Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems

Wanghong Yuan
Department of Computer Science
University of Illinois at Urban-Champaign
Urbana, IL 61801, USA
1-217-333-1515
wyuan1@cs.uiuc.edu

Klara Nahrstedt
Department of Computer Science
University of Illinois at Urban-Champaign
Urbana, IL 61801, USA
1-217-244-6624
klara@cs.uiuc.edu

ABSTRACT

Battery-powered mobile devices are becoming increasingly important computing platforms, which require low energy consumption while meeting the resource demands of a dynamic application workload. Most proposed dynamic voltage scaling (DVS) algorithms, targeting either best-effort or hard real-time systems, however, cannot be directly applied to such open mobile systems. This paper presents a framework to integrate DVS into soft real-time (SRT) scheduling for open mobile systems, achieving energy saving of DVS while preserving resource guarantees of SRT scheduling. The integrated framework makes three major contributions. First, multimedia applications reserve resource based on their average resource usage, without the knowledge of worst-case execution time, which is difficult to estimate in an open mobile environment. Second, the SRT scheduling ensures the correctness of reservation admission and enforcement in a variable speed context. Finally, the DVS manager reduces the processor energy consumption by utilizing the unallocated resource, reclaiming the allocated but unused resource, or avoiding the unused resource. Our extensive simulation results demonstrate that our framework is able to save 4% to 32% energy while slightly affecting application performance.

Categories and Subject Descriptors

D.4.1 [**Process Management**]: Scheduling; D.4.7 [**Organization and Design**]: Real-time systems and embedded systems

General Terms

Design, Algorithms

Keywords

Multimedia, Scheduling, Power Management

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NOSSDAV'02, May 12-14, 2002, Miami, Florida, USA.
Copyright 2002 ACM 1-58113-512-2/02/0005…$5.00.

1. INTRODUCTION

Open mobile systems are becoming increasingly an integral part of ubiquitous computing environment, where a dynamic mix of soft real-time (SRT) multimedia and best-effort (BE) applications run on mobile devices. On one hand, these mobile devices need powerful processors, with high energy consumption, to satisfy the demanding resource requirements of multimedia applications. On the other hand, however, they are typically battery-powered with a restricted battery energy budget, thus requiring a low energy consumption to maximize the battery lifetime. Hence, the operating system of such devices needs to manage CPU resource and energy in a coordinated way, so that the processor consumes as little energy as possible while meeting application resource demands.

Dynamic voltage scaling (DVS) is typically used to reduce the processor energy consumption by dynamically adjusting the speed, and hence power, according to application workload [7][18][19][24][25]. The workload is either predicted using some heuristics or estimated from applications' worst-case execution time (WCET). The DVS algorithms are shown to be effective for general-purpose best-effort systems [13][18][24], where an inaccurate workload prediction will not significantly affect application performance, and hard real-time systems [19][20][25], where a precise estimation of WCETs is available. These algorithms, however, cannot be directly applied to open mobile systems for the following reasons. First, the heuristic prediction is typically inaccurate, and hence may violate the timing constraints of multimedia applications. Second, it is difficult to precisely estimate WCETs, because of the uncertainty resulted from several low-level mechanisms in open systems, e.g., caching and interrupts. Third, even with a precise estimation, the WCET-based speed setting would be inefficient, because multimedia applications typically present a high variation of execution time, thus with an average much lower than the worst-case. Finally, the dynamic speed setting also challenges resource reservation and SRT scheduling mechanisms for multimedia applications, because they typically assume a constant processor speed and do not address, for example, how to admit and enforce reservations in a DVS context [11][16][21][23].

In this paper, we propose a framework to integrate dynamic voltage scaling into soft real-time scheduling for open mobile systems, achieving energy saving of DVS while preserving performance guarantees of SRT scheduling. The SRT scheduling enables each multimedia application, as well as all BE applications as a whole, to reserve CPU resource and enforces reservations in a variable speed context. The DVS mechanism is integrated into the SRT scheduling, hence called *SRT-DVS*, to slow down the processor as much as possible to save energy while meeting application resource requirements. Our integrated SRT-DVS framework makes three major contributions. First, multimedia applications make resource reservation based on their average CPU usage, thus removing the need of knowing WCETs. Overruns resulted from the average-based reservation are protected and handled in a bounded time. Second, we build a variable speed constant bandwidth server (VS-CBS) algorithm, based on CBS [1], to ensure the correctness of reservation admission and enforcement, thereby delivering resource guarantees to applications, in a variable-speed context. Finally, the SRT-DVS algorithms reduce the processor power by utilizing the unallocated resource, reclaiming the allocated but unused resource, or adapting reservation to avoid the unused resource.

The rest of the paper is organized as follows. First, section 2 introduces the SRT scheduling. Section 3 describes three integrated SRT-DVS algorithms. Section 4 presents the simulation evaluation. Finally, section 5 concludes the paper.

2. SOFT REAL-TIME SCHEDULING

In this section, we first introduce the workload model and assumptions, and then present the SRT scheduling scheme, followed by an analysis of the SRT scheduling.

2.1 Workload Model and Assumptions

The typical workload in open mobile systems includes multimedia and traditional best-effort tasks, which can be dynamically added and removed from the system. Multimedia tasks usually present soft real-time constraints, and hence are called *SRT tasks*. Each task τ_i (i = 0, 1, …) consists of a sequence of jobs $J_{i,k}$ (k = 1, 2, …), where $J_{i,k}$ denotes the k^{th} job of task τ_i. Each job $J_{i,k}$ is characterized by an arrival time $r_{i,k}$, a finishing time $f_{i,k}$, and an execution time $e_{i,k}$ at the highest CPU speed. Each SRT task τ_i is characterized with two additional parameters (E_i^{avg}, P_i), where E_i^{avg} denotes the average execution time at the highest speed, achievable through probing or profiling, and P_i denotes the desired activation period between successive jobs. Each SRT job $J_{i,k}$ has a soft deadline $d_{i,k} = r_{i,k} + P_i$.

