Feedback Scheduling of Power-Aware Soft Real-Time Tasks

Alberto Soria-Lopez[γ], Pedro Mejia-Alvarez, Julio Cornejo
CINVESTAV-IPN. Sección de Computación, [γ]Departamento de Control Automático
Av. I.P.N. 2508, Zacatenco. México, DF. 07300
soria@ctrl.cinvestav.mx, {pmejia,jcornejo}@cs.cinvestav.mx

Abstract

In this paper we propose an energy-based feedback control scheduling framework for power-aware soft real-time tasks executing in dynamic environments, where real-time parameters are not known a priori.

We propose the use of a controller based on an Energy Savings Ratio, which allows higher energy savings when higher missed deadlines are allowed to ocurr. The scheduler uses the energy feedback to calculate the amount of workload to be adjusted and provides the input for a variable voltage optimization algorithm (VVA). The VVA algorithm is a greedy algorithm that adjusts the workload to optimize power consumption by computing a near optimal solution for the tasks's voltage/speed selection problem.

Extensive sets of tests are executed to simulate the performance of our energy feedback scheduling power-aware architecture under overload and underload conditions. Simulation results show that the proposed architecture is capable of handling real-time tasks with unknown arrivals and execution times, and derive a system in which power savings are maximized.

1. Introduction

Energy management is becoming the *limiting factor* for the functionality of real-time embedded and portable devices because advances in battery technology are progressing slowly whereas computation and communication demands of this devices are increasing rapidly. Energy minimization is critically important for devices such as laptop computers, PCS telephones, PDA's and other mobile and embedded computing systems simply because it leads to extended battery lifetime.

Scheduling policies such as EDF or RMS [6] are capable of handling task sets with sophisticated timing parameters such as, aperiodic task arrivals, precedence constraints, shared resources, or execution on distributed environments. However, it is known that the execution of this scheduling policies is performed using an *open loop* [15]. The term *open loop* refers to the fact that once a task set is scheduled, their parameters (i.e., task's execution time, period or deadlines) can not be adjusted in response to workload variations [15]. Open loop scheduling algorithms provide efficient performance guarantees in predictable environments, where the workload does not experience changes and can be precisely modeled. However, those real-time systems achieve poor performance in unpredictable environments, where the workload experience frequent changes. Another important issue is that open loop schedulers are often designed based on worst-case behavior of real-time tasks. This is an appropriate solution for real-time systems with statically known time properties, but not for battery operated real-time systems where workloads are variable and the system has various unpredictability degrees. The results obtained from this over-provisioning is a system with low performance and low utilization.

In this work, we introduce a feedback scheduling architecture for power-aware real-time tasks. The main parts of this architecture are an *energy-based feedback scheduler* and a *power-aware optimization algorithm*. The feedback scheduler attempts to keep the CPU utilization at high level, to achieve high energy savings, and to distribute the computing resources among real-time tasks to maximize control performance. The energy saving is used as the control variable in the system configuration. The feedback scheduler, based on a P (proportional) controller, calculates the amount of workload to be adjusted and provides the input for the variable voltage optimization algorithm (VVA) [10]. The VVA algorithm is a greedy algorithm that computes a near optimal solution for the voltage (speed) selection problem. The VVA algorithm adjusts the workload by selecting a set of speeds for the execution of each task in the system. The process of selecting the execution speeds while minimizing the energy consumption in the system is the main goal of the VVA algorithm. The feedback scheduling architecture accepts workloads that exhibit a large variability and that execute on a processor capable of handling several (discrete) speeds of execution.

The rest of this paper is organized as follows. In Section 2 related models and related work is reviewed. In Section3,

Proceedings of the Sixth Mexican International Conference on Computer Science (ENC'05)
0-7695-2454-0/05 $20.00 © 2005 IEEE

the system and energy models used are defined. In Section 4, the power-aware scheduling is formulated as an optimization problem. In Section 5 the feedback scheduling power-aware architecture is described and in section 6, the control task model is defined. In Section 7 simulation results are presented to show the performance of our framework. Finally, in Section 8 concluding remarks are presented.

2. Related Work on Variable Voltage Scheduling

In the realm of control techniques used in the scheduling of real-time systems, the related work is divided in the following categories including real-time scheduling[3]: integrated control and multimedia (quality-of service), control and power-aware scheduling, and integrated control and computing applications.

