UNIVERSITÄT SALZBURG

Power-aware Temporal Isolation with Variable-Bandwidth Servers

Silviu S. Craciunas   Christoph M. Kirsch   Ana Sokolova

Technical Report 2010-02   April 2010

Department of Computer Sciences

Jakob-Haringer-Straße 2
5020 Salzburg
Austria
`www.cosy.sbg.ac.at`

Technical Report Series

Power-aware Temporal Isolation with Variable-Bandwidth Servers*

Silviu S. Craciunas Christoph M. Kirsch Ana Sokolova

Department of Computer Sciences
University of Salzburg, Austria
firstname.lastname@cs.uni-salzburg.at

Abstract. Variable-bandwidth servers (VBS) control process execution speed by allocating variable CPU bandwidth to processes. VBS enables temporal isolation of EDF-scheduled processes in the sense that the variance in CPU throughput and latency of each process is bounded independently of any other concurrently running processes. In this paper we aim at reducing CPU power consumption with VBS by CPU voltage and frequency scaling while maintaining temporal isolation. Scaling to lower frequencies is possible whenever there is CPU slack in the system. We first show that, in the presence of CPU slack, frequency scaling of EDF-scheduled, possibly non-periodic tasks (as they arise with VBS) is safe up to full CPU utilization and propose a frequency-scaling VBS algorithm that exploits CPU slack to minimize operating frequencies with maximal CPU utilization while maintaining temporal isolation. This may lead to improvements in power consumption while hiding the real-time effects of frequency scaling. Additional power may be saved by redistributing computation time of individual processes while still maintaining temporal isolation if the system has knowledge of future events. We introduce an offline algorithm as an optimal baseline and an online algorithm that approximates the baseline. While the offline algorithm works for various, possibly complex power consumption models, the online algorithm may reduce power consumption only for a simplified power consumption model by reducing the CPU utilization jitter in the system.

1 Introduction

We study methods to reduce CPU power consumption of software processes scheduled with variable-bandwidth servers (VBS) [8, 7]. A VBS is similar to a constant-bandwidth server (CBS) [1]. The difference is that while CBS allocates a constant fraction of CPU time to a process (the server bandwidth) at a constant granularity (the server period), VBS allows the process to change both server bandwidth and server period. This enables a VBS process to change its execution speed at runtime, as long as the resulting CPU utilization remains under a given

* Supported by the EU ArtistDesign Network of Excellence on Embedded Systems Design and the Austrian Science Funds P18913-N15 and V00125.

bandwidth cap. Similar to CBS, multiple VBS processes are EDF-scheduled with deadlines equal to the server periods.

Process code that is executed at a constant speed with VBS is called an action. A VBS process is therefore a sequence of actions. The response time of an action is the real time it takes an action to execute from arrival to termination. For each action of a VBS process, there exist lower and upper bounds on response times and thus also on response time jitter that are independent of any other, concurrently running processes, as long as CPU utilization (the sum of all bandwidth caps) is less than or equal to 100% [8, 7] (Section 3). VBS enables temporal isolation of processes on the level of individual actions. Temporal isolation allows actions to maintain their response time bounds and jitter even in the presence of other, concurrently running processes in the system.

Modern processors often support dynamic scaling of CPU voltage and operating frequency, which opens up the general possibility to reduce CPU power consumption. We first relate our work on CPU voltage and frequency scaling with VBS to other research in the area of power-aware computing (Section 2).

We then show that, in the presence of CPU slack, frequency scaling of EDF-scheduled, possibly non-periodically arriving tasks (such as the tasks from the actions of VBS processes) is safe up to full CPU utilization. We identify the sources of CPU slack in VBS-scheduled systems and propose a frequency-scaling VBS algorithm, which exploits CPU slack to minimize operating frequencies with maximal CPU utilization while maintaining temporal isolation (Section 4).

In a VBS-scheduled system we distinguish two types of slack, static and dynamic. Static slack is given by the total sum of bandwidth caps of the VBS processes. With static slack, the operating frequency is scaled back to the reciprocal of the static slack computed once at boot time and never changed at runtime. Considering dynamic slack enables further reductions in operating frequency. Dynamic slack may occur at runtime and varies depending on action parameters. We distinguish two types of dynamic slack, action and termination slack. Action slack occurs whenever a VBS process switches to an action that utilizes the CPU below the bandwidth cap of the process. Termination slack is the result of the so-called VBS termination strategy, which delays the transition from one action to the next by postponing the actual completion of an action until its logical termination at the end of the server period in which the action completed. The termination slack of an action may be used to decrease CPU utilization by decreasing the action's server bandwidth such that the action still completes within its response time bounds. We present methods on how to exploit both types of dynamic slack, individually and combined.

We consider the problem of reducing power consumption further by giving the scheduler even more freedom to redistribute computation time of actions among the server periods during which the actions execute without affecting the actions' original response time bounds (Section 5). In order to use this freedom, the system must have knowledge of future actions. Furthermore, the problem of reducing power consumption depends on the specific power profile of the CPU as well as the number of frequency switches performed. We present an optimal offline algorithm that minimizes a given power-consumption function (that may incorporate such a power profile and switching overhead) by computing the best possible configuration of server bandwidths for every server period during which an action executes. The optimal offline algorithm is a baseline for an online algorithm that is feasible for real-time systems. Given a simplified power consumption model, the online algorithm decreases CPU utilization jitter to approximate the optimal algorithm by steering actual CPU utilization towards a computed average.

We show in a series of experiments using simulated processes and actions that by combining the two types of slack more power can be saved than by exploiting just action slack alone, and additionally that the best power consumption savings are achieved through the optimal offline algorithm (Section 6).

We conclude the paper and give an outlook on future work in Section 7.

2 Related Work

Reducing the operating frequency of a CPU allows a reduction of its supply voltage [5, 16] and may therefore reduce power consumption. The consumed power P is proportional to the square of the voltage level V. This dependency is typically expressed as

$$P \propto f \cdot V^2,$$

where f is the clock frequency.

We relate our work to previously published research in the area of power-aware scheduling algorithms. The main technique used is Dynamic Voltage scaling (DVS), which in the context of real-time systems aims at reducing power consumption without negatively affecting the timing properties of tasks.

Most DVS algorithms for real-time systems reclaim the unused bandwidth of tasks that finish their execution faster than their worst-case execution times indicate. The VBS process model differs from the standard model in that it assumes constant execution time for an action. Furthermore, we aim at reducing CPU power consumption while maintaining temporal isolation on the level of actions rather than tasks. In [12, 11], periodic processes with deadlines equal to periods are considered. There are two phases, an offline phase, which calculates the voltage level such that, if all processes run for their worst-case execution times, the timing requirements are still met, and an online phase which tries to reclaim unused execution time. In addition to the two phases, the work in [16] presents a look-ahead mechanism that tries to reduce power consumption beyond the conservative approaches by determining future computation needs of processes. A static optimal solution assuming worst-case workload, online speed reduction considering actual workload, and an online speculative speed adjustment to anticipate early completions are presented in [2]. Other studies use similar approaches by relaxing the periodicity assumption on the process model [17, 19]. Similar to our algorithm, the work in [17] modifies the frequency of EDF-scheduled systems at runtime according to the current utilization of released tasks.