Throughout this paper, we make three assumptions: (1) SRT tasks are independent of each other and BE tasks, and the scheduling overhead is negligible. (2) The processor is able to operate with a discrete speed set $S = \{s_1, s_2, \ldots, s_m \mid s_1 < s_2 < \ldots < s_m = 1\}$, where s_i ($1 \le i \le m$) is normalized as the ratio to the highest speed, and thus represents the relative performance. For example, the (normalized) speed set of a processor is {3/5, 4/5, 1}, if the processor can run with three frequencies: 300MHz, 400MHz, and 500MHz. (3) When speed is scaled to s, the execution time of a job is scaled by a factor $1/s$, while the soft deadline, if any, remains unchanged.

2.2 Scheduling Scheme

We first introduce the variable speed constant bandwidth server (VS-CBS), which is extended from the CBS algorithm [1] to be applied in an energy-aware context. A VS-CBS is characterized by two parameters (Q, T), where Q and T are the maximum budget and period of the server, respectively. The ratio $U = Q/T$ denotes the server bandwidth. At each instant, a deadline d_k and a budget c are associated with the server. When a served job executes for t units of time at speed s, the budget c is decreased by $t{\cdot}s$. Whenever c is decreased to 0, the budget is recharged to Q and a new deadline is generated as $d_{k+1} = d_k + T$. Each server has a ready queue containing jobs ready to be executed, and is in one of three states: *idle*, *active*, or *serving* at any instant. A server is *idle* if its ready queue is empty. With a non-empty ready queue, a server is in *serving* state when it is scheduled to execute a job, or in *active* state otherwise. Figure 1 shows the finite state automata of a VS-CBS server.

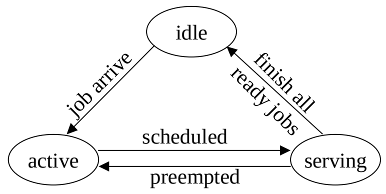

Figure 1. Finite state automata of a VS-CBS server

Similar to [6], we use a two-level hierarchical scheduling scheme, as shown in Figure 2. Each SRT task τ_i (i ≥ 1) reserves E_i^{avg} units of time every period P_i, and is served by a dedicated server *VS-CBS*$_i$ ($Q_i = E_i^{avg}$, $T_i = P_i$). All BE tasks are served together by a shared server *VS-CBS*$_0$ (Q_0, T_0), implicitly reserving Q_0 units of time every T_0 together. Q_0 and T_0 can be static or dynamic depending on the SRT-DVS algorithm, as illustrated in Section 3. The SRT scheduler maintains all VS-CBSs and schedules them on an earliest deadline first (EDF) basis, i.e., always selecting an active server with the earliest deadline. If a *VS-CBS*$_i$ (i ≥ 1) is selected, it executes jobs within task τ_i on a first come first service (FCFS) order. In contrast, *VS-CBS*$_0$ executes BE jobs on a round-robin (RR) basis. This paper does not address the details of RR scheduling of BE jobs, thereby not differentiating individual BE jobs, unless otherwise stated.

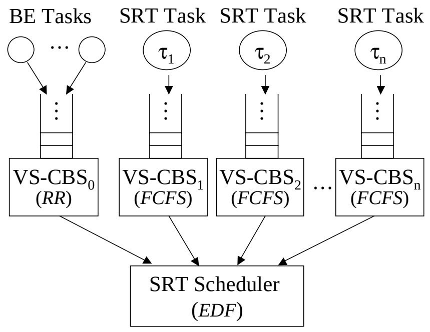

Figure 2. Two-level hierarchical soft real-time scheduling

Figure 3 illustrates a SRT scheduling example, where all BE tasks and SRT task τ_1 are served by *VS-CBS*$_0$ (Q_0 = 2, T_0 = 7) and *VS-CBS*$_1$ (Q_1 = 2, T_1 = 5), respectively. The processor speed is set to 0.5 and 1, when BE jobs and SRT jobs are executed, respectively. Initially at time 0, only *VS-CBS*$_0$ is active with c_0 = 2 and $d_{0,1}$ = 7. Therefore, BE jobs are executed. At time 2 when job $J_{1,1}$ arrives, c_1 is charged to 2 and a new deadline is generated as $d_{1,1}$ = 5, earlier than $d_{0,1}$, so $J_{1,1}$ is executed until time 4. At that time, the SRT scheduler preempts *VS-CBS*$_1$ (and hence $J_{1,1}$), because it generates a new deadline $d_{1,2}$ = 10, later than $d_{0,1}$.

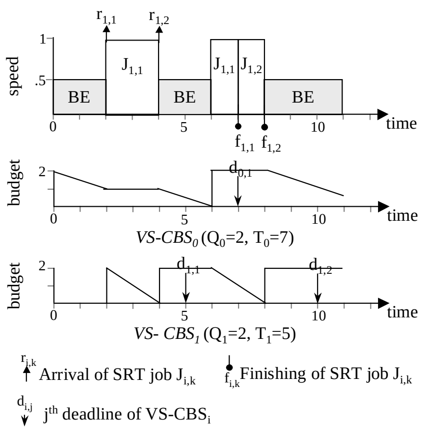

Figure 3. A SRT scheduling example

2.3 Analysis of SRT Scheduling

The SRT scheduling scheme presents two properties: (1) overrun protection and timely handling for SRT jobs, and (2) statistical performance guarantees to BE jobs, illustrated as follows.