In [13] sampling period selection for a set of control tasks is considered. The desired performance of a task is given as a function of its sampling frequency, and an optimization problem is solved to find the set of optimal task periods. The idea of using feedback scheduling in real-time systems has been proposed in [7], with the introduction of their FC-EDF (Feedback controlled EDF scheduling algorithm). A proportional controller regulates the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their CPU consumption. Several versions of each task are defined (which provide different quality of services), and the problem to solve is to maximize the quality of service provided by the set of tasks. The work presented in [7] has derived many other research works applying control theory in several case studies involving computing applications. The case studies are: Internet web servers [1], proxy cache relative hit ratio [8], microprocessor thermal management and real-time distributed systems [14]. In [5] a QoS framework is proposed for controlling the applications requests for system resources using the amount of allocated resources for feedback.

One of the main motivations of our paper, is that there is very few research work related to power-aware real-time scheduling using control theory techniques. The work in [12, 16] introduces a feedback controlled discrete VVS scheduling algorithm for periodic hard real-time tasks under the EDF scheduling policy. A PID controller is used to control incrementally the systems behavior to achieve its targets, while preserving the hard-real time requirements.

To our knowledge, no previous work has considered the scheduling of power-aware unpredictable real-time worloads where a given number of deadline misses are allowed.

3. Task Model

In our framework we consider periodic preemptive and soft real-time tasks running on one processor. Tasks are independent and have no precedence constraints. The arrival time b_i of task τ_i is unknown. The *life-time* of each task τ_i consists of a fixed number of instances r_i. After the execution of r_i instances, the task leaves the system[1]. We denote by C_i the worst-case number of CPU processor cycles required by τ_i. Under a constant speed V_i (given in cycles per second), the worst-case execution time of τ_i is $\hat{c}_i = \frac{C_i}{V_i}$. The period of task τ_i is P_i which is equal to the relative deadline (D_i) of the task. We assume that the tasks characteristics (e.g., C_i, P_i and D_i) are known at arrival time of the tasks. For C_i, only its worst-case value its known at arrival time, but its value varies at each instance. The real (measured) value of $\hat{c}_i$ will be considered as a_i, considering $0 \leq a_i \leq \hat{c}_i$.

A schedule of periodic tasks is *feasible* (no task misses its deadline), if each task is assigned at least C_i processor cycles before its deadline, at every instance. In our model, an specific number of deadline misses may be allowed to ocurr. The *utilization of a task* is the amount of processor load that the task demands for execution: $U_i = \frac{\hat{c}_i}{P_i}$ (or $\frac{C_i}{V_i P_i}$). We will use the RMS and EDF scheduling policies [6].

We assume that the CPU speed of any task can be changed at *discrete* levels between a minimum speed V_{min} (corresponding to a minimum supply voltage level necessary to keep the system functional) and a maximum speed V_{max}. V_{ij} denotes the execution speed of an instance of task τ_i when executes at speed j, where $V_{min} \leq V_{ij} \leq V_{max}$. The *utilization* of task τ_i when executes at speed j is denoted by $U_{ij} = \frac{C_i}{V_{ij} P_i}$.

The power consumption of the task τ_i is denoted by $g_i(V)$, assumed to be a strictly increasing *convex* function, specifically a polynomial of at least second degree. $g_i(V) \approx (K\ f\ V^2)$, where K is the *output capacitance*, and f is the frequency of the clock (its exact form depends on the technology used). If the task τ_i occupies the processor during the time interval $[t_1, t_2]$, then the *energy* consumed during this interval is $E(t_1, t_2) = \int_{t_1}^{t_2} g_i(V(t))dt$.

For each task τ_i in the system, we define a set of speeds of execution which will be called *class* N_i. The size of N_i depends on the number of discrete voltages that the processor supports. The energy consumption ratio of task τ_i, when executing at speed $j \in N_i$ is computed by

$$E_{ij} = \frac{e_{ij}}{e_{i1}} \tag{1}$$

where e_{ij} denotes the energy consumption of task τ_i executing at speed j, and e_{i1} is the energy consumed by task τ_i executing at its maximum speed.

The energy savings ratio of task τ_i when executing at speed $j \in N_i$, is computed by

$$Es_{ij} = (1 - \frac{e_{ij}}{e_{i1}}) * 100\% \tag{2}$$

1 We assume that some tasks that leave the system may return at a later time.

We assume that all items $j \in N_i$ follow a non-decreasing order. Each task τ_i in the system accrues an accumulated energy consumption $e_i(k)$ upon executing a number of instances during the interval of time between $[(k-1)W, kW]$, where k is defined as the *loop sampling time* and W is defined as the *loop sampling period*. $e(k)$ denotes the amount of energy consumption accrued by all the tasks in the system during $[(k-1)W, kW]$, that is,

$$e(k) = \sum_{i=1}^{n} e_i(k) \tag{3}$$

The maximum energy consumption of task τ_i during sampling time k, denoted as $e_i^{max}(k)$, is achieved when the task executes at its maximum speed, $j = 1$. Each task τ_i in the system accrues an accumulated energy savings ratio $Es_i(k) = (1 - \frac{e_i(k)}{e_i^{max}(k)}) * 100$, during $[(k-1)W, kW]$. $Es(k)$ denotes the amount of energy savings ratio accrued by all the tasks in the system during sampling time k, that is,

$$Es(k) = \left(1 - \frac{e(k)}{e^{max}(k)}\right) * 100 \tag{4}$$

where $e^{max}(k) = \sum_{i=1}^{n} e_i^{max}(k)$ denotes the maximum energy savings accrued by all tasks in the system during sampling time k. Note that all instances of task τ_i will execute at the same speed j during the sampling interval.

