to 1988, he worked on active perception and real-time control at the G.R.A.S.P. Laboratory of the University of Pennsylvania. His main research interests include real-time operating systems, dynamic scheduling algorithms, quality of service control, multimedia systems, advanced robotics applications, and neural networks. He has authored six books on real-time systems and over 200 papers in the field of real-time systems, robotics, and neural networks.