The average-based resource reservation may cause a job to overrun, i.e., executing for more than its reserved average time. A job will overrun, for example, if its actual execution time is 5ms while the average is 4ms. A VS-CBS server is able to protect overrun, thereby preventing an overrun job from interfering with the execution of other tasks. Intuitively, when a job overruns, the budget of the serving VS-CBS becomes 0 and a new deadline is generated. Therefore, the serving VS-CBS may be preempted without the earliest deadline, thus preempting the overrun job. On the other hand, a VS-CBS server can handle an overrun in a bounded time as long as $\sum_{i=0}^{n} U_i \le 1$. Assume job $J_{i,k}$ is preempted at time t^* for overrun protection and the overrun part is $o_{i,k}$, in terms of time units at the highest speed. Whenever selected by the SRT scheduler, *VS-CBS$_i$* executes $J_{i,k}$'s overrun part until $J_{i,k}$ finishes. To complete the overrun part, *VS-CBS$_i$* needs to consume its budget by an amount of $o_{i,k}$, as explained in Section 2.2. This consumption takes at most $\lceil o_{i,k} / Q_i \rceil$ period (i.e., time of $\lceil o_{i,k} / Q_i \rceil * T_i$), since Q_i amount of budget can be consumed every period. Therefore, job $J_{i,k}$ will finish its overrun part no later than $t^* + \lceil o_{i,k} / Q_i \rceil * T_i$.

For the example in Figure 3, job $J_{1,1}$ overruns and is preempted at time 4 when the budget of *VS-CBS$_1$* is decreased to 0. $J_{1,1}$ is executed again at time 6, and completes its overrun part (equal to 1 time unit) at time $7 \le 4 + \lceil 1/2 \rceil * 5 = 9$.

Moreover, the SRT scheduling also delivers statistical performance guarantees to BE jobs, whose execution time cannot be easily bounded. Assume BE jobs arrive according to a Poisson process with rate λ, and their execution times have an exponential distribution with parameter μ. That is, $P\{r_{k+1} - r_k \le \delta\} = 1 - e^{-\lambda\delta}$, $\delta \ge 0$; and $P\{e_k \le \phi\} = 1 - e^{-\mu\phi}$, $\phi \ge 0$, where r_k and e_k denote the arrival time and execution time of the k^{th} BE job J_k, respectively. Since *VS-CBS$_0$* serves BE jobs on a RR basis, we can model it as a *M/M/1/PS* queue [4] with parameters λ' and μ'. In this M/M/1/PS queue, the arrival time of each job remains unchanged, and thus the arrival intervals have the same probability distribution, i.e., $\lambda' = \lambda$. However, the execution time of each job is extended by a factor $1/U_0$, i.e., $e_k' = e_k / U_0$, because *VS-CBS$_0$* serves only Q_0 units of time every T_0, thus averagely executing $U_0 = Q_0/T_0$ amount of BE jobs every time unit. From $P\{e_k' \le \phi\} = P\{e_k / U_0 \le \phi\} = 1 - e^{-\mu' \phi}$, we can easily deduce that $\mu' = \mu / U_0$. Applying the properties of M/M/1/PS queue [4], the mean response time (including waiting and serving time) of job J_k with execution time e_k is $E[T(e_k)] = e_k/(1-\rho)$, when $\rho = \lambda'/\mu' = \lambda U_0 / \mu < 1$.

3. INTEGRATED SRT-DVS ALGORITHMS

To provide energy-saving capability for open mobile systems with the mix of SRT and BE tasks, we have developed three SRT-DVS algorithms. These algorithms integrate DVS mechanisms into the above SRT scheduling, achieving the energy saving of DVS while preserving the performance guarantees of SRT scheduling.

3.1 Utilization-based SRT-DVS

We first introduce a very simple SRT-DVS algorithm, which takes advantage of the unallocated resource (bandwidth) of the processor. There is some amount of unallocated bandwidth because the processor utilization is usually less than 100% even if all tasks run at their reserved execution time[1]. The unallocated bandwidth may eventually idle the processor, and hence cause energy wasting. This algorithm slows down the processor to avoid idle intervals resulted from the unallocated bandwidth.

The *processor utilization U* is defined as the sum of bandwidth of all VS-CBSs, i.e., $U = \sum_{i=0}^{n} U_i$, where *VS-CBS$_0$* serves BE tasks and each *VS-CBS$_i$* ($1 \le i \le n$) serves SRT task τ_i in the system. Utilization U represents the required percentage of CPU resource, when the processor runs at the highest speed. To meet this requirement, the processor can run at any speed s, whose relative performance is no less than the utilization, i.e., $U \le s$. The utilization-based algorithm (Utilization-SRT-DVS), as shown in Figure 4, selects the lowest speed s that can meet the utilization requirement, i.e., $\min_{s \in S}\{U \le s\}$, whenever the utilization changes upon a reservation request or release.

```
1. Initially, create VS-CBS_0 U_0 for all BE tasks, and set $U = U_0$.
   Set speed to $\min_{s \in S}\{U \le s\}$.
2. When a SRT task $\tau_i$ requests reservation with $(E_i^{avg}, P_i)$, if $U + E_i^{avg}/P_i > 1$, reject $\tau_i$, otherwise, admit $\tau_i$,
     2.1 Create VS-CBS_i with $(Q_i = E_i^{avg}, T_i = P_i)$ to serve $\tau_i$
     2.2 Increase $U$ by $U_i = Q_i / T_i$
     2.3 Set speed to $\min_{s \in S}\{U \le s\}$.
3. When a SRT task $\tau_i$ with $(E_i^{avg}, P_i)$ leaves the system
     3.1 Decrease $U$ by $U_i$
     3.2 Delete VS-CBS_i from the system
     3.3 Set speed to $\min_{s \in S}\{U \le s\}$.
```