Each task τ_i executing at speed j will have a utilization:

$$Ue_{ij} = \frac{\hat{c}_{ij}}{P_i} \tag{5}$$

where $\hat{c}_{ij} = C_i/V_{ij}$ denotes the worst-case execution time of task τ_i when executing at speed j. The worst-case execution time of task τ_i is computed by considering the worst-case number of CPU cycles of the task, C_i. The worst-case utilization of the task set is denoted by $Ue = \sum_{i=1}^{n} Ue_{ij}$. The worst-case utilization for the sampling time k is given by

$$Ue(k) = \sum_{i=1}^{n} \frac{\hat{c}_{ij}}{P_i} \tag{6}$$

The *measured utilization* $Um(k)$ of all tasks over sampling time k is computed by,

$$Um(k) = \frac{\sum_{i=1}^{n} A_{ij}(k)}{W} \tag{7}$$

$A_{ij}(k)$ denotes the real (measured) execution time of task τ_i over the sampling period, while executing at speed j. Note that $A_{ij}(k)$ is the sum of execution times of all instances of task τ_i over the sampling period. The number of instances of task τ_i in the sampling period is $M = \lceil \frac{W}{P_i} \rceil$. For different instances, the *real execution time* of task τ_i, $A_{ij}(k)$ may vary.

The *miss deadline ratio* $Mr(k)$ of all tasks over sampling time k is computed by,

$$Mr(k) = \frac{\text{Number of miss deadlines in sampling time k}}{\text{Total number of instances in sampling time k}} \tag{8}$$

In our implementation, we make the following additional assumptions: 1). *the scheduler will be capable of adjusting the workload* (by changing the speeds of each task) at task arrivals, task departures and at the end of each sampling period W. 2). *the time overhead associated with voltage switching is negligible*.

4. Formulation of the Problem

The problem to be solved in our power-aware real-time framework is to efficiently select voltages/speeds of execution in an unpredictable environment, where a percentage of deadline misses are allowed, such that the energy savings are maximized. This problem is formulated as the following power-aware optimization problem. At each task arrival or departure and the end of each sampling period W, the feedback scheduler must find a set of speeds **x**, for the execution of each task such that the energy savings of the system are maximized.
That is,

$$\textbf{maximize} \qquad Z = \sum_{i=1}^{n} \sum_{j \in N_i} E_{S_{ij}} x_{ij} \tag{9}$$

$$\textbf{subject to} \qquad Ue(k+1) \leq Ue(k) + \Delta U(k) \tag{10}$$

$$\sum_{j \in N_i} x_{ij} = 1, \; i = 1, \ldots, n$$

$$x_{ij} = \begin{cases} 1 & \text{if speed } V_{ij} \; (j \in N_i) \text{ is selected for task } \tau_i \\ 0 & \text{otherwise,} \end{cases}$$

Condition 10 (*workload limit*) indicates that the worst-case utilization $Ue(k+1)$ in the next sampling period $(k+1)$, must be less or equal than the utilization in the actual sampling period $Ue(k) + \Delta U(k)$. If $\Delta U(k)$ (the adjustment on the workload, provided by the controller) is a positive number, it will indicate that additional workload can be accepted in the system, and hence execution speed of the tasks should be decreased. In the other hand, if $\Delta U(k)$ is a negative number, it will indicate that the workload must be reduced, and hence the execution speed of the tasks should be increased. In both cases, our Feedback scheduling power-aware architecture will try to minimize energy consumption.

Since $Ue(k)$ denotes a worst-case utilization, the measured utilization $Um(k)$ will always be $Um(k) \leq Ue(k)$. However, $Um(k) \leq 100\%$ but $Ue(k)$ may be greater than 100 %. Note that if $Ue(k) > 100\%$ it will not necessarily imply that the task set is suffering an overload, because $Ue(k)$ denotes a worst-case value. Note that in our formulation, the energy savings are not linked with the Miss Ratio.

However, the higher value of $Mr(k)$ will allow higher energy savings.

The feedback scheduler is able to adjust the workload only at new task arrivals and departures and at the end of each sampling period. In any case, because of the high variability of the real-time workload it is possible that some overload occurs, during the last sampling period, and that some tasks may miss their deadlines, before the workload is adjusted to meet our optimality criteria.