Another class of algorithms, called intra-task DVS [3], make use of the execution characteristics of processes and require compiler support to reduce power consumption. Such methods allow CPU voltage and frequency scaling within the process boundary, by calculating the slack of executed segments to reduce the voltage for further segments. One important issue is how to divide a program into segments since more frequent changes in voltage result in more efficient use of slack but also imply more power and time overhead.

VBS is related to resource reservation scheduling mechanisms like CBS [1], elastic scheduling [6] and RBED [4]. Reduced power consumption for systems employing resource reservation techniques is presented in [18]. The algorithm, called GRUB-PA, reclaims unused processor capacity and uses it to slow down the processor without affecting the timing properties of processes. Elastic scheduling is combined with DVS [14] to either improve performance or power consumption for real-time systems. In [13] power management is integrated with the RBED scheduling framework by exploiting unused computation time generated by the early completion of processes. Additionally, a more exact power model and a frequency switching overhead analysis are also considered. Our method is different in that it does not rely on early completion of processes for minimizing power consumption and it maintains the relevant temporal properties on the level of actions rather than tasks.

3 VBS Scheduling

We briefly recall the necessary definitions and results of VBS scheduling which was introduced in previous work [8, 7]. A variable-bandwidth server (VBS) is uniquely determined by a utilization u that represents an upper bound bandwidth cap. A VBS process may allocate a fraction of CPU time over a time interval as long as this fraction stays below the given bandwidth cap. VBS is an extension of a constant-bandwidth server (CBS) [1]. While CBS allocates a constant fraction of CPU time to a process (the server bandwidth) over a constant time interval (the server period), VBS allows the process to change both server bandwidth and server period, as long as the bandwidth remains under the given bandwidth cap. This enables a VBS process to change its execution speed at any time. VBS employs virtual periodic resources [20], defined as a pair of a period and a limit (with bandwidth equal to the ratio of limit over period). A VBS can change its execution speed by changing the virtual periodic resource. A process running on a VBS can initiate a resource switch at any time. The portion of process code from a change in speed to the next change is called an action. A VBS process is therefore a sequence of actions. Note that the execution of a process can be an unbounded sequence of actions but the available virtual periodic resources are finite, which results in a finite schedulability condition. The time it takes to execute an action is explicitly modeled and called the response time of the action. The key result of VBS is that, for each action of a VBS process, there exist lower and upper bounds on response times and thus also on jitter that are independent of any other concurrently executing VBS processes, as long as system utilization (the sum of all bandwidth caps) is less than or equal to 100% [8, 7]. This property enables temporal isolation.

More formally, a process $P(u)$ corresponding to a VBS with utilization u is a finite or infinite sequence of actions,

$$P(u) = \alpha_0\alpha_1\alpha_2\ldots$$

for $\alpha_i \in \mathrm{Act}$, where $\mathrm{Act} = \mathbb{N} \times \mathcal{R}$, with $\mathcal{R}$ being the finite set of virtual periodic resources explained below. An action $\alpha \in \mathrm{Act}$ is a pair $\alpha = (l, R)$. The load l of an action is the exact amount of time the action needs to execute on R. The virtual periodic resource R is a pair $R = (\lambda, \pi)$ of natural numbers with $\lambda \leq \pi$, where λ denotes the limit and π the period of the resource. The limit λ specifies the maximum amount of time the process P can execute on the virtual periodic resource within the period π, while performing the action α. The utilization of R is $u_R = \frac{\lambda}{\pi} \leq u$.

3.1 Schedulability Analysis

Let $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ be a finite set of n processes with corresponding actions $\alpha_{i,j} = (l_{i,j}, R_{i,j})$ for $j \geq 0$. Each $P_i(u_i) = \alpha_{i,0}\alpha_{i,1}\ldots$ corresponds to a VBS with utilization u_i. Let $R_{i,j} = (\lambda_{i,j}, \pi_{i,j})$ be the virtual periodic resource associated with the action $\alpha_{i,j}$ with $l_{i,j}, \lambda_{i,j}$, and $\pi_{i,j}$ being the load, the limit, and the period for the action $\alpha_{i,j}$.

Proposition 1 ([7, 8]). *Given a set of VBS processes $\mathcal{P}$, if*

$$\sum_{i \in I} u_i \leq 1,$$

then the set of processes $\mathcal{P}$ is schedulable within the upper and lower response-time bounds, $b^u_{i,j}$ and $b^l_{i,j}$, explained below.

Given a schedule for $\mathcal{P}$, for each process $P_i \in \mathcal{P}$ and each action $\alpha_{i,j} = (l_{i,j}, R_{i,j})$ that appears in P_i we distinguish four absolute moments in time:

- Arrival time is the time instant at which $\alpha_{i,j}$ arrives. The first action has zero arrival time while for all others the arrival is the time instant at which the previous action of the same process has been terminated.
- Completion time of $\alpha_{i,j}$ is the time at which the action finishes executing its load.
- Termination time of α_i is the time at which the action is terminated. There is a difference between completion and termination. Termination is set to be at the end of the period within which the action has completed. The process can only invoke its next action if the previous one has been terminated.
- Release time is the earliest time when α_i can be scheduled. In VBS there are two release strategies which determine if the action is released immediately upon arrival (early strategy) or delayed until the next period instance (late strategy).

Given that the set $\mathcal{P}$ of VBS processes is schedulable, its schedule is obtained using EDF for the set of tasks resulting from each action. Namely, each action can be seen as a sequence of EDF tasks with release times at period instances (only the first task in the early strategy has release time equal to the arrival time), computation times equal to the limit (except for the last task in both strategies and for the first task in case of early release, which may have smaller computation time), and deadlines equal to the next period instance.

The upper bound for the response time of action $\alpha_{i,j}$ is given by

$$b^u_{i,j} = \pi_{i,j} - 1 + \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j}.$$

The lower response-time bound varies depending on the strategy used, namely

$$b^l_{i,j} = \begin{cases} \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \pi_{i,j} \text{ , for late release} \\ \\ \left\lfloor \frac{l_{i,j}}{\lambda_{i,j}} \right\rfloor \pi_{i,j} \text{ , for early release.} \end{cases}$$

Using these bounds, we can compute the response-time jitter of an action $\alpha_{i,j}$, i.e., the difference between the upper and lower bound on the response time. We differentiate between the logical response time jitter, which is the time from arrival until termination, and the actual response time jitter given by the time from arrival to completion. For both strategies the logical response time jitter is at most $\pi_{i,j} - 1$ while the actual is bounded by $2(\pi_{i,j} - 1)$. This differentiation is based on the time at which an action effect takes place, i.e., whether the effect of the action is at termination and not before, or whether the action has an effect on the system at completion.

3.2 VBS Utilization Slack

We minimize operating frequency of the processor by maximizing system utilization which may result in lower power consumption. In order to achieve this, we must identify the sources of slack in the system. This slack can be then used to scale down the maximum frequency f_{max} to a new frequency f_{new} such that all actions still respect their response time bounds. Let

$$\eta = \frac{f_{new}}{f_{max}}$$

be the frequency-scaling factor, the ratio between the new processor frequency and the maximum frequency. If the processor has been scaled with the factor η, a load of 1 time units will take $\frac{1}{\eta}$ time units.