Figure 4. Utilization-SRT-DVS algorithm

Figure 5 shows an example of the utilization-based SRT-DVS, where the speed set S = {0.25, 0.5, 0.75, 1}. Initially at time 0, there are only BE tasks served by *VS-CBS$_0$* with (Q_0 = 2, T_0 = 10)

[1] Note that all BE tasks implicitly reserve Q_0 units of time every period T_0 as a whole.

and no SRT tasks. The utilization $U = U_0 = 0.2$, and the speed is set to 0.25. At time 2 when SRT task τ_1 is admitted with reservation ($E_1^{avg} = 1$, $P_1 = 4$), $VS\text{-}CBS_1$ is created with ($Q_1 = 1$, $T_1 = 4$). The utilization $U = U_0 + U_1 = 0.45$, so the speed is set to 0.5. Similarly, the speed is set to 0.75 at time 3 when τ_2 is admitted with ($E_2^{avg} = 1$, $P_2 = 5$) and U is updated to 0.65. At time 6.33 when τ_1 leaves from the system, U is updated to 0.4, and the speed is set to 0.5.

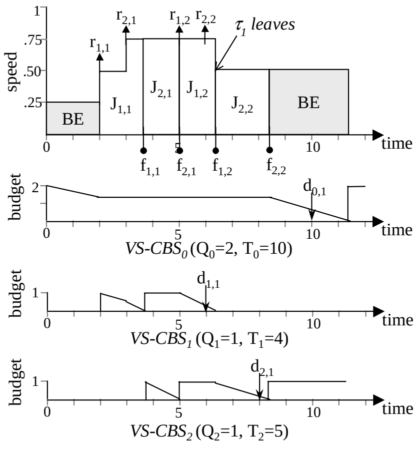

Figure 5. Example of Utilization-SRT-DVS

3.2 Bandwidth reclamation SRT-DVS

The Utilization-SRT-DVS algorithm discussed above exploits the unallocated bandwidth, when the processor utilization is less than 100%. However, it may not realize the full potential of energy saving, because it does not deal with the workload-variation slack, which happens when a job uses less than its reserved time. Although, similar to a CBS [1], a VS-CBS automatically reclaims any residual time resulted from early completion, the accumulated residual time may commonly cause a VS-CBS to idle with unused budget. The unused budget may be eventually wasted and idle the processor, thus wasting energy. Therefore, we propose a bandwidth reclamation SRT-DVS (Reclamation-SRT-DVS) algorithm to save more energy by efficiently reclaiming the reclaimable bandwidth, defined as follows.

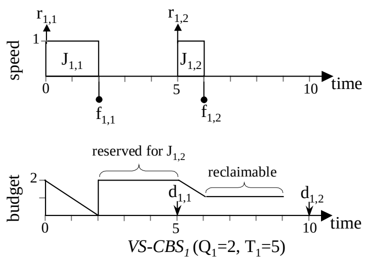

Figure 6. Reclaimable and un-reclaimable unused bandwidth

When a $VS\text{-}CBS_i$ ($i \geq 1$) becomes idle with a remaining budget c_i and deadline $d_{i,k}$, the reclaimable bandwidth u_i equals to c_i / T_i if CBS_i has completed job $J_{i,k}$ and 0 otherwise. The reclaimable bandwidth u_0 of $VS\text{-}CBS_0$ is defined as c_0/T_0 whenever it becomes idle, since the served BE jobs have no deadlines. From the above definition, we can see that the idleness of a $VS\text{-}CBS_i$ ($i \geq 1$) does not follow that the remaining budget, and hence bandwidth, is reclaimable, because it may be reserved for late jobs. Loosely speaking, the budget of a $VS\text{-}CBS_i$ between deadline $d_{i,k-1}$ and $d_{i,k}$ is essentially reserved for job $J_{i,k}$, and can be safely reclaimed only after having completed $J_{i,k}$. Figure 6 shows an example of reclaimable and un-reclaimable bandwidth up to time 9, where $VS\text{-}CBS_1$ is idle between 2 and 5, as well as between 6 and 9. The remaining bandwidth between 2 and 5 is reserved for job $J_{1,2}$ and hence un-reclaimable, while that between 6 and 9 is reclaimable after $J_{1,2}$ has completed.

