Feedback Dynamic Voltage Scaling DVS-EDF Scheduling: Correctness and PID-Feedback *

Yifan Zhu and Frank Mueller

Department of Computer Science/Center for Embedded Systems Research
North Carolina State University,Raleigh, NC 27695-7534

mueller@cs.ncsu.edu, phone: +1.919.515.7889, fax: +1.919.515.7925

Abstract

Dynamic voltage scaling (DVS) is a promising method for embedded systems to exploit multiple voltage and frequency levels and, subsequently, prolong battery life. In this paper, we present a novel approach using DVS technology as well as feedback control schemes on real-time embedded systems to achieve low energy consumption. Our method relies strictly on operating system support by integrating a DVS scheduler and a feedback controller on the base of the EDF scheduling algorithm. Contributions of this work include (1) techniques for preemption handling, (2) a simple feedback scheme as well as (3) a PID feedback scheme, (4) a proof of correctness and (5) an experimental evaluation. Experiments demonstrate the viability of our schemes for different system workloads with varying execution times. Simplistic proportional feedback outperforms prior work on Look-ahead EDF by up to 37% and is shown to work well for task sets with constant execution times. For task sets with predictably fluctuating execution times, the simple feedback scheme is enhanced by a PID controller to adapt to dynamically changing system workloads. Our DVS scheme is demonstrated to benefit significantly from PID-controlled feedback by achieving up to 14% additional energy savings over those without the PID feedback scheme. The resulting energy consumption from the PID feedback scheme match the observed power levels in constant execution time cases even when the actual execution times diverge as much as ±40% from their WCET between different jobs of the same task. To the best of our knowledge, our feedback DVS-EDF scheduling scheme outperforms previously published approaches in its ability and adaptivity to conserve energy, and it is proved to preserve the feasibility of a schedule.

1. Introduction

Energy consumption is a major concern for real-time embedded systems due to their limited battery lifetime. Energy saving techniques usually prolong the battery life but, at the same, time impair satisfaction of the timing constraints of the real-time applications. Recent trends in embedded hardware support multiple voltage and clock frequency settings at the processor level. Because power dissipation of a CMOS circuit is proportional to the clock frequency and to the square of the voltage [2], DVS technology is used to dynamically scale the voltage and frequency of a processor to reduce energy consumption and achieve an optimal energy management for embedded systems. The objective of this work is to exploit maximum energy saving through a novel dynamic voltage scaling (DVS) technology combined with different feedback schemes.

Traditionally, schedulability theory for real-time systems relies on *a priori* knowledge of the worst-case execution time (WCET) of hard real-time tasks to guarantee the feasibility of a schedule. However, experiments show a wide variation between longest and shortest execution times for many embedded applications. In [16], execution times of real-world embedded tasks vary by as much as 87% relative to their measured WCET. Budgeting for the WCET may result in excessive energy consumption even through actual utilization levels are low compared to the worst case. DVS can remedy this problem by lowering frequencies while still meeting deadlines in hard real-time schedules. Prior DVS techniques with real-time scheduling schemes have been demonstrated to obtain significant energy savings for time-constrained embedded systems [15, 12, 1, 4, 1, 8].

In this work, WCET may be derived by means of static timing analysis or experimental assessment while actual execution time is measured during task execution. Our DVS schemes capitalize the potential of savings due to the severe differences between WCET and actual execution times. However, pure DVS techniques do not perform well for dynamic systems where the system workloads are hard to model accurately. Feedback control techniques have been exploited by mapping control theory methodology to real-time scheduling in prior work to address that problem [10, 11, 14]. Our work goes beyond the techniques in these previous studies by combining feedback control schemes into the DVS scheduler for real-time embedded systems. Both simple feedback schemes and PID feedback schemes are exploited to make the system adaptive to varying workloads as well as most realistic real-time application environments. Our experiments demonstrate that the resulting energy savings exceed those of the previously published work.

2. Energy-Aware EDF Scheduling

Energy-awareness functionality can be integrated into real-time embedded systems with different kinds of real-time scheduling policies. We focus on earliest deadline first (EDF) scheduling [9], a dynamic scheduling paradigm, to incorporate energy awareness into the scheduling decision. Dynamic scheduling is usually more flexible and adaptable to systems where the workload varies significantly, which is often the case in real-time embedded systems with tasks' actual execution times being much less than their measured WCET. The following periodic task model is used in this work to describe various attributes of the workload:

- T_i: The i^{th} task in a task set T ($1 \le i \le n$). Each task T_i is a sequence of jobs.
- P_i: The period of T_i.
- C_i: The maximum computational budget (worst-case execution time) of T_i.

*Supported in part by NSF grant CCR-0208581.

- c_{ij}: Actual execution time for the j^{th} job of task T_i.

We assume relative deadlines of tasks equal to their period. By always scheduling the task with the earliest deadline first in a preemptive fashion, a schedule is feasible (*i.e.*, none of its deadlines are missed) if and only if the following necessary and sufficient condition holds: $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$. Based on this result, Pillai and Shin [12] extended this result to EDF with DFS modulation. The following schedulability test was exploited in [12]:

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq \alpha \qquad (1)$$

where $\alpha = \frac{f_i}{f_m}$ denotes the scaling factor representing the fraction of the current processor frequency f_i over the maximum frequency f_m. An optimal α value can be obtained from Equation 1 when $c_{ij}{=}C_i$. One of the limiting factors of this work was that it only implements an inter-task DVS scheme where an entire task was scaled to execute at a fixed frequency. Since a task's actual execution time c_{ij} is unknown in advance, the optimal α value is also difficult to obtain. The approach proposed in [12] also lacks adaptability to dynamically changing workloads. Our work proposed a more aggressive approach to reduce energy consumption as early as possible as detailed in the next section.

3. DVS-EDF Scheduling

In the following, we develop a framework for greedily scheduling real-time tasks under DFS/DVS and guaranteeing deadlines. One objective of our approach is to aggressively and optimistically reduce energy consumption as early as possible. A second objective is to exploit the existing invocations of the scheduler during execution to perform frequency scaling so that DFS/DVS essentially comes for free (other than the calculation of frequency levels). The third objective is to utilize feedback mechanisms for selecting appropriate scaling levels in order to reduce the energy consumption for task completion at a certain scaling level. We assume a set of periodic tasks with known deadlines equal to their periods. Our approach derives actual execution savings from a statically compiled worst-case schedule for the hyperperiod. The worst-case schedule is dynamically compared to actual executions to determine the lowest possible frequency scaling levels without missing deadlines.

An optimal scheduling can be obtained from Equation 1, but there is no way to know in advance the optimal value of the scaling factor α in an unpredictable environment where each task's actual execution time is unknown until the task completes. The worst-case execution time of each task is generally less than its actual execution time. Instead of scaling at a uniform speed for all tasks, only the frequency of the current task (the task with the earliest deadline under EDF) is scaled in our scheme. All remaining tasks execute at the maximum frequency f_m with a scaling factor of 1. Hence,

$$\alpha^{-1}\frac{C_k}{P_k} + \sum_{i\in\{1,\ldots,n\}\setminus\{k\}} \frac{C_i}{P_i} \leq 1 \qquad (2)$$

The motivation of scaling only the current task is that a greedy scheme usually yields a near-optimal result when optimal solutions are infeasible due to their computational overhead. The current task is always scaled as much as possible to achieve a local minimum in energy consumption. Since the actual execution time of a task is typically smaller than its WCET C_i, the current task can still finish early and produces slack for next tasks to scale frequencies again.