Consider a set of two VBS processes with one action each, namely $P_1(0.25)$ with action $\alpha_{1,1} = (5, (1, 4))$ and $P_2(0.25)$ with action $\alpha_{2,1} = (6, (3, 12))$. Scheduled with VBS, the system would be idle for 50% of the time. We can use this idle time to reduce the frequency of the processor such that the actions still meet their response time bounds. In this simple example, the system behaves like an ordinary periodic EDF-scheduled system due to no action switches. For EDF, computing the minimum speed at which the processes still meet their deadlines is straightforward. For a system of n periodic EDF-processes with utilization

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i},$$

where C_i is the computation time and T_i is the period of process P_i, the minimum frequency at which the processes still meet their deadline is given by the frequency-scaling factor $\eta = U$ [21]. Since in this example the two VBS processes are periodic EDF processes, we can scale the frequency of the processor to 50% such that the actions become $\alpha_{1,1} = (10, (2, 4))$ and $\alpha_{2,1} = (12, (6, 12))$. The response time bounds of the actions are the same even when running at this lower frequency.

In general the VBS process model differs from the periodic EDF process model so we cannot use this metric straightforwardly due to the changing and interleaving of the actions. We also cannot use traditional slack estimation techniques that take advantage of the WCET of processes because we assume that the load of an action is fixed. A VBS system has two types of slack, static and dynamic. The static slack results from the total system utilization, i.e. the sum of the bandwidth caps. This static slack provides a reference frequency for our system of VBS processes.

The dynamic slack results from the property of VBS that actions may have utilization lower than the bandwidth cap and from the termination strategy. In the following section we will describe the static and dynamic slack and use the dynamic slack to reduce the frequency even further than the reference frequency while still allowing all actions to finish their execution within their response time bounds.

4 Frequency-scaling VBS

4.1 EDF frequency scaling

Before we proceed with frequency-scaling VBS, we present a general result for frequency scaling of EDF tasks, on which all our other results rely.

Lemma 1. *An EDF-schedulable set of tasks with release times, computation times, and deadlines, is still schedulable if the processor frequency in between any two release times is set to at least $U_c \cdot f_{max}$, with U_c being the current total utilization of all released tasks in the considered interval of time between two releases.*

Proof. We are first going to prove the following auxiliary statement. Let $T = \{\tau_i \mid \tau_i = (C_i, T_i) \text{ for } 1 \leq i \leq n\}$ be an EDF-schedulable set of tasks released at time 0, with task τ_i having computation time C_i and deadline T_i. Moreover, without loss of generality we may assume that T is ordered by the deadlines, that is $T_i \leq T_j$ for $i \leq j$. Let r be the first time instant at which a new task is released. Then scaling the frequency by the total utilization in the time interval $[0, r)$ is safe. More precisely, the utilization of all tasks remaining at time r, having scaled the frequency in $[0, r)$, is not larger than the original utilization of the tasks in T.

The utilization of task τ_i is the ratio $u_i = \frac{C_i}{T_i}$. The total utilization of the set T is therefore

$$U = \sum_{i \leq n} \frac{C_i}{T_i}.$$

Since the set T is EDF schedulable, we have that $U \leq 1$, and therefore it allows for frequency scaling by factor $\eta = U$. Assume there is a frequency scaling by U in the interval $[0, r)$. This means that the actual amount of work done in this interval is (at most) $r \cdot U$, but this work takes time (at most) r.

Let k be the largest task index such that

$$r - \sum_{i<k} \frac{1}{U} \cdot C_i \geq 0,$$

meaning that $r - \sum_{i \leq k} \frac{1}{U} \cdot C_i < 0$. If no such index exists, then all tasks, even though scaled, can finish their computation up to time r, so the statement trivially holds. Note that it is also possible that $k = 1$.

The crucial fact now is that in the interval of time $(0, r)$ no task is released. Therefore, using that the tasks were ordered by deadlines, the schedule after scaling looks like this: Task τ_1 runs first having the earliest deadline for $\frac{1}{U} \cdot C_1$ time units (if $k > 1$), then task τ_2 runs for $\frac{1}{U} \cdot C_2$ time units (if $k > 2$), and so on, task τ_{k-1} runs for $\frac{1}{U} \cdot C_{k-1}$ time units, and finally task τ_k runs for whatever remains until time r. It is important to notice that

$$\sum_{i \leq m} \frac{1}{U} \cdot C_i \leq T_m$$

for all $m \leq n$ since

$$\sum_{i \leq m} \frac{C_i}{T_m} \leq \sum_{i \leq m} \frac{C_i}{T_i} \leq U.$$

As a result, also $T_k > r$.

By this, the tasks up to τ_{k-1} finish their computation up to time r, whereas part of the work of τ_k remains, and the tasks $\tau_{k+1}, \ldots, \tau_n$ have not even been scheduled yet, i.e., all their computation work remains to be done after time r. Note that the actual work (scaled down to original frequency) done by τ_k up to time r can be computed as the difference from all the actual work done up to r and the work done by all tasks with smaller index (earlier deadline):

$$r \cdot U - \sum_{i<k} C_i.$$

Therefore, the remaining computation work of τ_k is then

$$C_k - \left(r \cdot U - \sum_{i<k} C_i \right).$$

and hence the "remaining utilization" of τ_k or rather the utilization of the k-th task after time r is

$$\begin{aligned}\frac{C_k - \left(r \cdot U - \sum_{i<k} C_i\right)}{T_k - r} &= \frac{\sum_{i\le k} C_i - r \cdot U}{T_k - r} \\ &= \sum_{i\le k} \frac{C_i}{T_k - r} - \frac{r \cdot \sum_{i\le n} \frac{C_i}{T_i}}{T_k - r}.\end{aligned}$$

Furthermore, the "remaining utilization" of τ_i for $i > k$ is

$$\frac{C_i}{T_i - r}.$$

No other tasks remain in the system. We need to show that the total utilization of the remaining tasks does not exceed the original remaining utilization at time r which consists of the utilization of all original tasks with deadlines after r. Let m be the largest index such that $T_m \le r$. We have that $m < k$.