The Reclamation-SRT-DVS algorithm, as shown in Figure 7, attempts to achieve more energy saving by reclaiming the unused bandwidth of an idle VS-CBS to slow the processor more. This algorithm adjusts the speed again to return the reclaimed bandwidth, when the server becomes serving. Note that the SRT scheduler should not take advantage of the reclaimed bandwidth during admission control, since the reclaimed bandwidth has already been reserved by a SRT task (or BE tasks as a whole). Otherwise, the system may be overloaded. For example, assume $VS\text{-}CBS_0$ becomes idle with $u_0 = 0.1$ when the current utilization $U = 0.95$. If the SRT scheduler admitted a SRT task with $E^{avg}/P = 0.1$ by using condition $U - u_0 + E^{avg}/P = 0.95 < 1$, the system would be overloaded with utilization $U = 0.95 + 0.1 = 1.05 > 1$, when $VS\text{-}CBS_0$ becomes serving later. Therefore, in this algorithm, we use a variable U^* to keep track of the amount of reclaimed bandwidth, instead of directly modifying utilization U, when adjusting the speed.

```
1. Initially, set $U^* = 0$.
2. Upon admission control
    2.1 Invoke Utilization-SRT-DVS algorithm
    2.2 Set speed to $\min_{s \in S}\{U \leq s\}$
3. When $CBS_i$ becomes idle
    3.1 Increase $U^*$ by $u_i$
    3.2 Set speed to $\min_{s \in S}\{U - U^* \leq s\}$
4. When $CBS_i$ becomes serving
    4.1 Decrease $U^*$ by $u_i$
    4.2 Recharge $c_i$ to $Q_i$ and generate a new deadline
    4.3 Set speed to $\min_{s \in S}\{U - U^* \leq s\}$
```

Figure 7. Reclamation-SRT-DVS Algorithm

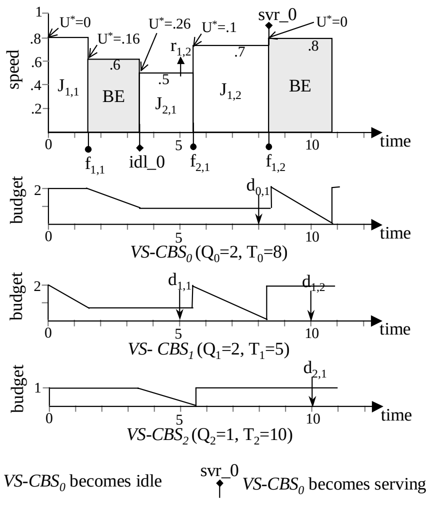

Figure 8. Example of Reclamation-SRT-DVS

Figure 8 illustrates an example of the Reclamation-SRT-DVS algorithm. Initially all VS-CBSs are active with utilization U = 0.75, and the speed is set to 0.8 by the Utilization-SRT-DVS algorithm. At time 1.5 when job $J_{1,1}$ completes early, $VS\text{-}CBS_1$ becomes idle with reclaimable bandwidth u_1 = 0.16, so U^* is increased to 0.16 and the speed is set to 0.6. Similarly, at time 3.5, the speed is set to 0.5 with U^* = 0.26. At time 5.5 when $VS\text{-}CBS_1$ is scheduled to serve job $J_{1,2}$, U^* is decreased to 0.1 and the speed is set to 0.7. Similarly, at time 8.36 when $J_{1,2}$ completes on time and $VS\text{-}CBS_0$ is scheduled with U^* = 0, the speed is set to 0.8.

3.3 Adaptive SRT-DVS

As stated in section 2, we assign the maximum budget and period of $VS\text{-}CBS_i$ (i ≥ 1) with the reservation parameters of the served SRT task τ_i, that is, $Q_i = E_i^{avg}$ and $T_i = P_i$. However, it is difficult to appropriately assign Q_0 and T_0 for $VS\text{-}CBS_0$, since it serves all BE tasks, which do not reserve resource explicitly. The above two SRT-DVS algorithms assume predefined and static parameters (Q_0, T_0), which may overload the system if U_0 is too small, or waste resource and energy if U_0 is too large. Although the Reclamation-SRT-DVS algorithm reclaims the unused bandwidth of $VS\text{-}CBS_0$ if U_0 is larger than the actual BE workload, the processor first runs with a higher speed and only is slowed after $VS\text{-}CBS_0$ becomes idle. Potential energy saving can be achieved if the bandwidth is initially set to a lower value to avoid (or reduce) idle intervals of $VS\text{-}CBS_0$ with unused bandwidth. The final SRT-DVS algorithm that we propose tries to implement the potential energy saving by dynamically adapting the bandwidth of $VS\text{-}CBS_0$, and hence is called Adaptive-SRT-DVS algorithm.

This algorithm, as shown in Figure 9, adapts the bandwidth of $VS\text{-}CBS_0$ by comparing its ready queue length L, in terms of the number of jobs, with a threshold pair (L_{low}, L_{high})[2]. Bandwidth U_0 is increased by the minimum of $1\text{-}U$ and αU_0, if L is greater than L_{high}, and decreased by βU_0 if L is less than L_{low}, where $0<\alpha, \beta<1$. The processor utilization is correspondingly changed whenever U_0 adapts, so admission control is required to ensure $U \leq 1$.

```
1. Initially, set $L = 0$.
2. Upon a BE job arrives, increase $L$ by 1
    2.1 If $L>L_{high}$, increase $U_0$ and $U$ by $min(\alpha U_0, 1–U)$, set $Q_0 =U_0T_0$
    2.2 Set speed to $\min_{s\in S}\{U \leq s\}$
3. Upon a BE job completes, decrease $L$ by 1
    3.1 If $L < L_{low}$, decrease $U_0$ and $U$ by $\beta U_0$, set $Q_0 = U_0T_0$
    3.2 Set speed to $\min_{s\in S}\{U \leq s\}$
```

Figure 9. Adaptive-SRT-DVS Algorithm

The adaptive SRT-DVS algorithm only handles the adaptation of $VS\text{-}CBS_0$ serving BE tasks, and hence should be used together with the bandwidth reclamation SRT-DVS algorithm to achieve more energy saving. A similar adaptation approach can also be applied to $VS\text{-}CBS_i$ (i ≥ 1), when the served SRT task changes its resource requirements. For example, an MPEG video decoding task may significantly vary execution time of frame decoding with scene changes, and hence requires changing its resource reservation. As a result, the server should also adapt its maximum budget and bandwidth correspondingly.