In order to obtain a scaling factor α as small as possible, we split each task T_k into two subtasks T_A and T_B. The corresponding execution time is also split into two parts:

$$C_k = C_A + C_B \qquad (3)$$

While greedy scaling is a DVS scheme on the inter-task level, task-splitting is a DVS scheme on the intra-task level, which is expected to result in an even lower energy consumption. In the ideal case, an energy-optimal schedule is obtained when each task can be split into infinitely many small pieces and when voltage scaling occurs continuously during execution. However, this is unrealistic since both frequency and voltage changes incur overhead to system energy consumption. Excess splitting increases energy consumption to where its overhead offsets any savings. Furthermore, since energy consumption is proportional to frequencies and to the square of the voltage, the lowest energy consumption occurs upon a uniform frequency. This uniform frequency can be derived as the minimum of a second-order polynomial (a third-order polynomial when considering voltage as a function of frequency) upon perfect knowledge of actual execution time. Since actual execution times are generally not known, we restrict the number of frequency and voltage changes for each task to be no more than two. While T_B always executes at the maximum frequency level f_m, T_A is able to execute at a lower frequency level than it could without task splitting. Our scheme aims at finishing actual execution within T_A while reserving enough time in T_B to meet the deadline if the WCET is exhibited in full. With this scheme, we can safely scale the frequency within T_A using available slack while T_B executes at maximum frequency following a last-chance approach [3]. Let s_k be the slack passed to T_k when T_k is released, then [5]:

$$\alpha = \frac{C_A}{C_A + s_k} \qquad (4)$$

Other methods, such as idle time utilization and slack passing, are also used in our scheme to achieve a low energy consumption as detailed in [5]. Both schemes, only briefly outlined here, are based on a comparison between the *actual* schedule and the worst-case or *maximal* schedule, the latter being constructed statically ahead of any execution (or at admission time). The key to idle utilization is to create an idle task, which increases utilization under WCET to 100% while its actual execution is always zero. Equation 2 tries to exploit all available idle slack from a task's activating time to its deadline and uses it to scale the task at a low speed corresponding to a low frequency and voltage level in the processor. By choosing the minimum period of all tasks for the idle task, slack in the maximal schedule becomes available early for scaling other tasks with non-zero execution budgets. Slack passing is another technique that exploits early completion of a task by passing the unused remianing time to next task. Based on the maximal schedule, eligible portions for slack passing can be determined while preserving the feasible of a schedule, and are augmented by any idle slots in the interval between release time and deadline. The slack for the current task can be computed by the following equation:

$$S = S + idle(d1...d2) \qquad (idle\ slots) \qquad (5)$$

where S is the slack passed from the previous task and idle(d1..d2) is the amount of idle between the previous and the current task's deadline. Next, we demonstrate that preemption handling requires special handling, which was recently outlined in a prior workshop paper [17].

3.1 Preemption Handling

When preemption occurs, the preempted task will relinquish its remaining slack and pass it on to the next task, just as it does when a task completes. It follows a greedy scheme in that we try to pass as much slack as possible to scale running tasks and speculate on early completion to aggregate more slack. But there are two differences here. First, the preempted task itself cannot generate any slack based on its own execution at preemption points since task's completion time is unknown. Hence, no additional slack is added to its inherited total slack. Second, the preempted task still needs some time to complete its execution in the future. The remaining execution time must be reserved in advance to avoid future deadline misses caused by over-exploiting slack from other tasks. At the preemption point, the expected remaining execution time $left_{ij}$ of the preempted task is:

$$left_{ij} = C_i - c_{ij} \times \alpha\prime \qquad (6)$$

Our slack passing scheme promises that the preempted task will not miss its deadline by reserving corresponding slack:

$$S = S - left_{ij} \qquad (\textit{future slots}) \qquad (7)$$

The old slack is derived from Equation 5 and the resulting slack S can be passed to the next task.

Future slot allocation in this manner is essential to ensure the feasibility of the schedule under DVS. Future slots will be allocated only if the worst-case schedule does not include sufficient slots for the preempted task's job between the preemption point and its deadline. We devised multiple schemes for reserving these slots.

- Forward sweep: When a task $T1$ is preempted and requires $left_{1j}$ future slots, the preempting task $T2$ deducts this amount from its available slack S. If $left_{1j} > S$, then $T2$ remains without slack. If another task $T3$ is initiated, the calculation repeats itself.
- Backward sweep: Future slots of $T1$ are allocated in idle slots within the worst-case schedule from its deadline $d1$ backwards. Any of these idle slots become unavailable for slack generation, *i.e.*, these slots are excluded in Equation 5.

An example is depicted in Figure 1. The upper time line of idle slots presents a excerpt of the worst-case schedule that depicts idle task allocations, only. The lower time line shows the dynamic schedule of tasks. Upon release of $T2$ at t2, $T1$ is preempted. Let us assume that $T1$ does not have sufficient static slots (three slots) beyond t2 to finish its execution. Hence, it has to rely on future idle slots. During $T2$'s execution, $T3$ is released. Both $T2$ and $T3$ have smaller deadlines than $T1$ ($d2 < d3 < d1$). Subsequently, $T1$ only resumes some time after $T3$ completes.

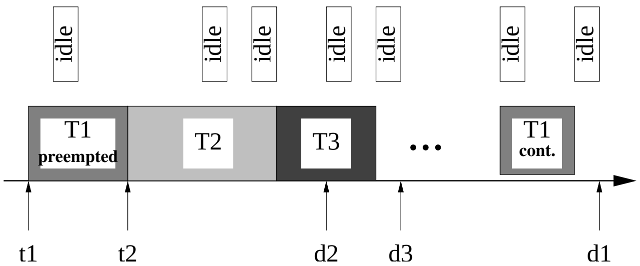


Fig. 1: Future Slot Reservation

Future slot allocation of $T1$ then depends on the chosen scheme. The forward sweep results in zero idle slack for $T2$ and $T3$ since idle slots during the tasks' periods are not sufficient to cover $T1$'s future needs of three slots at the respective invocation times. The backward sweep, on the other hand, reserves the last 3 idle slots (from d1 backwards), such that $T2$ and $T3$ may consume at least two and one idle slots for scaling, respectively, even if they use up their time quantum in full.

Overall, the forward sweep is not as greedy as the backward sweep in the sense that earlier tasks may not be scaled due to $T1$'s future slots. A forward sweep is likely to result in zero slack for the preempting task $T2$ if $P2 << P1$, *i.e.*, if its period is much shorter. There are simply fewer idle slots available, which may not suffice to cover $T1$'s future requirements. More idle slots past $d2$ will be required in this case. The backward sweep always results in the most greedy approach in delaying the needs of $T1$ as long as possible. This is consistent with the observation that early completion is likely to generate slack for each task, a property inherent to our algorithm. In the following, we focus on another key issue, namely on how to choose a suitable value for $C_A + C_B$ ($0 \leq C_A \leq C_k$) to get a small scaling factor α. Since C_A is based on the estimated worst-case execution time, the basic objective is to let C_A approximate T_k's actual execution time c_{ij} so that the actual execution of T_k can be completed at the low frequency level corresponding to α. In the following sections several feedback schemes are described to determine the value of C_A in our DVS scheme.

4. Proportional Feedback

Feedback control is one of the fundamental mechanism for dynamic systems to achieve equilibrium. In a feedback system, some variables, *i.e.*, controlled variables, are monitored and measured by the feedback controller and compared to their desired values, so-called set points. The differences (errors) between the controlled variables and the set points are fed back to the controller for further actions. Corresponding system states are usually adjusted according to the differences to let the system variables approximate the set points as closely as possible.

In the previous section, we stated that the effectiveness of our DVS-EDF scheme depends on the correctness of the estimation for C_A, the execution time of the first part of a task after task splitting. If C_A can always be close enough to the actual execution time of the next job of the task c_{ij}, there is no need for the task to enter the second subtask portion and switch to the maximum system frequency. Hence, the entire task can execute at a low system frequency such that a near-optimal energy consumption can be obtained. Since a task's actual execution time tends to vary, a closed-loop feedback control becomes a natural choice to determine the value for C_A in our scheme.