Hence, we need to show that

$$\sum_{i\le k} \frac{C_i}{T_k - r} - \frac{r \cdot \sum_{i\le n} \frac{C_i}{T_i}}{T_k - r} + \sum_{i>k} \frac{C_i}{T_i - r} \le \sum_{i=m+1}^{n} \frac{C_i}{T_i}. \tag{1}$$

Inequality (1) is equivalent to

$$\sum_{i\le k} \frac{C_i}{T_k - r} + \sum_{i>k} \frac{C_i}{T_i - r} \le \sum_{i=m+1}^{n} \frac{C_i}{T_i} + \frac{r \cdot \sum_{i\le n} \frac{C_i}{T_i}}{T_k - r}$$

which, after multiplying by $T_k - r$ and some rearranging, becomes

$$\sum_{i\le k} C_i + (T_k - r) \cdot \sum_{i>k} \frac{C_i}{T_i - r} \le T_k \cdot \sum_{i=m+1}^{n} \frac{C_i}{T_i} + r \cdot \sum_{i\le m} \frac{C_i}{T_i}.$$

Having that

$$\sum_{i\le m} C_i \le r \cdot \sum_{i\le m} \frac{C_i}{T_m} \le r \cdot \sum_{i\le m} \frac{C_i}{T_i}$$

it would suffice to prove that

$$\sum_{i=m+1}^{k} \frac{C_i}{T_k} + \frac{T_k - r}{T_k} \cdot \sum_{i>k} \frac{C_i}{T_i - r} \le \sum_{i=m+1}^{n} \frac{C_i}{T_i}. \tag{2}$$

We have $\frac{C_i}{T_k} \leq \frac{C_i}{T_i}$ for $i \leq k$ since then $T_i \leq T_k$ and hence

$$\sum_{i=m+1}^{k} \frac{C_i}{T_k} \leq \sum_{i=m+1}^{k} \frac{C_i}{T_i}.$$

For the rest, if $i > k$, we first notice that

$$\frac{T_k - r}{T_k} \cdot \frac{C_i}{T_i - r} = \frac{C_i}{T_i} \cdot \frac{(T_k - r) \cdot T_i}{T_k \cdot (T_i - r)}.$$

Hence, it would suffice to prove that

$$(T_k - r) \cdot T_i \leq T_k \cdot (T_i - r) \text{ for } i > k$$

which holds using the following arguments. For $i > k$, we have $T_i \geq T_k$ and hence $(T_k - r) \cdot T_i = T_k \cdot T_i - r \cdot T_i \leq T_k \cdot T_i - r \cdot T_k = T_k \cdot (T_i - r)$. As a result, we have that for $i > k$,

$$\frac{T_k - r}{T_k} \cdot \frac{C_i}{T_i - r} \leq \frac{C_i}{T_i}$$

which proves the inequality (2), completes the proof of inequality (1), and completes the proof of the auxiliary statement.

Finally, the same arguments apply between any two consecutive releases.

4.2 Static Slack

In the VBS process model, each process P_i has a bandwidth cap u_i which represents the maximum utilization that any of its actions may have. As a consequence of Lemma 1, if the sum of all bandwidth caps is less than 1, we can safely scale down the processor frequency by a frequency-scaling factor equal to the sum of the bandwidth caps.

Proposition 2. *A set of processes $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$, with a total system utilization*

$$U = \sum_{i \in I} u_i,$$

is schedulable within the response-time bounds if the processor frequency is at least $f_{ref} = U \cdot f_{max}$.

We call f_{ref} the reference frequency. If used, this reference frequency is set once and never changed at run-time.

4.3 Dynamic Slack

Dynamically, at runtime, the operating frequency can sometimes be reduced further than the reference frequency. As stated before, dynamic slack arises out of two VBS properties, hence we distinguish two types of dynamic slack.

- **Termination slack**, resulting from the VBS termination strategy.
- **Action slack**, generated by an action having utilization less than the bandwidth cap of the process.

The termination strategy ensures that if an action finishes its load at some time within a period, the termination is postponed until the end of that period. We compute at every arrival of a new action the termination slack of the action and assign a new limit for the action that represents the minimum time per period that does not cause the action to exceed its upper response time bound.

For example, an action $\alpha = (55, (30, 100))$ could run for 28 time units every period and still meet its response time bound 200, therefore the virtual periodic resource for this action can be changed from $(30, 100)$ to $(28, 100)$ and the resulting slack of 2 time units per period can be used to scale down the processor.

At every time instant t when an action has an arrival the scheduler computes the new limit considering the termination slack for the arriving action as

$$\lambda^*_{i,j} = \left\lceil \frac{l_{i,j}}{n_{i,j}} \right\rceil,$$

where $n_{i,j} = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil$ is the number of periods needed for the action $\alpha_{i,j}$ to finish its load. Note that the new limit never exceeds the old limit ($\lambda^*_{i,j} \leq \lambda_{i,j}$), so this change does not influence schedulability. The action $\alpha_{i,j}$ will thus be transformed into an action $\alpha^*_{i,j} = (l_{i,j}, (\lambda^*_{i,j}, \pi_{i,j}))$.

The action slack arises from newly arriving actions. If the utilization of the new action is lower than the bandwidth cap for the process, dynamic slack is introduced in the system. The scaling factor is computed as the sum of remaining utilizations of the active actions (the current actions of each VBS process). The algorithm is invoked at the time instants when an action has a release. The correctness of the algorithm is ensured by the following corollary of Lemma 1, which holds since between two action releases, even though there are more task releases, the total utilization remains constant.

Proposition 3. *Let $\mathcal{P} = \{P_i(u_i) \mid 1 \leq i \leq n\}$ be a schedulable set of VBS processes, with a total utilization cap $U = \sum_{i \in I} u_i \leq 1$, where $\alpha_{i,j}$ with a virtual periodic resource $(\lambda_{i,j}, \pi_{i,j})$, for $j \geq 0$ are the actions of process P_i. This set of processes is schedulable within the response-time bounds if in between two action releases the processor frequency is at least $f_{new} = U_c \cdot f_{max}$ where $U_c = \sum_{i=1}^{n} \frac{\lambda_{i,j_i}}{\pi_{i,j_i}}$ is the total utilization of all released actions α_{i,j_i} in the considered interval of time between two action releases.*

Note that the two types of slack can be exploited separately or together. Using only the termination slack may reduce the actual response time jitter of the action, while using just the action slack does not modify the original limit of the action. It is only when used together that the minimum possible operating frequency is achieved and CPU utilization is maximized.

4.4 FS-VBS Algorithm

We present the algorithm that computes the minimum frequency at which the processor can run such that all actions meet their response time bounds. The two types of dynamic slack can be used separately or together, we present the combined algorithm in Listing 1.

Algorithm 1 FS-VBS(t)

```
Require: t
 1: $\mathcal{AA}$ = ARRIVAL[t]
 2: for all $\alpha_{i,j} \in \mathcal{AA}$ do
 3:     $n_{i,j} = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil$
 4:     $\lambda_{i,j} = \left\lceil \frac{l_{i,j}}{n_{i,j}} \right\rceil$
 5: end for
 6: $\mathcal{RA}$ = RELEASED[t]
 7: for all $\alpha_{i,j} \in \mathcal{RA}$ do
 8:     $U_c = U_c + \frac{\lambda_{i,j}}{\pi_{i,j}}$
 9: end for
10: return $f_{new} = U_c \cdot f_{max}$
```

The algorithm is invoked at every time instant t given by the release or arrival of an action. The first part of the algorithm (line $1-5$) adjusts the limit of every action taking advantage of the termination slack while the second part (line $6-9$) computes the utilization of all released actions. For the termination slack, we compute the minimum limit per period for all actions that have an arrival at time t such that all actions will still respect their response time bounds. For the action slack, we consider all actions that are released in the system at time t, i.e. the current actions of the processes at time t. In the end, the algorithm computes the minimum operating frequency in relation to the maximum frequency available. We can now safely scale down the processor to f_{new} since we know that the system of VBS-scheduled actions will meet their response time bounds.