5. Feedback Scheduling Power-Aware Architecture

In order to meet our optimality criteria, the feedback scheduler adjusts the load of the system by controlling the speed of execution of the tasks. The architecture proposed to solve the problem is illustrated in Figure 1, and described in detail in the following subsections.

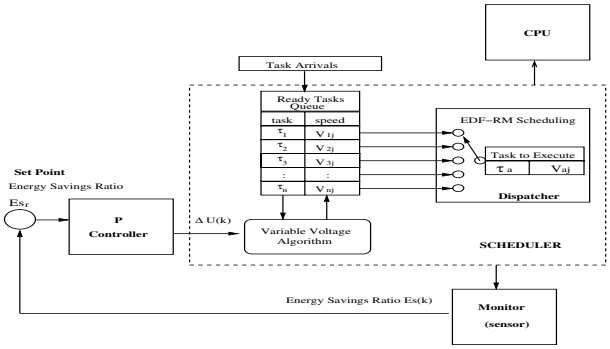

Figure 1: Feedback Scheduling Power-Aware Architecture

5.1. Task Arrivals and Execution

The inputs to our system are real-time tasks that arrive in the system at unknown times. The feedback scheduler contains an acceptance mechanism which decides whether or not the arriving tasks can be accepted in the system. The acceptance mechanism is based on the following condition:

If all tasks in the system are already executing at its maximum speed levels, and the measured utilization $Um(k)$ is at 100%, then no new tasks will be allowed to enter in the system. Otherwise, the arriving tasks will be accepted in the system.

Due to the high variability of the real-time workload, $Um(k)$ will be subject to frequent changes.

Once a task is accepted, it will be sent to the ready task queue of the scheduler and the variable voltage algorithm (VVA) will execute to compute a new set of speeds for each task in the system. While the VVA algorithm is computing the set of speeds (by solving the optimization problem), it will consider the most recently used *workload limit* (condition 10 of the optimization problem), $Ue(k)$.

Finally, the dispatcher will select for execution the tasks with highest priority according to a pre-defined scheduling policy (RMS or EDF[6]).

Once the dispatcher selects the task τ_a for execution, it will execute at the speed V_{aj} selected by the VVA Algorithm. While task τ_a is executing it will change its speed only until the next sampling period. When a task leaves the system, VVA will execute to re-compute the speed of the remaining tasks in the system.

5.2. Feedback Control Loop

The feedback scheduler uses a closed control loop[2, 7]. The control law is evaluated at the end of each sampling period. The control law includes a *monitor* for *energy savings ratio*, a *Proportional controller* and a *feedback real-time scheduler* which includes the VVA algorithm, and a *dispatcher*.

1. Energy Monitor. The energy monitor verifies the energy savings ratio, $Es(k)$ (controlled variable). While tasks are executing, the feedback scheduler will record the values of the controlled variable at each sampling period k.

2. Proportional Controller. The Proportional Controller [2] (PC) compares the control variables against a *set point data* (previously computed) and produces an *error*. This error denotes the difference between the set point data and the controlled variable. At each sampling period, the PC computes the amount of workload that is necessary to adjust in order to reduce the error as much as possible. The workload change computed by the PC in $Ue(k)$ is $Ue(k+1) = Ue(k) + \Delta U(k)$. The PC compensates workload variations (using its control function), while keeping the *controlled variable* as close as possible to the *set point value (performance reference value)*. The PC controller transfers the *workload adjustment* $\Delta U(k)$, to the variable voltage algorithm, for computing a new set of speeds for the tasks in the system. Hence, the output of the controller is $\Delta U(k)$.

The PC controller used in our feedback scheduling framework supports the following control variables:

- **Controlled Variable.** This is the variable controlled by the feedback scheduler to obtain the required performance of the system. The controlled variable is computed at the end of the sampling period. In our case we use the *energy savings ratio* $Es(k)$ as our controlled variable. This controlled variable is defined over the sampling period $[(k-1)W, kW]$.
- **Performance Reference.** The performance reference denotes the performance required for the system to function according to a set point data. The performance reference used in our framework is the *target energy savings ratio*, Es_r. For example, a value of $Es_r = 40\%$ will indicate that the PC must try to

change (reduce or increase) the workload so as to reach the target value provided and to reduce the error to zero. The error in the energy savings ratio, $Es(k)$ is computed by $E_{error} = Es_r - Es(k)$, that is the difference between the target energy savings and the measured energy savings.

- **Manipulated Variable.** This is a system parameter that the controller can change dynamically. The change produced can affect the controlled variable. In our framework, the manipulated variable is the worst-case utilization $Ue(k)$. Usually $Es(k)$ increases when $Ue(k)$ increases. Note that $Ue(k)$ may be higher than 100%.