According to the above objective, we choose the value of C_A as the controlled variable while c_{ij} is chosen as the set point. First, a simple feedback control scheme is used in our DVS-EDF algorithm. C_A is chosen as 50% of the WCET for the first instance of each task. While half of the task's execution is budgeted at a low frequency, half of it is reserved at the maximum frequency. The task can still meet its deadline, even if the worst case is exhibited. Initially, the energy consumption may be significant and is likely to differ from the optimal case due to inappropriate estimations of the actual execution time. Over time, we replace C_A with the actual execution time of the task based on the execution time fed back after each task completion.

The average value of execution times over past executions is utilized to anticipate future C_A portions. On the average, this scheme allows us to complete the entire task's budget at a low frequency level, which closely approximates the optimal energy-saving schedule. Let C_{Aj} be the anticipated C_A value for the j^{th} job of a task. We define the following equations to get the anticipated C_A value for job $j+1$:

$$\begin{aligned} &C_{A1} = 0.5 \times WCET \\ &C_{A(j+1)} = \frac{C_{Aj} \times (j-1) + c_{ij}}{j}, j \geq 1 \end{aligned} \tag{8}$$

Here, c_{ij} is the actual execution time of the j^{th} job of task T_i. Each time a job completes execution, its actual execution time is fed back and aggregated to anticipate the next job's actual execution time, which is further used to calculate an ideal scaling factor for that task.

Although such a simple feedback scheme only considers a pure gain adjustment over the anticipated C_A value, experiments show that this scheme works quite well for real-time task sets where each task either has a constant actual execution time or it has an execution time varying within a small bounded range. For task sets with highly fluctuating execution times, a more sophisticated feedback scheme is required, which is detailed in the next section.

5. PID Feedback Controller

The simple feedback scheme described in the previous section follows a proportional adjustment of frequency levels relative to the average execution time. In practice, real-time applications, such as media decoder or image processing systems, often experience fluctuating execution times over a period of time. The fluctuations may result in tendencies leading to higher processing demands up to some point and receding demands after this peak point. In order to devise a DVS-EDF algorithm adaptive to such a dynamic environment, more sophisticated feedback schemes are needed.

Past work in dynamic real-time scheduling has demonstrated that adaptive techniques derived from control theory can enhance a schedule by reacting to tendencies in execution time fluctuations [10]. The adaptive techniques generally rely on a PID feedback controller, which is widely used in engineering areas. A PID controller consists of three different elements, namely, proportional control, integral control, and derivative control. Proportional control influences the speed of the system adapting to errors, which is defined as the difference between the controlled variable and the set point, by a pure proportional gain item. Integral control is used to adjust the accuracy of the system through the introduction of an integrator on past error histories. Derivative control usually increases the stability of the system through the introduction of a derivative of the errors. The PID feedback controller can be described in three major forms: the ideal form, the discrete form and the parallel form. Although the discrete form is often used in digital algorithms to keep tuning similar to electronic controllers, the parallel form is the simplest one, and the integral and derivative actions are independent of the proportional gain in the parallel form. We choose the following parallel form as the base of our PID feedback implementation:

$$output = K_p * \epsilon(t) + \frac{1}{I}\int \epsilon(t)\,dt + D\frac{d\epsilon(t)}{dt} \tag{9}$$

where K_p, I and D are the proportional, integral and derivative coefficients, respectively, and $\epsilon(t)$ is the system error.

We integrated the above PID controller into our DVS-EDF feedback scheme to control the scaling level in response to recent tendencies in fluctuations. As in the simple feedback scheme, we still choose C_A as the controlled variable and c_{ij} as the set point. The difference between the controlled variable and the set point, *i.e.*, the error, is measured periodically by the PID controller. Its output is fed back to the DVS-EDF scheduler to adjust the value for C_A. Let C_{Aij} be the estimated C_A value for the j^{th} job of a task T_i. The following discrete PID control formula is used in our DVS-EDF scheduler:

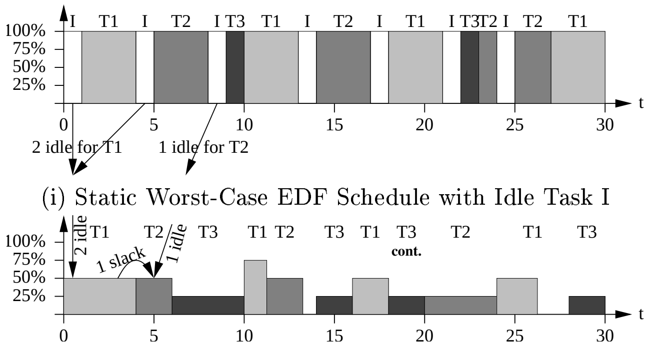
$$\begin{aligned} &\Delta C_{Aij} = K_p * \epsilon(t) + \frac{1}{I}\sum_{IW} \epsilon(t) + D\frac{\epsilon(t)-\epsilon(t-DW)}{DW} \\ &C_{Ai(j+1)} = C_{Aij} + \Delta C_{Aij} \end{aligned} \tag{10}$$

where K_p, I and D are proportional, integral, and derivative coefficients, respectively, and $\epsilon(t) = c_{ij} - C_{Aij}$ is the monitored error. The output ΔC_{Aij} is fed back to the system and is used to regulate the next anticipated value for C_A. IW and DW are tunable parameters such that only the errors from the last IW (and DW) task jobs will be considered in the integral (and derivative) term. We use $DW = 1$ to limit the history, which ensures that multiple feedback corrections do not affect one another. Details about tuning these coefficients and results are presented in the experimental section to demonstrate the benefits of PID-controlled feedback for frequency scaling.

6. Example

Combinning all the techniques illustrated above, we now turn to a description of the entire algorithm. Our algorithm starts with an offline construction of the static worst-case EDF Schedule within the interval of the hyper-period. Figure 2(i) shows an example of such a static worst-case EDF schedule. All scheduling events (including task release, preemption, resumption, and completion) of the worst-case EDF schedule are stored in a look-up table to reduce time complexity.

Next, the original task set is scheduled according to our algorithm (without the idle task), just as for regular EDF scheduling. Additional operations to calculate slack and to set the CPU frequency/voltage are inserted at scheduling points. As shown in Figure 2(ii), when the first task T (with the earliest deadline) is activated at time 0, its initial slack is assigned according to Equation 5: $S = S + idle(d1..d2) = 0 + idle(0..d_T)$. The initial S and d1 is set to 0 since no previous task has been scheduled. The value of $idle(0..d_T)$ is obtained from the pre-calculated worst-case EDF schedule. Then, a frequency scaling factor $alpha$ is set according to Equation 4: $alpha = C_A/(C_A + S)$ (with C_A = 50% WCET, initially). The CPU frequency is set to $alpha * f_m$. When the first task completes, unused slack is adjusted and passed on to the next task according to Equations 6 and 7. The estimated



(i) Static Worst-Case EDF Schedule with Idle Task I

(ii) Our Feedback DVS at Beginning of 1st Hyperperiod

Fig. 2: Discrete Scaling Levels for 3 Tasks

Procedure Initialization
for each $T_k \in \{T_1, T_2, \ldots, T_n\}$ **do**
$C_{Ak} \leftarrow C_k/2$
$left_{k0} \leftarrow C_k$
$t_i \leftarrow 0$
$U \leftarrow \frac{C_1}{P_1} + \frac{C_2}{P_2} + \ldots + \frac{C_n}{P_n}$
$P_{n+1} \leftarrow P_1$
$C_{n+1} \leftarrow P_1 \times (1 - U)$
$c_{n+1} \leftarrow 0\}$
$slack \leftarrow 0$