5 Look-ahead FS-VBS

Up to this point we maximize CPU utilization by switching to the lowest possible operating frequency such that all actions still respect their response time bounds. This may lead to improvements in power consumption. However, our assumptions are based on a simple power model and disregarded the switching cost (both in power and time). The power consumption of a system is typically non-linear and depends largely on the platform and workload [22]. Switching to the lowest possible frequency by exploiting the static and dynamic slack of VBS processes may not lead to the best possible power savings due to the complex power profile of the system as well as the overhead introduced by the frequency switches [15].

In general, in terms of power consumption, we have to consider the active energy (E_a) consumed by the CPU at different frequencies and the energy usage introduced by the switching of frequencies (E_s) over a time interval Δt, i.e.

$$E = (E_a + E_s) \cdot \Delta t.$$

In most studies, it is assumed, for simplicity, that the active energy consumption E_a is proportional to V^2. More accurate models can be found in [23, 24]. The time overhead generated by the switches could be readily incorporated into the schedulability analysis by accounting for it using an overhead accounting framework, cf [9].

Our analysis now aims at improving power consumption beyond what has been presented in Section 4. Depending on the system model, more energy can be saved by allowing an action more freedom on how the load is executed within the periods, i.e., an action may assign a different limit for every period of its execution as long as the original response time bounds are met. In order to use this freedom, an action must have knowledge of the future.

In the remainder of the paper we concentrate on describing more advanced methods to reduce CPU power consumption while maintaining response times, using future knowledge. First, we present an optimal offline method that computes limits for all period instances of the actions that will result in the best power savings. The energy consumption function can be plugged in depending on the specific power profile of the system. We consider a simple power model in which energy consumption depends on frequency but also on the number of frequency switches and present the optimal offline algorithm for it. We then present an online algorithm that approximates the offline algorithm for that particular power model.

5.1 Optimal Offline FS-VBS

We start from the ideal case, where the sequence of action changes is fully known. Since a process can run for an infinite amount of time, yet the changes of actions are known, we assume that action changes are periodic, i.e., the process can be seen as a finite loop that executes infinitely often.

For any process $P_i(u_i)$ let $\rho_i : \mathbb{N} \to \mathcal{R}$ be a function that keeps trace of the resources used. That is $\rho_i(j) = R_{i,j}$ if and only if $\alpha_{i,j} = (l_{i,j}, R_{i,j})$, i.e., $R_{i,j}$ is the resource used by the action $\alpha_{i,j}$. A process $P_i(u_i)$ is a loop if there exists a number $s_i \in \mathbb{N}$ such that $\rho_i(j + s_i) = \rho_i(j), \forall j \in \mathbb{N}$. This may seem as a serious restriction, but actually it is only used in Equation (3) and (4). Moreover, it is common that processes are actually loops.

Considering an action $\alpha_{i,j}$ that runs (normally, in the standard VBS algorithm) for $n_{i,j} = \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil$ periods, we aim at finding values $\lambda_{i,j,k}$ for $k = 1, \ldots, n_{i,j}$ such that if the action runs for time $\lambda_{i,j,k}$ in the k-th period instance of $\alpha_{i,j}$

then the power consumption, given a certain power model, is minimal. This amounts to a constrained optimization problem for minimizing power consumption. Namely, we look for values $\lambda_{i,j,k}$ minimizing the power consumption function which we denote by

$$F(\lambda_{i,j,k} \mid i \in I, j \geq 0, 1 \leq k \leq n_{i,j}).$$

Note that if we consider $\Pi = \{\pi_{i,j} \mid i \in I, j \geq 0\}\}$ then any interval $[n \cdot gcd(\Pi), (n+1) \cdot gcd(\Pi)]$ for $n \in \mathbb{N}$ is contained in a single period instance of each active action. This means that in such an interval, the total utilization of the system is calculated as

$$\sum_{i \in I} \frac{\lambda_{i,j,k_n}}{\pi_{i,j}},$$

where λ_{i,j,k_n} is the new limit for the period instance $\pi_{i,j}$ that contains the mentioned interval of the active action $\alpha_{i,j}$.

The total power consumption is the sum of the power consumption in each interval $[n \cdot gcd(\Pi), (n+1) \cdot gcd(\Pi)]$ for $n \in \mathbb{N}$, thus we can write that

$$\begin{aligned} &F(\lambda_{i,j,k} \mid i \in I, j \geq 0, 1 \leq k \leq n_{i,j}) = \\ &\quad gcd(\Pi) \sum_{n \in \mathbb{N}} F_n(\lambda_{i,j,k_n} \mid i \in I, j \geq 0). \end{aligned}$$

As in the general model, we can write

$$F_n(\lambda_{i,j,k_n}) = E_{a,n}(\lambda_{i,j,k_n}) + E_{s,n}(\lambda_{i,j,k_n}).$$

By plugging particular power-consumption functions $E_{a,n}$ and $E_{s,n}$, one gets the global power consumption function. We minimize $F(\lambda_{i,j,k})$ subject to the following constraints imposed by the semantics of VBS. For all $\lambda_{i,j,k}$,

$$\sum_{k=1}^{n_{i,j}} \lambda_{i,j,k} \leq \left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \lambda_{i,j},$$

expressing that the sum of $\lambda_{i,j,k}$ for an action $\alpha_{i,j}$ does not exceed the amount of work the action could do in its periods in the standard VBS execution. Next,

$$\sum_{k=1}^{n_{i,j}} \lambda_{i,j,k} \geq l_{i,j},$$

expressing that the action $\alpha_{i,j}$ will execute its entire load $l_{i,j}$. It may be that in a period instance an action does not execute any of its load, yet the new limits can not be negative, $\lambda_{i,j,k} \geq 0$. In order to respect the response-time jitter, we have to ensure that the action will not finish earlier than its lower response time bound. Therefore, in the last period instance, the action must execute for at least

1 time unit and additionally, the action must not complete its entire load in the previous period instances. We express this through the following two constraints

$$\sum_{k=1}^{n_{i,j}-1} \lambda_{i,j,k} < l_{i,j},$$

$$\lambda_{i,j,n_{i,j}} \geq 1.$$

Next, for every interval $[n \cdot gcd(\mathit{\Pi}), (n+1) \cdot gcd(\mathit{\Pi})], n \in \mathbb{N}$, the system may not be over-utilized, namely

$$\sum_{i \in I} \frac{\lambda_{i,j,k_n}}{\pi_{i,j}} \leq 1,$$

where k_n is as before.

As an instantiation, we consider the simplified model in which the active energy consumed is proportional to the square of the voltage and the number of frequency switches. In this model, we reduce the power consumption by reducing the CPU utilization jitter. CPU utilization jitter is the difference between the system utilization at a certain time and a computed average system utilization over the whole life-time of the system.