3. Variable Voltage Algorithm. The dynamic voltage scaling algorithm (VVA) is capable of adjusting the speed of execution of each task in the system, according to the workload adjustment $\Delta U(k)$ provided by the PC controller. The VVA changes in consequence the worst-case utilization $Ue(k)$ on each sampling period. The change in $Ue(k)$ is performed by changing the speeds of execution of the tasks in the system.

As stated in the optimization problem, the objective of the VVA algorithm is to adjust the $Ue(k)$ workload to $Ue(k+1) = Ue(k) + \Delta U(k)$. 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. In order to solve our optimization problem, we propose to use our previously developed Variable Voltage Scheduling Algorithm (VVA) [10] which solves the power-aware optimization procedure in a low computation time. This VVA allows the feedback scheduler to handle power-aware real-time tasks with low cost while maximizing the energy savings of the system. The objective of the VVA is to solve the power-aware optimization problem described by Equations 9 and 10. This optimization problem is formulated as a multiple-choice knapsack problem (MCKP) with binary variables [9]. Its solution is based on the optimization procedure of the PORTS Scheduling Server [10].

The optimization procedure used to solve our problem 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 optimization procedure, proposed in [10] is based on the reduction of the MCKP to the equivalent KP using the convex hull concept [9].

6. Control Task Model

In this section, we will formulate the mathematical model [2] of the power-aware architecture shown in Figure 1. The *controlled system* shown in the dotted rectangle of Figure 1 includes a variable voltage algorithm (VVA) and a real-time scheduler.

Despite the difficulty to model and control non-linear systems, such as the power-aware real-time system proposed in this paper, for design purposes, the system will be considered linear. The mathematical model proposed is shown in Figure 2. Our goal in the modeling is to find the *transfer function* (in the z domain) used by the system to transform the input (change in requested utilization $\Delta U(k)$) to its corresponding output (energy savings ratio, $Es(k)$).

Starting from the control input $\Delta U(k)$, the total utilization $Ue(k)$ represents the integration of the control input. $Ue(k)$ is computed by,

$$Ue(k+1) = Ue(k) + \Delta U(k) \tag{11}$$

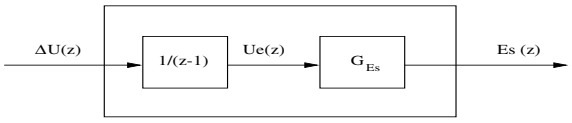

Figure 2: Model of the Controlled System

Since task execution times are unknown and time is variant, the energy savings ratio $Es(k)$ computed will be affected by function $G_S(k)$. This function represents the workload variation and is difficult to model for the following reasons,

- *unpredictability on the execution time of the tasks*: tasks may have different execution paths, because of the structure of the tasks code (e.g. if, else, case, for, while, etc).
- *unpredictability caused by voltage/speed changes on the tasks*: The voltages computed by the VVA algorithm for the execution of the tasks are not known a priori. This causes an additional unpredictability on the execution time of the tasks.
- *workload variation*: tasks arrivals are unknown and execution time between instances may vary. This may cause variations on the estimated utilization $Ue(k)$.

Given that $G_{Es}(k)$ is time-variant, we can use the worst-case value of $G_{Es} = max\{G_{Es}(k)\}$. G_{Es} is defined as the Gain obtained for the worst-case utilization ratio. The value of $G_{Es}(k) = 1.1$ was obtained from experimental data.

The transfer function from $\Delta U(k)$ to $Es(k)$ is given by,

$$\frac{G_S}{z-1} \tag{12}$$

6.1. Energy Savings Control Algorithm

At each sampling period, the controller computes the control signal $\Delta U(k)$ based on the *Energy Savings Error*, $E_{error}(k) = Es_r - Es(k)$.

In this paper we use a proportional controller that is capable of achieving zero steady state error given the system model that includes an integrator. The closed loop transfer function using the controller with gain K_p is,

$$\frac{K_p G_S}{z-(1-K_p G_S)} \tag{13}$$

The controller gain K_p allows to place the only pole $1 - K_p G_S$ for this transfer function. In the z-plane, the stability boundary is the unit circle $|z| = 1$. The system is stable when all poles are located inside the unit circle and unstable when the pole is located outside [7]. We choose a value of $K_p = 0.9091$ that allows a zero overshoot and guarantees the stability of the control loop.

7. Simulation Experiments

The following simulation experiments were designed to test the performance of our Feedback Power-Aware Scheduling Scheduler (FPAS). The feedback scheduler will emulate the execution of a variable speed processor.