Procedure TaskActivation($\mathbf{T_{ij}}$)
if processor was idle for d **then**
$slack \leftarrow slack - d$
if T_{pk} preempted/interrupted **then**
$left_{pk} = C_p - c_{pk} \times \alpha\prime$
$slack \leftarrow slack - idle(d_{ij}..d_{pk})$
if $left_{pk} > slots(T_{pk}, now..d_{pk})$ **then**
$reserve_{pk} \leftarrow left_{pk} - slots(T_{pk}, now..d_{pk})$
allocate $reserve_{pk}$ in
$idle(now..d_{pk}) + completed(now..d_{pk})$
$slack \leftarrow slack - reserve_{pk}$
else (T_{pk} *completed execution*)
if $now > d_{pk}$ **then**
$slack \leftarrow slack - idle(d_{pk}, now)$
$slack \leftarrow slack + idle(d_{pk}..dij)$
$\alpha\prime \leftarrow \min\{\frac{f_1}{f_m}, \ldots, \frac{f_m}{f_m} | \frac{f_i}{f_m} \geq \frac{C_{Aij}}{C_{Aij}+slack}\}$
if ($\alpha\prime = 1$) **then**
$C_A \leftarrow 0$
else
$C_A \leftarrow slack \times \alpha\prime/(1 - \alpha\prime)$
SetInterrupt($T_i, C_A/\alpha\prime$)
SetFrequency($\alpha\prime$)

Procedure TaskCompletion($\mathbf{T_{ij}}$)
$slack \leftarrow slack - c_{ij} + C_i$
$\epsilon \leftarrow c_{ij} - C_{Aij}$
$\Delta C_{Aij} \leftarrow K_p * \epsilon(t_i) + \frac{1}{I} \sum_{IW} \epsilon(t_i) + D\frac{\epsilon(t_i) - \epsilon(t_i - DW)}{DW}$
$C_{Ai(j+1)} = C_{Aij} + \Delta C_{Aij}$
$t_i \leftarrow t_i + 1$
$left_{i(j+1)} = C_i$
if $reserve_{ij} > 0$ **then**
release $idle(now..d_{ij})+$
$completed(now..d_{ij})$ up to $|reserve_{ij}|$
Procedure SetInterrupt($\mathbf{T_{ij}, C_A}$)
Set timer interrupt for T_{ij},
triggered C_A time units later
Procedure SetFrequency($\alpha\prime$)
$f \leftarrow \alpha\prime \times f_m$

Fig. 3: Pseudocode of Feedback DVS Scheme

value of C_a for the first task is also updated according to our feedback scheme. When the second task is scheduled, its slack is again determined by Equation 5, this time with a non-zero S on the right hand side of the equation if the first task ever passes any unused slack to it. Its frequency setting level is determined in a similar way as the first task. For later task instances, the feedback scheme chooses a C_A as close as possible to the tasks' actual execution times. Hence, the entire task is scaled at a low frequency level. Preemption handling, as described in Section 3.1, is also applied but not shown here to simplify the example.

An algorithmic description of our DVS-EDF scheme integrated with the PID feedback control is given in Figure 3. This algorithm is a refinement of our previous work [5] and reflects the PID feedback scheme and preemption handling with future slot reservation. We use the following notation:

- T_{ij}: the j-th job of task T_i
- ij, pk : indices for the current and previous tasks relative to T_{ij}
- now: the current time
- r_{ij}: the release time of T_{ij}
- d_{ij}: the deadline of T_{ij}
- C_i: the WCET of T_i (without scaling)
- c_{ij}: the actual execution time of T_{ij} up to now (with scaling)
- $left_{ij}$: the remaining WCET of T_{ij} (without scaling)
- $slack$: system current slack
- $idle(t1..t2)$: the amount of idle slots between times [t1,t2]
- $completed(t1..t2)$: slots of already completed tasks between times [t1,t2]
- $slots(T_{ij}, t1..t2)$: the amount of time slots reserved for T_{ij} in the worst case between times [t1,t2]

The effect of the PID feedback scheme is shown in the following example. Consider a task set of three tasks T1={12,32}, T2={12,40} and T3={4,65} where $T_i = \{C_i, P_i\}$ denotes task T_i's worst case execution time C_i and its period P_i. The actual execution times of different jobs of a task fluctuate according to a sine curve with a minimal value of 50% WCET and a maximal value of 100% WCET. Figure 4(a) is a snapshot of the DVS-EDF schedule for this task set with a simple feedback scheme. Figure 4(b) depicts the DVS-EDF schedule for the same task set using feedback with PID parameters CP=0.9, CI=0.08 and CI=0.1.

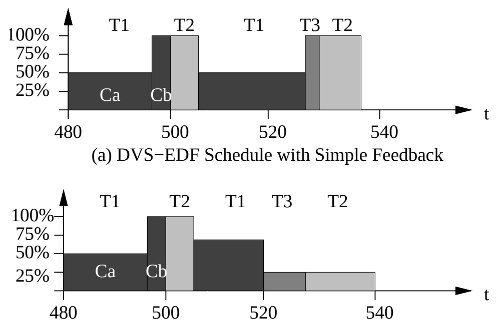
We can see from the figures that the first job of T_3 and the second job of T_2 are scheduled to run at a much lower frequency level in the PID feedback schedule than in the simple feedback schedule. The first job of T_3 with an actual execution time of 2.57 starts at time 524 in the simple feedback schedule and at time 520 in the PID feedback schedule. The simple feedback scheme computes an average execution time C_{avg} of 3.09 for T_3 according to Equation 8, but the PID feedback scheme determines an average execution time of 3.06 for T_3 according to Equation 5, which is a closer approximation to T_3's actual execution time. With the closer approximation of C_{avg}, the PID scheduler is able to scale the task more aggressively than the simple feedback scheduler. Similarly, the simple feedback scheme can only compute an average execution time of 5.26 for the second job of T_2, which has an actual execution time of 7.07. But the PID feedback scheme can get an actual execution time of 6.76 for it, which is again closer to T_2's actual execution time. These examples demonstrate the property of our DVS-EDF scheduler with the PID feedback scheme that the PID scheduler can result in higher energy savings than simple feedback when a system experiences fluctuating task execution times.

7. Correctness

In this section, we show the correctness of our DVS-EDF algorithm, as stated by the following theorem.

THEOREM 1 (CORRECTNESS). *The feedback DVS-EDF algorithm results in a feasible schedule for a set T of tasks with periods equal to their relative deadlines if a feasible schedule exists for T under preemptive EDF.*

Since T has feasible schedules under EDF, we consider only the *maximal* schedule here, *i.e.*, the schedule produced by EDF when the execution time of every task's job has its maximum value given by the WCET. We call the schedule produced by our DVS-EDF algorithm the *actual* schedule,



(a) DVS−EDF Schedule with Simple Feedback

(b) DVS−EDF Schedule with PID Feedback

Fig. 4: Schedules with Simple and PID Feedback

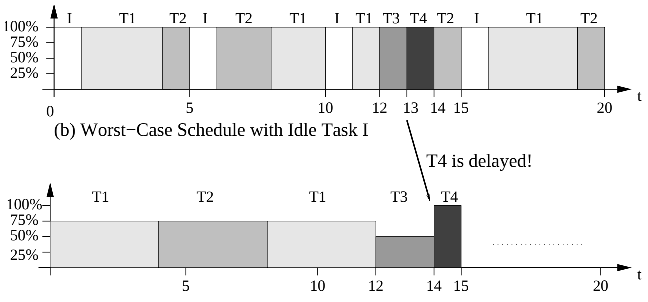
where the execution time of every task's jobs may be scaled. Let s_i and s_i^+ be the absolute start times for T_i according to the actual and the maximal schedule of T, respectively. (We use the shortcut of T_i to denote some j^{th} job T_{ij} of task T_i in the following, and we omit index j in other terms as well to simplify the notation.) Let s_i be the actual start time of T_i. Similarly, let f_i and f_i^+ be the absolute completion times of T_i for the actual / maximal schedule, respectively.

In traditional EDF scheduling, any job's actual start time s_i is less than or equal to its worst-case start time in the maximal schedule. But this is no longer the case in our DVS-EDF schedule. Because DVS-EDF may scale a job's execution time to be larger than its WCET value, a job's actual start time may be later than its start time in the maximal schedule. The next example shows a case where a job's actual start time exceeds its worst-case start time.