In order to compute E_a and E_s, we start by computing the upper response-time bound for a loop iteration of process P_i now assumed to be a loop, as

$$b_i^u = \sum_{j=1}^{s_i} b_{i,j}^u. \tag{3}$$

The average utilization u_i^{avg} of process P_i is therefore

$$u_i^{avg} = \frac{\sum_{j=1}^{s_i} \left(\left\lceil \frac{l_{i,j}}{\lambda_{i,j}} \right\rceil \lambda_{i,j} \right)}{b_i^u}. \tag{4}$$

Next, we compute the average utilization of a system of n processes $\{P_1, \ldots, P_n\}$ as the sum of average process utilizations, i.e.,

$$u^{avg} = \sum_{i=1}^{n} u_i^{avg}.$$

The active energy consumption in the interval $[n \cdot gcd(\mathit{\Pi}), (n+1) \cdot gcd(\mathit{\Pi})]$, for $n \in \mathbb{N}$, is proportional to the square of the voltage, thus we try to minimize

$$E_{a,n} = \left(u^{avg} - \sum_{i \in I} \frac{\lambda_{i,j,k_n}}{\pi_{i,j}} \right)^2.$$

Recall that the number of frequency switches over any such interval is at most 1. Hence, the energy consumed by a frequency switch is a constant γ. The function to be minimized is therefore

$$F(\lambda_{i,j,k}) = gcd(\mathit{\Pi}) \sum_{n \in \mathbb{N}} \left(\left(u^{avg} - \sum_{i \in I} \frac{\lambda_{i,j,k_n}}{\pi_{i,j}} \right)^2 + \gamma \right). \tag{5}$$

In real-time systems this solution may not be feasible due to the large number of variables that need to be stored (one for each period instance of each action) and due to the computational complexity of finding the optimal values. We therefore elaborate on an online algorithm that approximates the offline algorithm under the same power-consumption model.

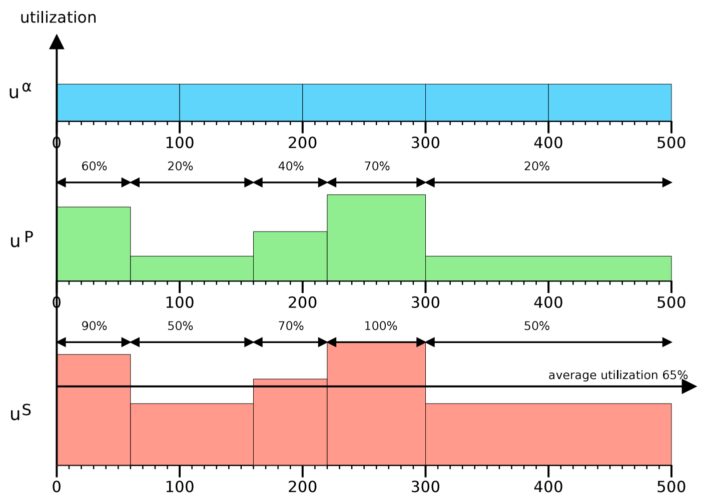

Fig. 1. Utilization jitter with fixed limits for every period of α (standard VBS semantics).

5.2 Look-ahead Online FS-VBS

Consider the example in Figure 1. We only look at one action α of one process with period 100 and utilization 30% (u^{α}). The action needs 5 periods to finish its load of 148. The actions of other processes in the system that run concurrently generate a changing system utilization, u^P in Figure 1, resulting in a total CPU utilization u^S. We only look at one action at a time and we will henceforth refer to it simply as α, dropping the indices i, j, in order to simplify the notation. At time t, when the action α arrives, we calculate the utilization over every interval where the actions of the other processes in the system change, starting from t until the time of termination of α. For the example in Figure 1 we have the intervals of changing CPU utilizations in the system shown in Table 1, where u^{α}, u^P, and u^S are as before. The total system utilization u^S is given by $u^P + u^{\alpha}$.

The algorithm computes the average system utilization over each period instance of action α. Additionally, we also compute the total average system utilization u^{avg} as in Section 5.1. In the k-th period instance of α, let u^e_k denote the utilization error, which is the difference between the average utilization in

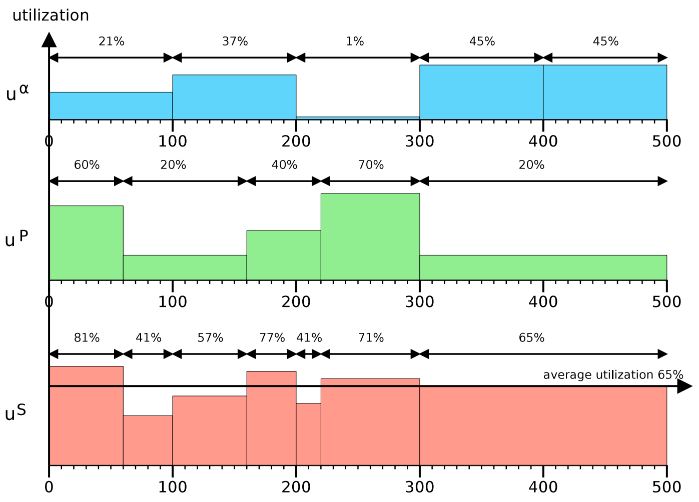

Fig. 2. Reduced utilization jitter with modified limits for each period of α.

Interval	u^α	u^P	u^S
$[0, 60)$	30%	60%	90%
$[60, 160)$	30%	20%	50%
$[160, 220)$	30%	40%	70%
$[220, 300)$	30%	70%	100%
$[300, 500)$	30%	20%	50%

Table 1. Utilization of α, concurrently running processes and total CPU utilization in subintervals of $[0, 500]$.

the k-th period instance and the total average system utilization u^{avg}. The new utilization for the k-the period instance of α is computed as

$$u_k^{\alpha *} = u^\alpha - u_k^e.$$

In the example, the first period instance of α is the interval $[0, 100)$. In this interval the system utilization is 90% from time 0 to 60, and 50% from time 60 to 100. Therefore the average system utilization for this period instance is 74%. Since the $u^{avg} = 65\%$, the utilization error for this period instance is 9%.

Table 2 shows the resulting utilization error and new utilization for every period instance of α.

Ideally, if action α would be modified to have the utilization in each period instance equal to the computed new utilization, the utilization jitter would be minimal. However, there are two issues to be addressed before the action can be changed. One issue is that the response-time bounds of α should not change. By modifying the utilization in each period instance, the limit of the action for each period changes. In the example, the new limits will be $21, 37, 1, 45$, and 45 for

k	Interval	u_k^e	$u_k^{\alpha *}$
0	$[0, 100)$	9%	21%
1	$[100, 200)$	-7%	37%
2	$[200, 300)$	29%	1%
3	$[300, 400)$	-15%	45%
4	$[400, 500)$	-15%	45%

Table 2. Utilization error and new utilization for α in different period instances.

each of the 5 period instances respectively. We have to make sure that the load of the action, which is in this case 148, can be executed with the new limits in the same number of period instances as in the standard VBS algorithm.

We introduce the notion of positive and negative utilization bound. The positive utilization bound δ^+ denotes the maximum amount of utilization that can be added to the action without affecting the lower response-time bound. The negative utilization bound δ^- denotes the maximum amount of utilization that can be subtracted from the action without affecting the upper response-time bound. Thus, the utilization bounds give the amount of error in utilization that can be compensated for without violating the response-time bounds. The utilization bounds are computed as follows, with l, λ, and π being the load, the limit, and the period of the considered action α,

$$\delta^+ = \frac{\lceil \frac{l}{\lambda} \rceil \lambda - l}{\pi},$$

$$\delta^- = \frac{\lfloor \frac{l}{\lambda} \rfloor \lambda - l}{\pi}.$$

Note that if $\lceil \frac{l}{\lambda} \rceil = \lfloor \frac{l}{\lambda} \rfloor$ then $\delta^- = \frac{1-\lambda}{\pi}$. In the example we have $\delta^- = -0.28$ and $\delta^+ = 0.02$.