[2] This algorithm would be more efficient if the length L were measured as the sum of execution time of ready jobs. The execution time of a BE job, however, is not known until the job completes. The simple threshold heuristic is adopted here because of its little overhead.

4. SIMULATION

In this section, we evaluate the energy saving and application performance guarantees of the proposed SRT-DVS algorithms. The results are initial, and obtained by simulation.

4.1 Methodology

We developed a simulator to dynamically adjust the processor speed according to each SRT-DVS algorithm, given input of a hardware profile and workload. The hardware profile describes the multiple operating points of the processor, each with a separate speed and power consumption. Unless otherwise specified, we use the following hardware profile {(0.2, 2), (0.4, 3), (0.6, 4), (0.8, 5), (1, 6)}, where 0.2, 0.4, 0.6, 0.8, and 1 denote the processor speeds, and 2, 3, 4, 5, and 6 denote the corresponding power.

Both BE and SRT tasks are dynamically generated for each random interval, uniformly distributed between some ranges. Each BE task has a random execution time uniformly distributed between 4ms and 5ms. The periods of SRT tasks are uniformly distributed between 10ms and 30ms, and average execution time of SRT tasks is a percentage, uniformly distributed between 5% and 15%, of the period. Finally, each SRT task releases a job every period, and the execution time of a job varies around ±10% of the task average execution time. We evaluate four categories of workload, as shown in Table 1, by varying the arrival interval ranges of BE and SRT tasks.

Table 1. Workload categories

Category	*BE arrival interval range*	*SRT arrival interval range*
sparse-sparse	20 – 30ms	300 – 400ms
intensive-sparse	5 – 10ms	300 – 400ms
sparse-intensive	20 – 30ms	100 – 200ms
intensive-intensive	5 – 10ms	100 – 200ms

The primary metrics for our evaluation are energy consumed, deadline miss ratio of SRT tasks, and throughput of BE tasks. BE throughput is defined as the ratio of the total work units[3] of completed BE tasks to the length of simulation time. We also examine another metric, energy per work unit (EPWU), because each SRT-DVS algorithm admits different number of SRT tasks and hence completes different total amount of work during the simulation.

4.2 Results

According to the above metrics, we compare our proposed SRT-DVS algorithms to each other and to a no-DVS algorithm, in which the processor always runs at speed 1. In all simulations, we assume that there is no shutdown of processor and the processor

[3] It is said that the processor completes one unit of work when it executes one time unit at speed 1.

still consumes energy when idling. In our simulations, we set $\alpha = 0.3$, $\beta = 0.2$, $L_{low} = 6$, and $L_{high} = 30$. Unless specified otherwise, the initial bandwidth of $VS\text{-}CBS_0$ is initialized to 0.2 in the following simulations.

4.2.1 Varying workload

In our first experiment, we examine the results of SRT-DVS algorithms for each workload category. Figure 10 (a), (b), (c), and (d) show energy consumed, EPWU, SRT deadline miss ratio, and BE throughput, respectively, for each workload category.