Consider the task set in Figure 5(a). Its worst-case schedule with an idle task and its actual schedule under DVS-EDF are shown in Figure 5(b) and Figure 5(c), respectively. When task T_3's second job starts at time 12 in the actual schedule, its absolute deadline is at time 18. There is only one idle slot between time 12 and time 18, which can be used to scale T_3 at a 50% frequency level. Since T_3's actual execution time equals its worst-case execution time, it runs for 2 time units and ends at time 14 with an actual execution time of 2. When T_4 starts execution at time 14, it has been delayed by one time unit relative to its start time in the worst-case schedule.

Task T_i	WCET C_i	Period P_i	c_i	r_i
1	3 ms	8 ms	3 ms	0 ms
2	3 ms	10 ms	3 ms	4 ms
3	1 ms	14 ms	1 ms	4 ms
4	1 ms	20 ms	1 ms	0 ms
idle	1 ms	5 ms	1 ms	0 ms

(a) Task Set



(b) Worst−Case Schedule with Idle Task I

(c)Actual DVS−EDF Schedule

Fig. 5: Delayed Start of Tasks due to Scaling

Proof

Let us return to the correctness of our algorithm. In order to prove the theorem, we will first prove the following lemma:

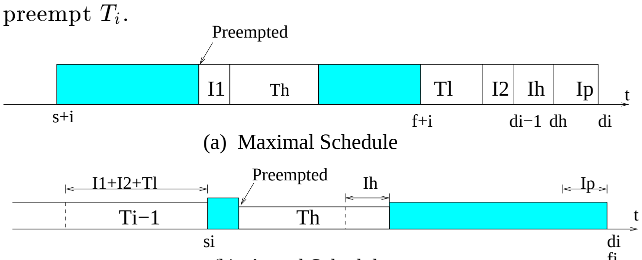
LEMMA 1. *The delay between a job's actual start time and its start time in the maximal schedule is bounded in feedback DVS-EDF by the following inequation:*

$$s_i - s_i^+ \leq idle(f_{i-1}^+, d_{i-1}) + \sum_{T_l \in [f_{i-1}^+, d_{i-1}]; d_l > d_i} C_l \qquad (11)$$

where $idle(f_{i-1}^+, d_{i-1})$ is the length of all idle slots existing between $[f_{i-1}^+, d_{i-1}]$ in the maximal schedule. C_l is the worst-case execution time of job slot T_l in the maximal schedule with a priority lower than T_i. f_{i-1}^+ and d_{i-1} are the completion time and absolute deadline of job T_{i-1}, which is the latest job executed before T_i.

Proof: In the following proof, when a task's execution time is mentioned, we do not distinguish between a split task and a non-split task. This is because our task splitting scheme, as stated by Equation 3, does not change the total execution time of a task's job. When a task is split into two parts (T_A and T_B) running at different frequency levels, their total execution time still equals the original execution time when the task is not split (ignoring DVS overhead). The ratio of T_A and T_B is determined by our feedback schemes, which tries to keep the task at the low frequency level (T_A) as long as possible. But neither the task splitting method nor the feedback schemes impact the timing constraints of the task set. Hence, we describe task properties in the following proof without mentioning task splitting or feedback.

First, the highest priority job always starts at its release time in both algorithms. Hence, $s_i - s_i^+ = 0. \qquad (12)$
Suppose that the i^{th} highest priority jobs satisfy the lemma. We need to show that T_{i+1}, the $(i+1)^{th}$ highest priority job in T, also satisfy the lemma. We only need to consider the case where $s_{i+1} > s_{i+1}^+$, since this is where DVS-EDF diverges from conventional EDF. Because DVS-EDF is priority driven, the only reason for T_{i+1} to be delayed is that some higher priority jobs are still running at time s_{i+1}^+. Let T_i be the latest high priority job executed before T_{i+1}, and let T_{i-1} be the latest high priority job executed before T_i in the actual schedule. Without loss of generality, we assume that in the maximal schedule there are m ($m \geq 0$) idle slots and q ($n \geq 0$) low priority job slot in $[f_{i-1}^+, d_{i-1}]$, namely I_1, I_2,...,I_m and T_l, T_2,...,T_q. We have $\sum_{i=1}^{m} I_i = idle(f_{i-1}^+, d_{i-1})$. Similarly, let $I_h = idle(d_{i-1}, d_h)$ and $I_p = idle(d_h, d_i)$. It is possible that T_i be preempted by a high priority job during its execution, as denoted by T_h in Figure 6. Since both T_{i-1} and T_h have priorities higher than T_i, we have $d_i \geq d_{i-1}$ and $d_i \geq d_h$. We note that at time s_i^+ in the maximal schedule, all other jobs with priorities higher than T_i must have completed, and all other low priority jobs will not be scheduled before f_i^+. Only newly released high priority jobs can execute in $[s_i^+, f_i^+]$ and may preempt T_i.



(a) Maximal Schedule

(b) Actual Schedule

Fig. 6: Transform Maximal to Actual Schedule

Let Equation 11 be the induction hypothesis. By substituting $idle(f_{i-1}^+, d_{i-1})$ with $\sum_{i=1}^{m} I_i$, we can assume that the following holds for the i^{th} highest priority job of T_i in the actual schedule produced by DVS-EDF (cf. Figure 6(b)):

$$s_i - s_i^+ \leq \sum_{i=1}^{m} I_i + C_l = idle(f_{i-1}^+, d_{i-1}) + \sum_{T_l \in [f_{i-1}^+, d_{i-1}]; d_l > d_i} C_l \quad (13)$$

To prove that the lemma is also true for T_{i+1}, we transform the maximal schedule into the actual schedule produced by DVS-EDF. Our induction hypothesis is equivalent to the following transformations: We move I_1, I_2,...,I_m and T_l backward to s_i^+, and we move the corresponding portion of T_i forward. These transformations are legal since idle slots can be moved arbitrarily within $[r_i, d_i]$. The high priority job T_h is left untouched, because T_h always preempts T_i at s_h^+ in the actual case, *i.e.*, $s_h = s_h^+$. When T_i is preempted at time s_h, the forward slack reservation scheme in DVS-EDF reserves $C_i - (s_h - s_i)$, the worst-case remaining execution time left for T_i, from the slots T_l, I_m,...forward. The backward slack reservation scheme reserves the above amount of time from the slots I_p, I_h,...,backward. In either case we denote the total execution time of reserved slots by C_R. At time s_h^+, the frequency scaling decision is made for T_h. The scheduler collects all available idle slots and early completion of low priority job slots in $[s_h^+, d_h]$ in the maximal schedule excluding any slots reserved for future resumption of preempted tasks. The final amount of slack available for T_h equals to $\sum_{i=2}^{m} I_i + I_h + C_l - C_R$. T_h uses the slack to scale itself to a lower frequency and voltage level. These steps are equivalent to the transformations that move the non-reserved portion of I_2,...,I_m,I_h and T_l backward and move the corresponding portion of T_i forward. The result is shown in Figure 6(b). When T_i resumes execution, it can be scaled again exploiting slack from the idle slots (such as I_p in Figure 6) or early-completed task slots before d_i. Similar transformations apply when moving I_p backward and T_i forward. T_i releases all its unused slack when it completes and passes it on to following tasks. Although we only show four idle slots and one low priority job slot in Figure 6, we can simply repeat the transformation if more slots exist. Except for the idle slots and early completion of low-priority job slots (high priority job slots cannot be used as slack in any case), there are no other cases where T_i will be moved forward and thus be delayed during the above transformations. Hence, the following inequation holds:

$$f_i - f_i^+ \leq idle(f_i^+, d_i) - C_R + \sum_{T_l \in [f_i^+, d_i];\ d_l > d_{i+1}} C_l \qquad (14)$$

Because $d_i \geq d_{i-1}$ and $d_i \geq d_h$, the aforementioned transformations never move T_i forward beyond d_i. Hence, T_i will not miss its deadline after these transformations. If the start time of T_{i+1} is delayed in the actual schedule by T_i, we have: $s_{i+1} = f_i$ and $s_{i+1}^+ \geq f_i^+$. From the above equation we get:

$$s_{i+1} - s_{i+1}^+ \leq f_i - f_i^+ \leq idle(f_i^+, d_i) + \sum_{T_l \in [f_i^+, d_i];\ d_l > d_{i+1}} C_l \quad (15)$$

Hence, the lemma holds for T_{i+1}. The lemma implies that the DVS-EDF algorithm always results in a feasible schedule as long as a feasible EDF schedule exists since every feasible EDF schedule can be transformed into a DVS-EDF schedule and no jobs miss their deadlines after these transformations.