The utilization bounds can offer a trade-off between performance of the algorithm and temporal isolation of the action. If the utilization bound is set to larger values then the action may be faster than the lower response time bounds or slower than the upper response time bound, increasing the response time jitter by one or more periods, but it can result in overall lower utilization jitter.

We compute the sum of the utilization error over the whole execution time of the action, i.e. $u^{\alpha,e} = \sum_k u_k^e$. In the above example $u^{\alpha,e} = 1\%$, which means that if we were to modify the utilization of the action in each period instance according to the table, we would have an error in the overall utilization of the action of 1% which will be reflected in the response time of the action.

Another issue is that in each period instance, the total system utilization must be lower than or equal to 100%. A change in utilization can occur at any time during a period instance of an action but there is only one limit we can set for a period instance. We address this issue in the algorithm by adding a flag that specifies if at every time instant of every period instance the system is not over-utilized. Only if this is the case, the limits of every period instance of α can be changed according to the computed new utilization.

The look-ahead online algorithm is presented in Listing 2.

Algorithm 2 Look-ahead FS-VBS(t)

```
Require: t
 1: $\mathcal{AA}$ = ARRIVAL[t]
 2: for all $\alpha \in \mathcal{AA}$ do
 3:     $n = \lceil \frac{l}{\lambda} \rceil$
 4:     $c_\alpha = true$
 5:     for $k = 0$ to $n$ do
 6:         $u_k^e = \frac{\sum_{i=k\pi}^{(k+1)\pi} u^S(i)}{\pi} - u^{avg}$
 7:         $u^{\alpha,e} = u^{\alpha,e} + u_k^e$
 8:         for $i = k\pi$ to $(k+1)\pi$ do
 9:             if $u_k^e < u^S(i) - 1$ or $\frac{\lambda}{\pi} - u_k^e > 0$ then
10:                 $c_\alpha = false$
11:             end if
12:         end for
13:     end for
14:     if $\delta^- < u^{\alpha,e} \leq \delta^+$ and $c_\alpha$ then
15:         for $k = 0$ to $n$ do
16:             $\lambda_k^* = \lambda_k - u_k^e \pi$
17:         end for
18:     end if
19: end for
```

For every action α that has an arrival at time t the algorithm starts by calculating total number of period instances (n) the action needs to finish its load (line 3). The flag c_α denotes whether the limits of action α can be changed. In the beginning we set it to *true* (line 4). Next, for each period instance of the action, we compute the utilization error u_k^e for the k-th instance (line 6) and the total utilization error $u^{\alpha,e}$ (line 7). Here, $u^S(i)$ refers to the total system utilization at time i. We check that for each time instant of the k-th period instance the new utilization $\frac{\lambda}{\pi} - u_k^e$ will not be negative or over-utilize the CPU (line 9). If this is the case for any time instant then we set the flag c_α to *false* signaling that the action cannot be changed (line 10). If the total utilization error $u^{\alpha,e}$ is within the computed utilization bounds and the flag is *true* (line 14) we can compute the new limits λ_k^* for the action (line 16).

This method is conservative in that it relies on other actions to reduce the utilization jitter if the current action cannot be changed. More accurate methods to reduce the utilization jitter are subject of future work.

In the example, since at no point the CPU becomes under- or over-utilized and also the total utilization error is within the positive and negative utilization bounds, we can change the limits of the period instances for the action according to Table 2. In Figure 2 the solution is shown where the controlled process from the example modifies its utilization to compensate for the changing CPU utilization thereby reducing the utilization jitter.

Both the optimal offline and the online FS-VBS methods compute values for the limits for each period instance of the actions in the system. At each point in time where the resulting CPU utilization changes we switch to a new frequency given by the frequency scaling factor equal to the sum of the utilizations for the current period instances of the released actions, in accordance to Lemma 1.

6 Experiments

We have conducted a series of experiments, using different simulated processes and actions, that show the effects on the power consumption of exploiting action and termination slack with the FS-VBS algorithm and also using the optimal offline method. We use a simulated DVS-capable platform that has a continuous set of available frequencies and a linear power consumption model, i.e., the relation between the frequency and the corresponding voltage level is linear. The power consumed is proportional to the square of the voltage.

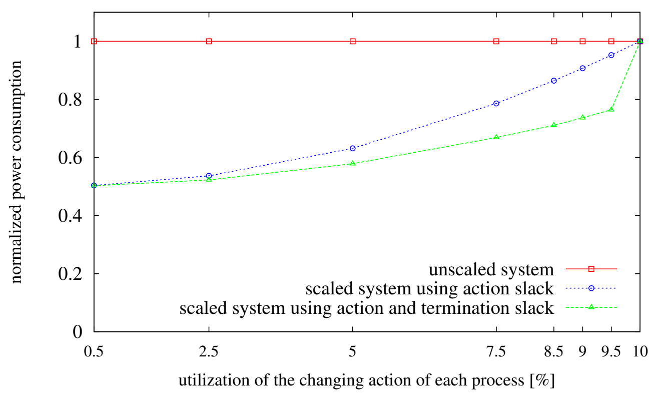

Fig. 3. Normalized power consumption for a system of 10 processes with increasing utilization.

For the first experiment we run 10 processes with two actions each. The first action of each process has a load of 400 and uses the resource $(100, 1000)$. The period of the resource used by the second action is also 1000. We vary the limit and load of the second action as follows: the limit λ is chosen from the set $\{5, 25, 50, 75, 85, 90, 95, 100\}$ and the load l is chosen to be $3\lambda + 1$. This results in an increasing utilization (x-axis) for the second action. In Figure 3 we show the normalized power consumption (y-axis) using just action slack and using both action and termination slack in relation to the power consumption of an unscaled system.

In the second experiment (Figure 4) we show the normalized power consumption (left y-axis) for 10 sets of random processes and actions. The right y-axis

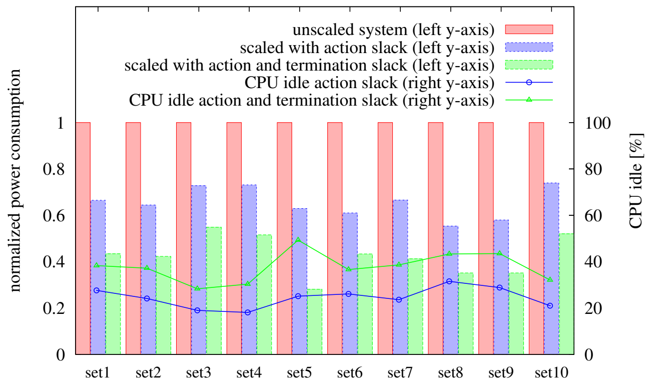

Fig. 4. Normalized power consumption for 10 sets of random processes.

shows the CPU idle percentage when using just action slack and then both action and termination slack. As expected, there is a correlation between CPU idle time and power consumption, i.e., the higher the CPU idle time the lower the power consumption. Compared to using just the action slack, more CPU idle time is generated by exploiting also the termination slack thus reducing power consumption even further.