A software simulator was developed on a Pentium IV (3 GHz) PC, under the *Linux Red Hat 8.0* Operating System. This simulator allows modifications on the control law, scheduling policies, control parameters or task generation methods, and is comprised of the following four modules:

- *Tasks Generation*: randomly generates tasks with their associated timing parameters (C_i, P_i, D_i, and tasks arrival times b_i).
- *Feedback Real-Time Scheduler*: simulates task executions following a pre-defined scheduling policy. Also, it is capable of monitoring the controlled variables, $Es(k)$, $Um(k)$ and $Mr(k)$ for the P controller.
- *P Controller*: calculates the required utilization change $\Delta U(k)$. The control parameters and controller gains for the FPAS controller are computed according to the values described in Table 1. In Section 6.1 we defined how the control parameters for the FPAS are computed. These controller parameters were computed based on [7].
- *Open Loop Control*: it simulates task executions following a pre-defined scheduling policy. In the Open Loop (OL) controller, tasks are executed at a fixed speed for the complete duration of each simulation, and its resulting $Es(k)$, $Um(k)$ and $Mr(k)$ are measured.
- *Variable Voltage Algorithm*: computes the execution speeds for each tasks. The FPAS Scheduler uses a near-optimal approximation algorithm (VVA) [10] described in Section 5.2.

The results obtained from our simulations are shown in Figures 3 and 4. Each plot in the graphs represents the average of 1000 simulation runs for the FPAS and Open Loop controllers. The total simulation time was 200 seconds. In this simulations we assume a set of speeds of executions as in the Crusoe TM5400 Processor[17], with normalized values in the interval of [0,1]. The normalized values of V_{ij} simulated are: { 0.285, 0.333, 0.380, 0.428, 0.476, 0.523, 0.571, 0.619, 0.666, 0.714, 0.761, 0.809, 0.857, 0.904, 0.952, 1.0 }. Each time a task executes for an interval of time $I = 0.5$ seconds (sampling period), its energy consumption [4] is computed by $E_i = I \cdot V_{ij}^2$.

The tasks timing parameters were computed as follows:

- Tasks time arrivals b_i were computed with an initial set of 5 tasks with $b_i = 0$, followed by 3 sets of 5 additional tasks with task arrivals of $b_i = 20, 40, 60$ seconds respectively. The initial utilization for the total number of tasks $n = 20$ tasks is considered $U_{ini} = \sum U_{i=1}^{20} = 95\%$.
- *Estimated (worst-case) execution time* $\hat{c}_{ij}$: this parameter is computed as $\hat{c}_{ij} = \frac{C_i}{V_{ij}}$, where $\hat{c}_{ij}$ is the execution time of task τ_i and is generated following a uniform distribution in the interval [0.3,0.8] ms.
- *Actual execution time* $A_{ij}(k)$: this parameter is computed following a uniform distribution with values in the interval $[(\hat{c}_{ij} - 0.2\hat{c}_{ij}\ ), (\hat{c}_{ij} + 0.2\hat{c}_{ij})]$.
- *Period* P_i: the period is computed as follows. $u_i = U_{ini}/n$, $\hat{P}_i = \hat{c}_{ij}/u_i$, $P_i = [\hat{P}_i, \hat{P}_i + 35ms]$.

The initial value of the speed of all tasks on each simulation is $j = 8$, $V_{i8} = 0.619$.

	Es	Um	Mr
K_p (RMS)	0.9091	0.185	0.35
K_p (EDF)	0.9091	0.185	0.129
Sampling Period	0.5 sec.	0.5 sec.	0.5 sec.

Table 1. Controller Parameters

The performance of the Feedback Schedulers is measured using the following metrics:

- **Percentage of Energy Savings,** $\%ES$: The percentage of energy savings for our Feedback Power-Aware

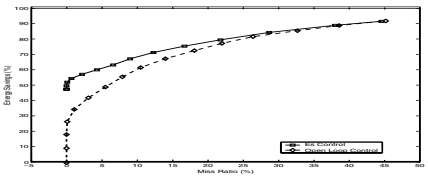

Figure 3: Energy Savings Vs Miss Ratio under RM

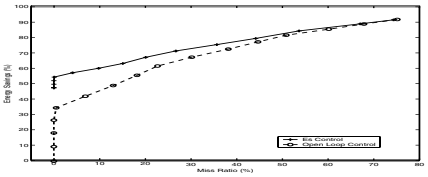

Figure 4: Energy Savings Vs Miss Ratio under EDF

Scheduler (FPAS) and for the Open Loop Control, for all tasks is computed by, $\%ES = \sum_{\forall k} Es(k) * 100$.

- **Miss Deadline Ratio,** Mr: this metric is computed by the ratio of the sum of missed deadlines on all tasks, MD, and the total number of deadlines of all tasks, TMD.