The complexity of our algorithm is O(n) for n tasks since at task activation and completion, the length of slots in the maximal schedule for the interval between release time and deadline of the current task have to be assessed. The number of slots in this interval is bounded by the number of tasks since only a constant number of jobs for each task and a constant number of preemptions may occur in this interval.

8. Experiments

We evaluate the performance of our feedback schemes in a simulation environment that supports DVS-EDF scheduling. Different task sets with the total utilization varying from 0.1 to 1.0 with an increment of 0.1 are used in the experiment. In each task set, the workload consists of 10 independent periodic tasks. Each task's relative deadline equals to its period. We assume that each task's worst-case execution time (WCET) is known a priori, which is usually the case for hard real-time systems. We let the actual execution time of each task vary in three different ways. In the first case, jobs of each task have a constant execution time for all its jobs. In the second case, each job has a random actual execution time. All randomizations utilize a normal distribution. This is expected to be the worst case for a feedback control system. In the third case, each job's execution time varies according to a sine curve. We choose the sine curve workload because it represents some of the practical real-time applications, such as multimedia systems and image processing systems, whose task execution times usually fluctuate within a bounded range over time. In order to further observe the behavior of the feedback scheme, we also simulate the mode changes representative for control systems. This is accomplished by dynamically changing the phase and amplitude of the sine curve for the actual execution times of jobs for the same task. Both the simple feedback scheme and the PID feedback scheme are applied to those different task sets to compare their impact on these two schemes. We also implemented the Look-ahead DVS algorithm by Pillai and Shin [12], albeit a modified version since the original algorithm may result in deadline misses (see Appendix for details). The modified Look-ahead algorithm is still the best dynamic DVS scheduling algorithm we know of. We provide different frequency and voltage settings and assume the processor will scale to the lowest frequency level during idle time since it is not realistic to put a processor into sleep mode for frequent task releases. We restrict ourselves here to report results based on four frequency and associated voltage settings, as depicted in Table 1. The choice of the frequency and voltage levels is consistent with Look-ahead DVS work [12] as well as experimental work with the StrongARM [13].

frequency	voltage
25%	2 V
50%	3 V
75%	4 V
100%	5 V

Tab. 1: Processor Model for Scaling

Power levels are calculated in our experiment using a simplified approach as $frequency \times voltage^2 \times time$ and integrated over time to obtain energy consumption. Power values for maximal scaling levels (without consideration of practical discrete frequency levels and task deadline misses) are also computed as the optimal value for the theoretical lower bounds.

8.1 Simple Feedback Scheme

Figure 7 compares the energy consumption of our simple feedback scheme with the Look-ahead DVS scheme [12] as well as the optimal case.

Each task instance has a constant actual execution time equal to 50% of its worst-case execution time. The figure shows that for low utilization our simple feedback scheme results in energy consumption close to the optimal case. At high utilization, our simple feedback scheme still produces low energy consumption, specifically values than do not exceed the optimal case by more than 15%. In all utilization

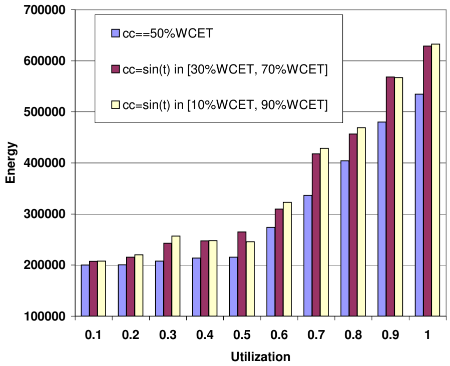


Fig. 8: Fluctuating Times

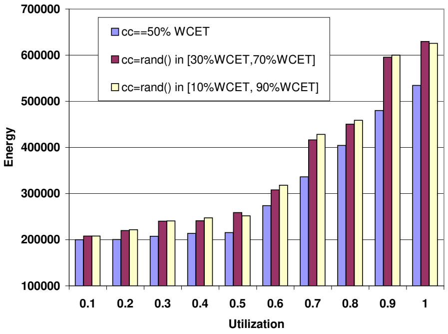


Fig. 9: Random Times

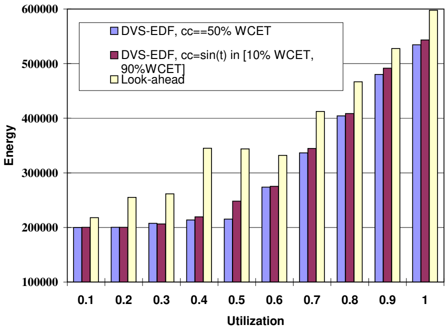


Fig. 10: PID cc=sin(t) 10-90% wcet

cases, our DVS-EDF scheme achieves much lower energy consumption than the Look-ahead DVS method, *i.e.*, up to 37% savings over Look-ahead scheme are observed.

The impact of dynamically fluctuating execution times on our simple feedback scheme is shown in Figures 8 and 9. Figure 8 depicts the energy consumption for task sets with actual execution times varying along two sine curves, one with an amplitude from 30% to 70% of their WCET, the other with an amplitude from 10% to 90% of their WCET. The energy consumption results produced by DVS-EDF with the simple feedback scheme for sine execution times are compared to the results for tasks with constant execution times equal to 50% of their WCET. One would expect these two results be close to each other since the average amplitude of the sine curves are also 50% of WCET. But Figure 8 shows energy consumption results for sine execution times up to 25% larger than those for constant execution times. This is not quite surprising because the simple feedback scheme only considers a proportional adjustment according to the average execution time history. It cannot adapt to variable execution times as well as it does for constant execution times. Figure 9 shows another set of experiments performed on tasks with randomly changing execution times. For each job, its actual execution time is generated randomly within a specific range. The fluctuating amplitude varies as in the previous tests. The average execution time for each task is still 50% of the WCET. Similar results are observed as in the sine execution time experiment. More energy is consumed

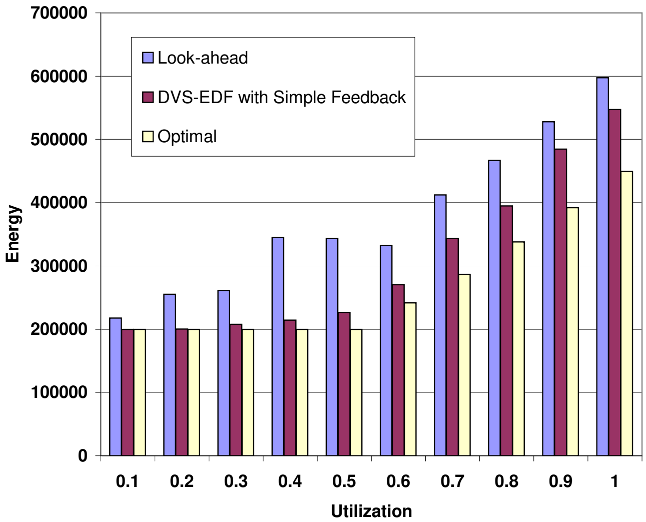


Fig. 7: Energy for $cc = 50\%$ of WCET

on random task sets than on constant execution time tasks.