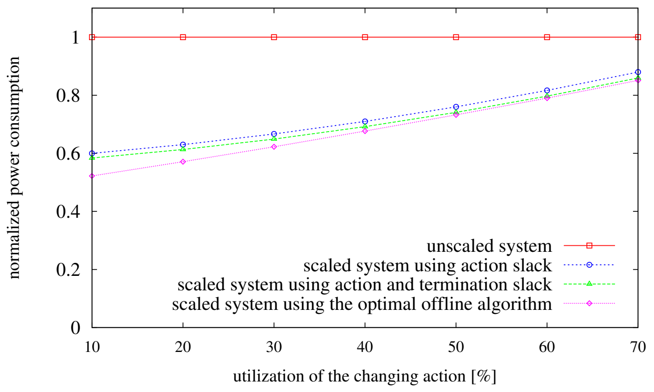

Fig. 5. Normalized power consumption for a system of 2 processes with increasing utilization.

The third experiment (Figure 5) shows the normalized power consumption (y-axis) using the optimal offline method compared to an unscaled system, a scaled system using action slack, and a scaled system using both action and termination slack. There are two processes in the system, process $P_1(30\%)$ has one action $\alpha_{1,1} = (2600, (300, 1000))$ and process $P_2(70\%)$ has three actions, namely: $\alpha_{2,1} = (1500, (1500, 3000))$, $\alpha_{2,2} = (2100, (2100, 3000))$, and $\alpha_{2,3} = (l_{2,3}, (\lambda_{2,3}, 3000))$. The limit of $\alpha_{2,3}$, $\lambda_{2,3}$, is chosen from the set $\{300, 600, 900, 1200, 1500, 1800, 2100\}$ which results in an increasing utilization for the action (x-axis). The load of the action is equal to its limit, i.e., $l_{2,3} = \lambda_{2,3}$, such that the action executes for only one period. We used AMPL/CPLEX [10] to find the optimal configuration of the limits of action $\alpha_{1,1}$ such that the objective function described in Equation (5) is minimized. Note that using the online algorithm for this experiment results in the same limits for the period instances of action $\alpha_{1,1}$ and therefore the same power savings as with the optimal offline algorithm.

7 Conclusion and Future Work

We presented methods that may reduce CPU power consumption with variable-bandwidth servers while maintaining temporal isolation of concurrently running processes. We have shown that, in the presence of CPU slack, frequency scaling of EDF-scheduled, possibly non-periodically arriving tasks (such as the tasks from the actions of VBS processes) is safe up to full CPU utilization and proposed a frequency-scaling VBS algorithm that exploits CPU slack to minimize operating frequencies with maximal CPU utilization while maintaining temporal isolation. Furthermore, we have shown that, given knowledge of future events, further reductions in CPU power consumption may be possible by allowing the scheduler to redistribute computation time of process actions among the server periods during which the actions execute without affecting the actions' original response time bounds. We presented an optimal offline algorithm that minimizes a given power-consumption function and an online algorithm that is feasible for real-time systems and approximates the offline algorithm with a simplified CPU power profile.

As future work we aim at improving and implementing the online algorithm in a real system and comparing it to the optimal offline algorithm using real and simulated workloads. An issue that also needs to be addressed is the change in response time bounds in a real system with discrete frequency levels. As discussed before, our analysis so far relies on a theoretical model where there are infinitely many available frequency levels. Switching to the nearest frequency (either smaller or greater) will potentially result in being faster or slower than the lower or upper response time bounds, respectively. We therefore plan to extend the existing schedulability analysis with response time bounds that take into account discrete frequency levels.

References

1. L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Journal of Real-Time Systems*, 27(2):123–167, 2004.

2. H. Aydin, P. Mejía-Alvarez, D. Mossé, and R. Melhem. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. RTSS*, page 95. IEEE Computer Society, 2001.
3. A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau. Profile-based dynamic voltage scheduling using program checkpoints. In *Proc. DATE*, page 168. IEEE Computer Society, 2002.
4. S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson. Dynamic integrated scheduling of hard real-time, soft real-time and non-real-time processes. In *Proc. RTSS*. IEEE, 2003.
5. T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. HICSS*, page 288. IEEE Computer Society, 1995.
6. G. C. Buttazzo, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51:289–302, 2002.
7. S. S. Craciunas, C. M. Kirsch, H. Payer, H. Röck, and A. Sokolova. Programmable temporal isolation through variable-bandwidth servers. In *Proc. SIES*. IEEE Computer Society, 2009.
8. S. S. Craciunas, C. M. Kirsch, H. Röck, and A. Sokolova. Real-time scheduling for workload-oriented programming. Technical Report 2008-02, University of Salzburg, September 2008.
9. S.S. Craciunas, C.M. Kirsch, and A. Sokolova. Response time versus utilization in scheduler overhead accounting. In *In Proc. RTAS*. IEEE, 2010.
10. R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Manage. Sci.*, 36(5):519–554, 1990.
11. W. Kim, J. Kim, and S. Min. A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Proc. DATE*, page 788. IEEE Computer Society, 2002.
12. C. M. Krishna and Y. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. *IEEE Trans. Comput.*, 52(12):1586–1593, 2003.
13. M. P. Lawitzky, D. C. Snowdon, and S. M. Petters. Integrating real time and power management in a real system. In *Proc. OSPERT*. IEEE Computer Society, 2008.
14. M. Marinoni and G. Buttazzo. Balancing energy vs. performance in processors with discretevoltage/frequency modes. In *Proc. RTCSA*, pages 294–304, 2006.
15. A. Miyoshi, C. Lefurgy, E. Van Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *Proc. ICS*, pages 35–44. ACM, 2002.
16. P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. SOSP*, pages 89–102. ACM, 2001.
17. A. Qadi, S. Goddard, and S. Farritor. A dynamic voltage scaling algorithm for sporadic tasks. In *Proc. RTSS*, page 52. IEEE Computer Society, 2003.
18. C. Scordino and G. Lipari. Using resource reservation techniques for power-aware scheduling. In *Proc. EMSOFT*, pages 16–25. ACM, 2004.
19. D. Shin and J. Kim. Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems. In *Proc. ASP-DAC*, pages 653–658. IEEE Press, 2004.
20. I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. RTSS*, pages 2–14. IEEE Computer Society, 2003.
21. Y. Shin, K. Choi, and T. Sakurai. Power optimization of real-time embedded systems on variable speed processors. In *Proc. ICCAD*, pages 365–368. IEEE Press, 2000.

22. D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser. Koala: a platform for os-level power management. In *Proc. EuroSys*, pages 289–302. ACM, 2009.
23. D. C. Snowdon, S. M. Petters, and G. Heiser. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *Proc. EMSOFT*, pages 84–93. ACM, 2007.
24. D. C. Snowdon, G. van der Linden, S. M. Petters, and G. Heiser. Accurate run-time prediction of performance degradation under frequency scaling. In *Proc. OSPERT*, 2007.