Our main interest in the simulations is to compare the energy savings as a function of the miss ratio for the FPAS and the Open Loop control. Figures 3 and 4 show the results using the RM and EDF scheduling policies respectively on underload and overload conditions. From these Figures it is important to note that, in our simulations, the target references for each type of controller were changed incrementally to reflect the variations on energy savings as a function of the miss ratio.

Target energy savings Es_r for RMS and EDF for the FPAS controller are the following: 0, 0.065, 0.13, 0.195, 0.26, 0.325, 0.39, 0.455, 0.52, 0.585, 0.65, 0.715, 0.78, 0.845, 0.91, 0.975. For the open loop control, the different energy savings were obtained by changing the speeds for all tasks from $V_{ij} = 0.285$ to $V_{ij} = 1.0$.

From Figures 3 and 4 we note that the Es control yields higher (or equal) energy savings than those obtained from the Open Loop Control. That is, for a given Miss Ratio the energy savings obtained by the Es control is higher (or equal) than the energy savings from the open loop control. The highest difference in energy savings of the FPAS controller (compared with that of the Open loop control) is obtained when miss ratio is less than 20 %. For the RMS scheduler the FPAS controller yields from 20% to 0.5% higher $\%Es$ when compared with the Open loop control. Similar results are obtained for the EDF scheduler. Note that when Miss Ratio is equal to 0 (zero) under RM, the system is able to achieve up to 52 % energy savings by the Es control, and 28 % energy savings by the open loop control. For the EDF scheduler, note that the system is able to achieve up to 55.5% energy savings for the Es control, and 36 % for the open loop control.

Under RM, the system yield similar performance when Energy savings are 87 % and Miss Ratio is equal to 35 %. While for EDF, the system yield similar performance when Energy savings are 89.5 % and Miss Ratio is equal to 68.5 %. Note that when Miss Ratio > 0, RM achieve better performance than EDF. For example, for a Miss Ratio equal to 30 % energy savings (Es control under RM) are equal to 84 %, while for EDF energy savings (ES control under EDF) are equal to 63%. From this results it is clear that, since RM misses less deadlines than EDF, energy savings for RM are higher than those achieved by EDF.

The graphs shown in Figures 3 and 4 allow us to estimate the performance of the Es control and the open loop control. According to the Miss Ratio restrictions provided by a given system, a system programmer can use these performance results to estimate the energy savings possible to be achieved by our FPAS algorithm.

With the above results it can be concluded that the FPAS Scheduler improves the performance of the Open Loop control when applied to a power-aware scheduling environment.

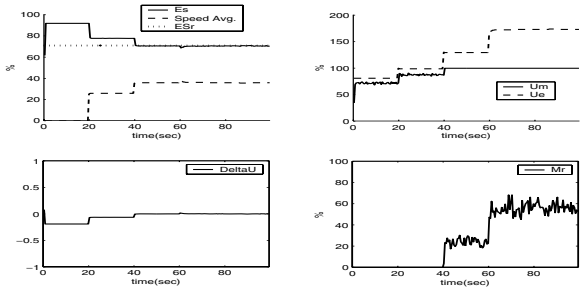

Figure 5: Energy Savings Control Example under RMS

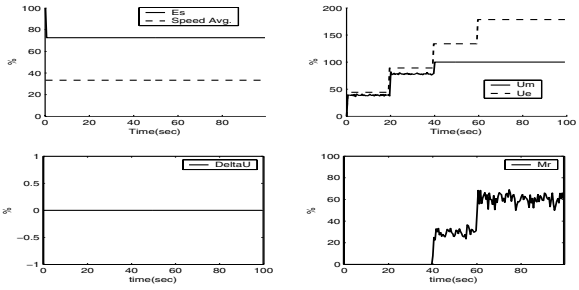

Figure 6: Open Loop Control Example under RMS

7.1. Examples of Control

In this section we provide an example of each of the controllers used in previous experiments.

As discussed before, Figures 3 and 4 show the energy savings obtained as a function of the Miss Ratio under RMS and EDF respectively. These results denote the average of 1000 simulations. In this section, we are interested in showing the behavior of the controls for one specific simulation executing under RMS. Figures 5 and 6 show simulation examples using the RMS scheduling policy for each type of control (energy savings Es and open loop control Ol). The average miss ratios for these examples are 34.24% and 38.21% for FPAS and open loop control respectively. Their corresponding energy savings are 75.208% and 72.643% for FPAS and open loop control respectively. The target references for the Es control is 71% and the constant speed for the Ol control is 0.571.

All Figures are divided in 4 graphs, where the x-axis denotes execution time in seconds. The upper left graph shows energy savings and speed average. The speed average is the average of the speeds of all tasks at a given time instant. The lower left graph shows the behavior of the control variable ΔU. The upper right graph shows the behavior of Um and Ue. The lower right graph shows the behavior of the deadline miss ratio variable.