These three experiments show that the simple feedback scheme can handle the constant execution time tasks well resulting in energy savings up to 37% over previous work. One shortcoming is that it does not adapt as well to variable execution times. A simple linear feedback scheme is usually not suitable in a dynamically changing environment. Inaccurate estimations of C_A cause more jobs to engage in executing their second subtask with the maximum frequency level. We will show in the following experiments how a PID feedback scheme overcomes this shortcoming.

8.2 PID Feedback Scheme

In order to evaluate the effect of the PID feedback scheme on practical real-time workloads, we implemented a PID feedback controller in the DVS-EDF simulation environment and tested different kind of task sets with 10 tasks in each task set. Figure 10 shows the energy consumption for a task set with a sine execution time distribution fluctuating from 10% to 90% of its WCET. In order to make a comparison, the energy consumption for task sets (with constant actual execution times equal to 50% of the WCET under both the DVS-EDF scheme and the Look-ahead DVS scheme) are also depicted in Figure 10. We expect that the energy consumption produced for the dynamic workloads be close to the results for constant workloads as long as they have the same average actual execution times. The PID coefficients of the sine execution time task set are tuned manually for it as $K_p = 0.9$, $I = 0.08$, $D = 0.1$, $IW = 10$ and $DW = 1$. It is observed that either increasing or decreasing the proportional coefficient produces less accurate system estimations for C_A. The derivative item is less significant compared to the other two parameters. The derivative window size is chosen to be 1, and the integral window size is chosen to be 10. Increasing the integral window size improves the energy saving effect in the very beginning, but when IW becomes larger than 10, no dramatic system performance improvements are observed.

In Figure 10, we see that the PID feedback for task sets with sine distributed execution time results in lower energy consumption than the Look-ahead DVS for all utilization levels. The maximum of energy saving over the Look-ahead scheme, up to 30%, is observed at the 40% utilization point. Although the task sets have dynamically fluctuating actual execution times, the DVS-EDF scheme with PID feedback results in slightly higher energy consumption than task sets with constant execution time. This is a significant improvement over our previous simple feedback scheme .

When we restrict the amplitude of the task execution

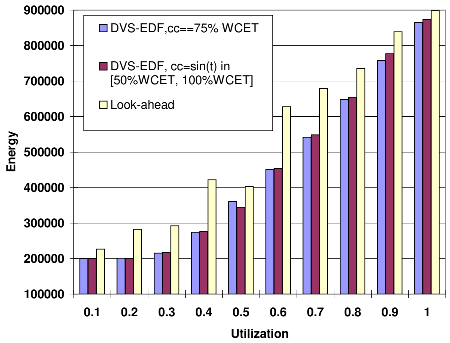


Fig. 11: cc=sin(t) 50%..100% wcet

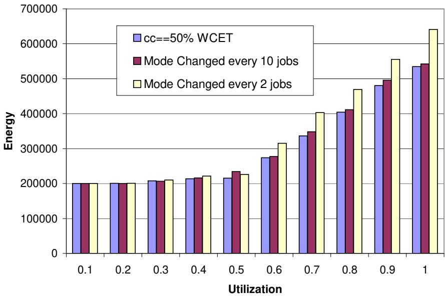


Fig. 12: PID sin(t)..sin(3t) 10-90% and 30-70% wcet

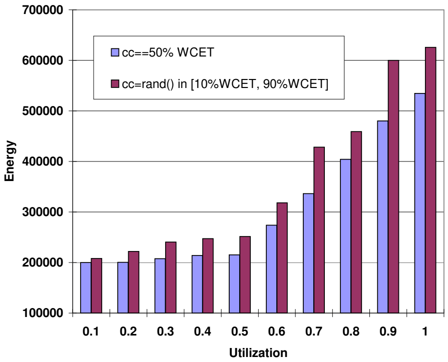


Fig. 13: PID w/ Random Times

times (still along a sine curve) to a smaller range from 50% to 100% of the WCET with an average of 75% WCET (as shown in Figure 11), PID feedback gives us energy consumptions even closer to those for the constant execution times task sets. At 50% utilization, the PID feedback scheme consumes even less energy than the constant execution time task set. This can be attributed to the adaptiveness of the PID controller when matching ideal α scaling levels and the actual α' levels supported by an architecture.

We further constructed task sets with mode change behavior, which is often observed in real control systems, by dynamically changing the phase of the sine curve for the tasks' actual execution times. Figure 12 shows the energy consumption results with PID feedback for two of these task sets whose modes alternate between sin(t) and sin(3t). Their amplitude changes between [10%WCET, 90%WCET] and [30%WCET, 70%WCET]. The first task set changes its mode every 10th job, while the second task set changes its mode every second job. The energy consumption results for constant execution time task set are also depicted in Figure 12 for comparison. We see that our PID feedback scheme still tracks the dynamic execution behavior of task sets well as long as the mode change does not occur too frequently within a PID feedback window, which is the case for the first task set whose mode changes every 10th job. It produces a schedule with an energy consumption close to the constant execution time case. Only when the mode change happens too frequently within the PID window, as the second task set shows in Figure 13, will our PID feedback controller fail to obtain useful information about the tasks' past execution times. Hence, we may observe high energy consumption. This demonstrates the limitation of feedback in general.

The limits of the PID feedback scheme are observed in an extreme case, where we apply PID feedback control on task sets with randomly distributed execution times between 10% and 90% of their WCET with an average of 50% WCET. As shown in Figure 13, PID feedback produces no additional benefits for the random execution time task sets and results in high energy consumption compared to the constant execution time cases. Randomly distributed execution time cannot provide any useful history information to the PID controller to adjust the scheduler for the execution of the next job of a task. Only when the tasks' behavior presents some continued tendencies over a period of time can the PID feedback scheme provide benefits in the estimation execution times for future jobs.

Overall, our Feedback DVS-EDF algorithm achieves a low energy consumption level for different workloads and exhibits considerable energy savings not only for ideal environments but also for most realistic scenarios. Our previous work [5] has shown that DVS-EDF algorithm outperforms Look-ahead DVS [12] in ideal environments where task sets have constant execution times. Our experiments here demonstrate that these results still hold for future reservation during preemption. They also show that PID feedback control in conjunction with our DVS-EDF scheme makes the scheduler more adaptive to dynamically changing system workloads. Experiments approximate the optimal energy consumption levels more closely than less adaptive schemes.

9. Related Work

There have been a number of efforts for applying feedback techniques into general-purpose control systems. But only recently did researchers begin to incorporate feedback control to real-time scheduler with timing constraints [10, 11]. Lu *at al.* proposed a feedback control real-time scheduling framework for unpredictable dynamic real-time systems where task execution times diverge from the worst case [11]. Real-time system performance specifications are analyzed and satisfied systematically through a control theory-based methodology. Dynamic models of real-time systems are developed, which can identify different categories of real-time applications with different feedback control algorithms. While their feedback control framework is mainly used to satisfy real-time system requirements, our algorithm focuses on exploiting feedback control schemes to reduce energy consumption in embedded systems.

Our work is more closely related to that described in [11, 14]. Lu *et al.* describe a formal feedback control algorithm combined with dynamic voltage/frequency scaling technologies for multimedia systems [11]. Both continuous and discrete DVS settings are exploited in a scheme to reduce energy consumption while still guaranteeing real-time requirements. An adaptive set-point is used to achieve fast responses with a stable multimedia throughput. Although their work and our approach exploit feedback control to DVS/DFS technologies, their work targets soft real-time requirements, *e.g.*, for multimedia systems. Our work focuses on hard real-time systems where timing constraints must not be violated.

A general energy management scheme with feedback control was proposed by Minerick *at al.* [14]. An average energy usage is achieved by continuously adjusting the power setting on the hardware architecture to meet the energy consumption goal. A PI (proportional and integral) feedback controller is used to adapt the proper power setting based on previous energy consumptions without the prediction of future system workloads. While their objective is to obtain low energy consumption for general purpose systems, our work targets real-time systems to exploit energy saving potentials through different feedback control schemes.