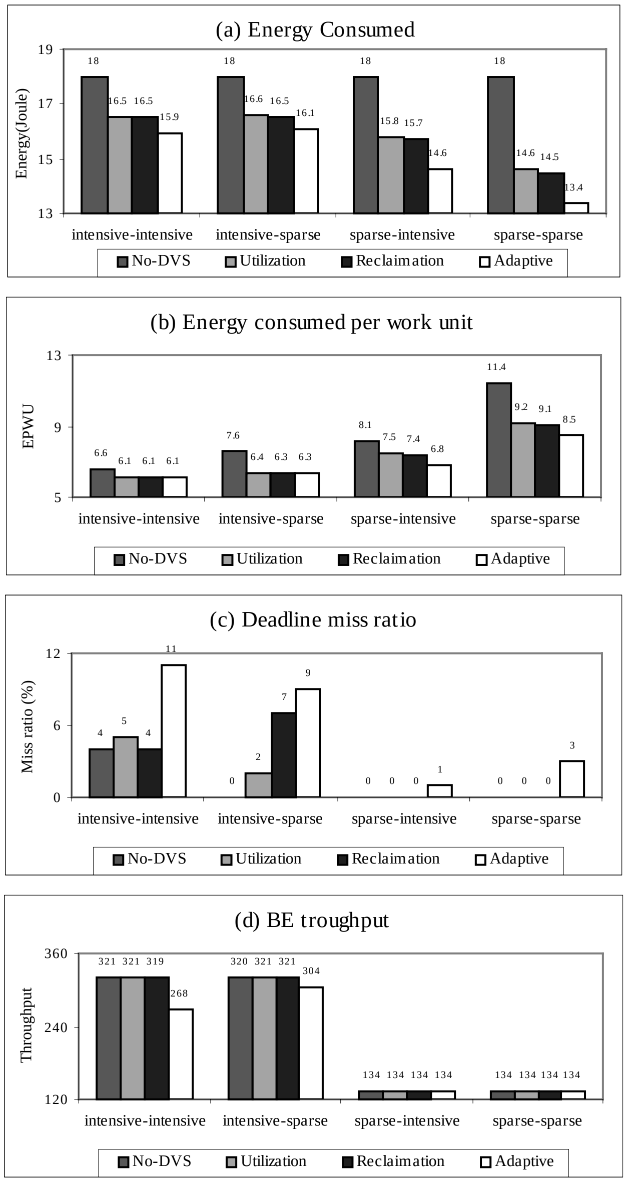

Figure 10. Effects by varying workload category

It is obvious that all SRT-DVS algorithms achieve large energy saving while not significantly degrading application performance. The variation of EPWU, deadline miss ratio, and BE throughput between categories is resulted from the fact that the processor idle time and completed work units are different for each category. In particular, BE throughput is larger when BE workload is intensive, thus completing more BE work during simulation. Similarly, the deadline miss ratios are smaller when workload is sparse, because SRT tasks are able to take advantage of unallocated processor bandwidth[4] for overrun handling, thereby reducing the deadline miss ratio. For each category, however, all SRT-DVS algorithms consume less energy and present lower EPWU than the no-DVS algorithm. The adaptive SRT-DVS algorithm, in particular, saves 26% energy for category sparse-sparse, while maintaining the same BE throughput and increasing the deadline miss ratio by only 3%. Since there is no significant difference of the SRT-DVS behavior when varying workload categories, we will use category sparse-sparse for further experiments.

4.2.2 Average v.s. worst-case based reservation

In the previous simulations, SRT tasks make resource reservation based on their average execution time. We increase the average execution time by 10% as the WCET, since job execution time varies around ±10% of task average execution time.

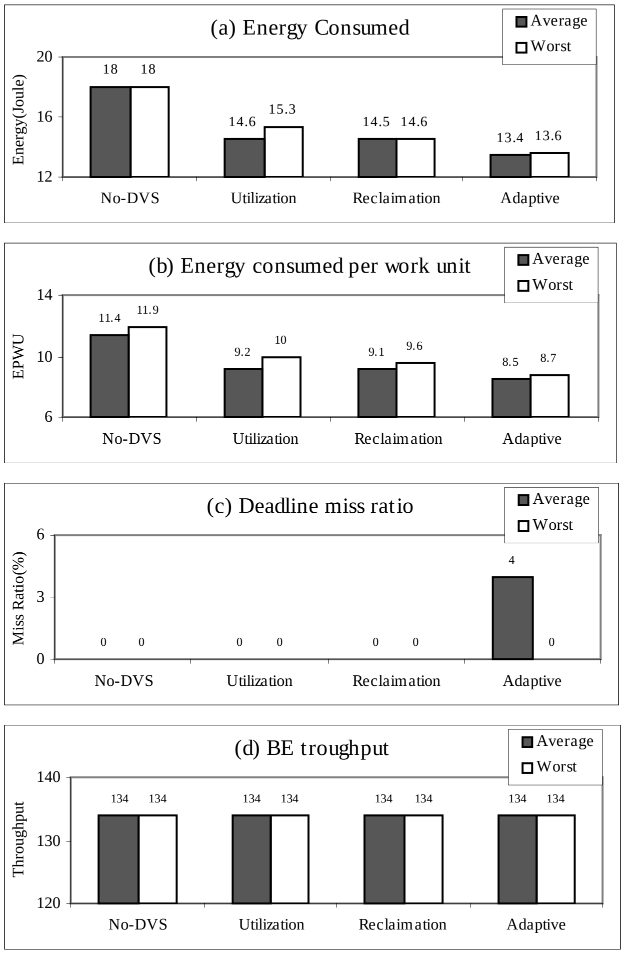

Figure 11. Average vs. worst-case based reservation

[4] There is some unallocated bandwidth with a discrete speed set, because the minimum speed determined by each SRT-DVS algorithm is typically larger than the performance requirements.

Figure 11 shows the differences between average and WCET based reservation. We notice that WCET-based reservation increases the energy consumption and EPWU. In particular, the utilization-based SRT-DVS algorithm consumes 5% more energy, while the reclamation and adaptive algorithms consumes only about 1% more energy. WCET-based reservation wastes more energy with more unused bandwidth, but the reclamation and adaptive algorithms are able to reclaim the unused bandwidth, thus reducing energy wasting.

4.2.3 *Varying bandwidth of VS-VBS$_0$*

Next, we examine the results of SRT-DVS algorithms by varying the initial bandwidth of *VS-CBS$_0$*. Figure 12 shows the results when U_0 is initialized to 0.2, 0.4, and 0.6, respectively.