From Figure 5 it can be noted that the FPAS control adjusts the tasks speeds to the lowest possible speed; this situation is achieved by the EGA algorithm, which tries to maximize energy savings assigning the lowest possible speed, based on $Ue(k)$. This behavior explains the higher energy savings for the FPAS control (Figures 3 and 4). It can be observed that Δ_U is adjusted to compensate for each of the three additional task sets that are loaded at $b_i = 20$, 40 and 60 seconds respectively. After $t = 40$, when the U_e is over 100% the Es control will adjust the speeds continuously in order to maintain the target energy savings reference.

From Figure 6 it can be noted that, now the open loop control will maintain a constant energy savings throughout the entire simulation time. As there are no speed changes during the entire simulation, it is not possible to achieve superior energy savings than those of the optimal EGA algorithm.

8. Conclusions

In this paper, we proposed a closed-loop feedback real-time scheduling architecture, where the workload is dynamically adjusted by a Feedback P (proportional) controller. The feedback scheduling architecture accepts workloads that exhibit a large variability in their workload and execute on a processor capable of handling several (discrete) speeds of execution and under strict limits on available battery power.

The main parts of this architecture are an *energy feedback scheduler* and a *power-aware optimization algorithm*. The feedback energy scheduler attempts to keep the CPU utilization at high level, minimize deadline misses, maximize energy savings and distribute the computing resources among real-time task to maximize control performance.

References

[1] T.F. Abdelzaher, K.G. Shin, N. Bhatti. "Performance Guarantees for Web Server End-Systems: A control Theoretical Approach", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 13, No. 1, Jan. 2002.

[2] Karl J. Astrom and Bjorn Wittenmark, "Computer Controlled Systems: Theory and Design". *Prentice Hall*, 1990.

[3] A. Cervin, J. Eker, B. Bernhardsson, K. Arzen. "Feedback-Feedforward Scheduling of Control Tasks", *Real-Time Systems*, Vol. 23, No. 1-2, July-September 2002.

[4] C. M. Krishna and Y. H. Lee. "Voltage Clock Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems". *IEEE Real-Time Technology and Applications Symposium*, 2000.

[5] B. Li. K. Nahrstedt. "A Control Theoretical Model for Quality of Service Adaptations". In *Proc. of the 6th. International Workshop on Quality of Service*, pp-145-153. 1998.

[6] C.L. Liu, J. Layland. "Scheduling Algorithms for Multiprogramming in Hard Real-Time Environments", *J. ACM*, Vol. 20, No. 1. Jan. 1973.

[7] C. Lu, J. Stankovic, S. Son, G. Tao. "Feedback Control Real-Time Scheduling: Framework, Modeling and Algorithms", *Real-Time Systems*, Vol. 23, No. 1-2, July-September 2002.

[8] Y. Lu, A. Saxena, T.F. Abdelzaher. "Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems". *International Conf. on Dist. Computing Systems*, 2001.

[9] S. Martello and P. Toth. "Knapsack Problems. Algorithms and Computer Implementations". *Wiley*, 1990.

[10] P. Mejia-Alvarez, E. Levner, D. Mossé, "Power Optimized Scheduling Server for Real-Time Tasks". *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2002.

[11] A. Dudani, F. Mueller, Y. Zhu. "Energy-Conserving Feedback EDF Scheduling for Embedded Systems with Real-Time Constraints." *ACM SIGPLAN Joint Conference on Languages, Compilers and Tools for Embedded Systems (LCTES'02),* June 2002.

[12] Y. Zhu, F. Mueller, Y. Zhu. "Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling." *IEEE Real-Time and Embedded Technology and Applications Symposium (RTS'04),* May 2004.

[13] D. Seto, J.P. Lehoczky, L. Sha, K. Shin. "On Task Schedulability in Real-Time Control Systems". *Proc. of the IEEE Real-Time Systems Symposium*, pp-13-21. Dec. 1996.

[14] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son. "Feedback Control Scheduling in Distributed Systems". *Proc. of the IEEE Real-Time Systems Symposium*, Dec. 2001.

[15] J. Stankovic, C. Lu, S.H. Son, G. Tao. "The Case for Feedback Control Real-Time Scheduling". *Proc. of the EuroMicro Conference on Real-Time Systems*, York, UK, June 1999.

[16] A. Varma, B. Ganesh, M. Sen, S.R. Choudhry, L. Srinivasan, B. Jabob. "A Control-Theoretic Approach to Dynamic Voltage Scheduling". *Proc. of the Compilers, Architectures and Sythesis for Embedded Systems, San Jose CA.*, 2003.

[17] www.transmeta.com