Other approaches related to our DVS scheme are [12, 1, 6]. Pillai and Shin proposed a set of dynamic DVS algorithms based on traditional hard real-time mechanisms, namely rate-monotone (RM) scheduling and EDF scheduling [12]. They extend the schedulability test of RM and EDF algorithms to incorporate CPU frequency scaling. Unlike our algorithm that applies frequency scaling to only the current task, they assume the same frequency scaling factor upon all tasks in a real-time task set. In their most aggressive variant, a look-ahead technique is used to achieve extensive energy savings by deferring as much work as possible. However, the frequency value obtained in their algorithm is not always the lowest possible frequency for a single task, as comparisons with the optimal case and with our work show.

Other aggressive real-time DVS schemes speculate on and exploit early completion of task executions based on statistical information about the workload for dynamic scheduling [1] and static priority scheduling [6]. Our scheme adapts even to dynamically changing execution demands, not just statistical information, due to its feedback mechanism, which is shown to be most effective for dynamic scheduling. The idea of deriving a feasible dual-level DVS schedule from an ideal case was first proposed by Gruian [6, 7]. It combines off-line and on-line scheduling, both at task level and task-set level. Stochastic data is used to derive energy-efficient schedules. We use a different method by splitting a task into two parts and always assigning the highest frequency to the second part. Our algorithm focuses on dynamic scheduling (EDF) while Gruian restricts his approach to fixed-priority schemes. Last-chance scheduling algorithms without energy considerations goes back at least to Chetto and Chetto [3]. We apply this philosophy in a DVS context but not to the entire task, we develop a novel variant based on task splitting with exactly two parts. As discussed in the motivation, the dual-subtask approach is optimal (due to the second/third order polynomial of the energy consumption as a function of frequency) if the first subtask is fully utilized while the second subtask never executes. Our feedback approach triggers this behavior, which is superior to Gruian's step-wise increase of frequencies in the stochastic approach.

Past DVS schemes were based on early completion of tasks and idle time up to the next task's activation. Statically compiled worst-case schedules (term the alpha-queue) were used to quantify these savings to the dynamic schedule by Aydin *et al.* [1]. Our work exploits not just the idle prior to the next task's activation but any idle time up to the deadline of the task from the static worst-case schedule. In addition, an "idle task" provides guarantees on slack generation for scaling that exceed those of prior work.

10. Conclusion

This paper presents a novel scheduling approach combining DFS/DVS with different feedback control schemes, which extends EDF in a most aggressive manner. The technique relies strictly on operating system support to implement both the real-time scheduler and feedback controller. Our contributions include techniques for preemption handling, feedback control and a proof of correctness of the proposed algorithm. A simple feedback scheme and a PID feedback scheme are applied on different system workloads with varying execution times. Simplistic proportional feedback is shown to work well for constant execution time task sets. For predictably fluctuating execution time task sets, simple feedback is enhanced by a PID-controller to adapt to dynamically changing system workloads. The feedback DVS scheduling scheme provides up to 37% additional savings over prior published work combined with low runtime complexity, and the approach scales for varying number of tasks. Its support for PID-controlled feedback is shown to result in an additional 30% energy savings even when execution times diverge as much as $\pm 40\%$ from their WCET between jobs of the same task, which is a commonly observed behavior in embedded control systems. Overall, the feedback DVS scheme is the most aggressive dynamic scheduling approach to conserve energy through DVS/DFS for hard real-time systems that we know of.

Acknowledgments

Ajay Dudani contributed towards our early work on the Feedback-DVS scheme [5]. Problems in the Look-ahead DVS-EDF algorithm were discovered together with Harini Ramaprasad and Sibin Mohan.

11. REFERENCES

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and agressive scheduling techniques for power-aware real-time systems. In *IEEE Real-Time Systems Symposium*, December 2001.

[2] A.P. Chandrakasan, S. Sheng, and R. W. Brodersen. Low-power cmos digital design. In *IEEE Journal of Solid-State Circuits, Vol. 27, pp. 473-484.*, April, 1992.

[3] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, October 1989.

[4] J. Kim D. Shin and S. Lee. Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design and Test of Computers*, March 2001.

[5] A. Dudani, F. Mueller, and Y. Zhu. Energy-conserving feedback edf scheduling for embedded systems with real-time constraints. In *ACM SIGPLAN Joint Conference Languages, Compilers, and Tools for Embedded Systems (LCTES'02) and Software and Compilers for Embedded Systems (SCOPES'02)*, pages 213–222, June 2002.

[6] F. Gruian. Hard real-time scheduling for low energy using stochastic data and dvs processors. In *Proceedings of the International Symposium on Low-Power Electronics and Design ISLPED'01*, Aug 2001.

[7] F. Gruian and Kuchcinski. Lenes: task scheduling for low-energy systems using variable voltage processors. In *Proceedings of ASP-DAC*, 2001.

[8] D. Kang, S. Crago, and J. Suh. A fast resource synthesis technique for energy-efficient real-time systems. In *IEEE Real-Time Systems Symposium*, December 2002.

[9] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[10] C. Lu, J. Stankovic, G. Tao, and S. Son. Design and evaluation of a feedback control edf scheduling

algorithm. In *IEEE Real-Time Systems Symposium*, December 1999.

[11] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *International Conference on Compilers, Architectures, and Synthesis for Embedded Systems*, pages 156–63, 2002.

[12] P. Pillai and K. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles*, 2001.

[13] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor, 2000.

[14] V. W. Freeh R. Minerick and P. M. Kogge. Dynamic power management using feedback. In *In Proceedings of Workshop on Compilers and Operating Systems for Low Power*, 2002.

[15] Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Int'l Conf. on Computer-Aided Design*, 2000.

[16] J. Wegener and F. Mueller. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, 21(3):241–268, November 2001.

[17] Y. Zhu and F. Mueller. Preemption handling and scalability of feedback dvs-edf. In *Workshop on Compilers and Operating Systems for Low Power*, September 2002.

Modified Look-ahead DVS-EDF

A number of DVS schemes were proposed by Pillai and Shin for scheduling hard real-time systems [12]. A simple, *static* scaling version uniformly scales the frequency for all tasks based on utilization tests for schedulability, both for rate-monotone and EDF scheduling. *Cycle-conserving* EDF lowers utilization upon task completion temporarily to the proportion of the actual execution time. *Look-ahead* EDF is an extension to these scheme that capitalizes on early task completion by deferring work for future tasks in favor of scaling the current task. Scaling of the current task occurs based on a modified utilization test that benefits from both idle slots and early task completion. At any completion (both early and on time), the utilization is effectively reduced for the completing task (up until its next release time).

Specifically, upon task completion, $cc_i = c\_left_1 = 0$ according to Cycle-Conserving EDF and Look-ahead EDF, respectively. The *defer* calculations of Look-ahead EDF then reassesses the utilization based on future and past deadlines for released and completed tasks, respectively.

We modified the Look-ahead EDF by setting $c\_left_i = C_i$ at task completion instead of assigning a zero value. In addition, we reassess the utilization *strictly* based on the next deadline in the future, irregardless of whether tasks are already released and not. This allows us to look ahead even further in the schedule and, thereby, potentially save additional energy by lowering frequencies more aggressively, and it retains the safety of the schedule by adhering to the EDF utilization test. If the WCET is not fully utilized, then other tasks may still benefit from early completion up to the threshold given by the idle times left in the schedule. This modified Look-ahead EDF scheme was implemented in our comparison and is shown to result in up to 34% lower energy consumption than the original scheme. On the average, the modified scheme saves an additional 5-11% of energy for utilizations between 25% and 100%. At high utilizations, our modification occasionally requires between 0.5-8% more energy, which is due to considering an actual time of $cc_i = 0$ in the original scheme up to the next release of a task. Hence, it would be possible to switch between the two schemes based on a utilization threshold as a trigger. Additional savings over the modified scheme due to early completion can only be obtained by considering the density of a schedule at some instance in time, such as given by the maximal schedule utilized in our feedback EDF scheme.