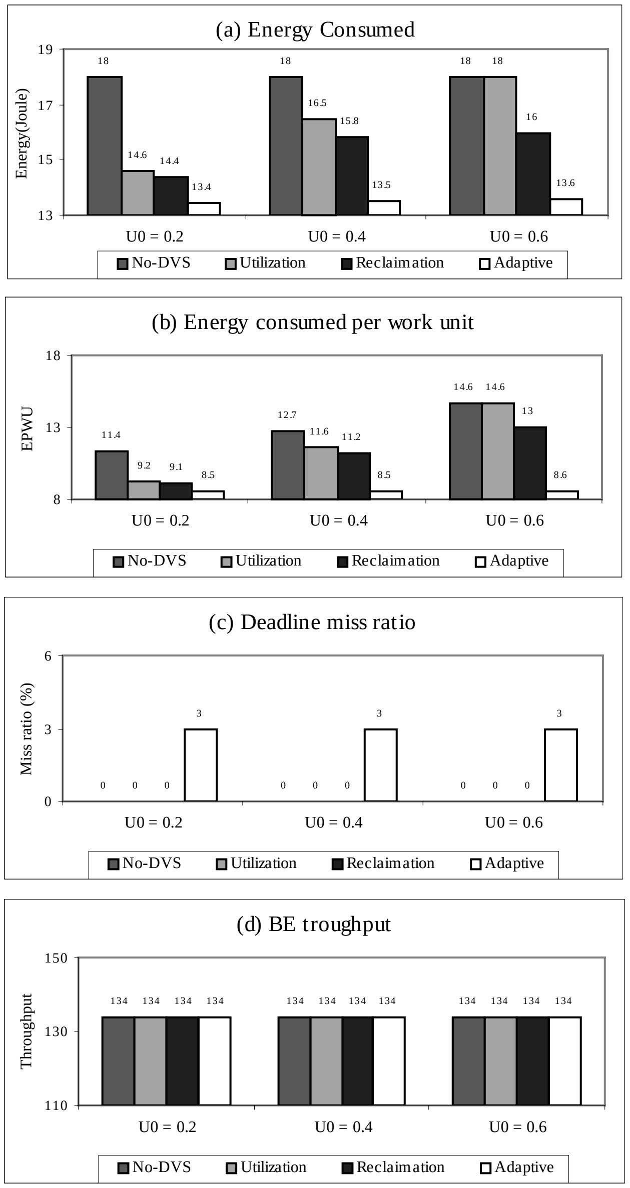

Figure 12. Effects by varying initial bandwidth of VS-CBS$_0$

We notice that the SRT-DVS algorithms save energy while keeping application performance (in terms of SRT deadline miss ratio and BE throughput) same. In particular, the adaptive SRT-DVS algorithm saves about 26% energy, because it efficiently adjusts U_0 to meet the actual BE requirements. The utilization-based algorithm, however, consumes same energy as the no-DVS algorithm when U_0 is initialized to 0.6, since the processor utilization is almost 100% after allocating 60% to BE tasks.

4.2.4 *Varying Hardware profile*

In the forth experiment, we investigate the effects of the SRT-DVS algorithms by varying the hardware profiles as follows:

- Profile 1 = {(0.2, 2), (0.6, 4), (1, 6)}
- Profile 2 = {(0.2, 2), (0.4, 3), (0.6, 4), (0.8, 5), (1, 6)}
- Profile 3 = {(0.1, 1.5), (0.2, 2), (0.3, 2.5), … (0.9, 5.5), (1, 6)}
- Profile 4 = {(0.01, 1.05), (0.02, 1.1), … (0.99, 5.95), (1, 6)}

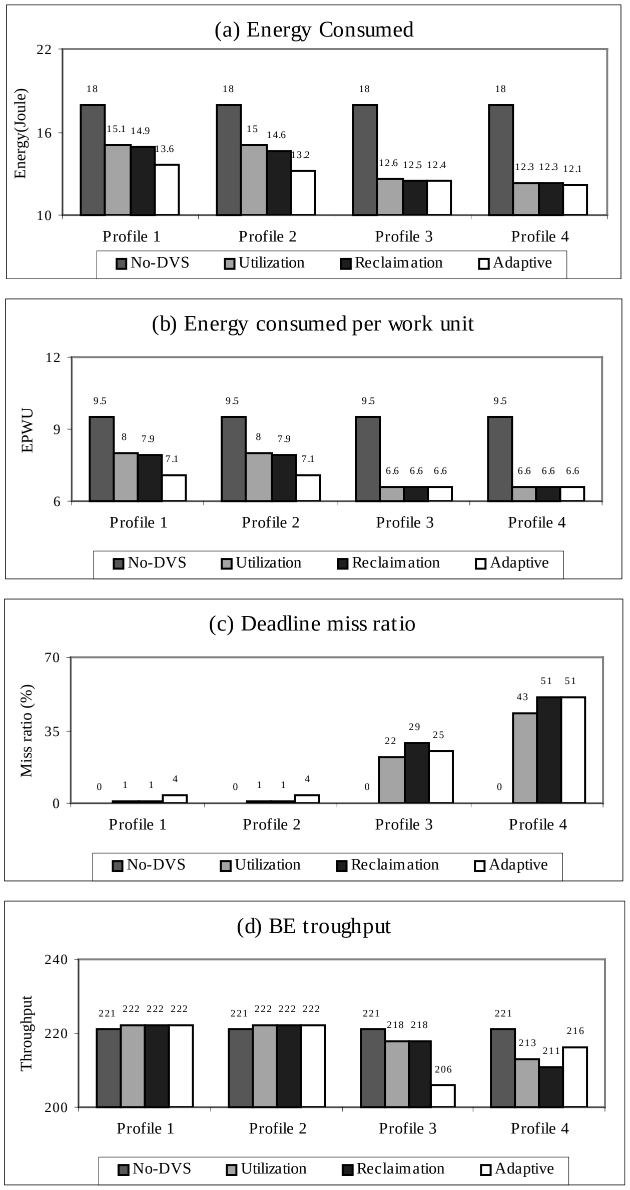

Figure 13. Effects by varying hardware profile

Figure 13 shows the results. It is obvious that the SRT-DVS algorithms save more energy when there is more operating points to select from. In particular, profile 3 and 4 saves about 32% energy, because they present speed options more close to the actual resource requirement, thus reducing as much unallocated bandwidth as possible. On the other hand, however, the deadline miss ratios are much higher in profile 3 and 4, because there is less unallocated background bandwidth for overrun handling.

4.2.5 Varying Processor Utilization

All the previous experiments assume that the scheduler tries to admit as many SRT tasks as possible, thus keeping the processor utilization close to 100%. In our final experiment, we examine the effects of SRT-DVS algorithms by varying the boundary of processor utilization, thereby limiting the number of admitted SRT tasks.

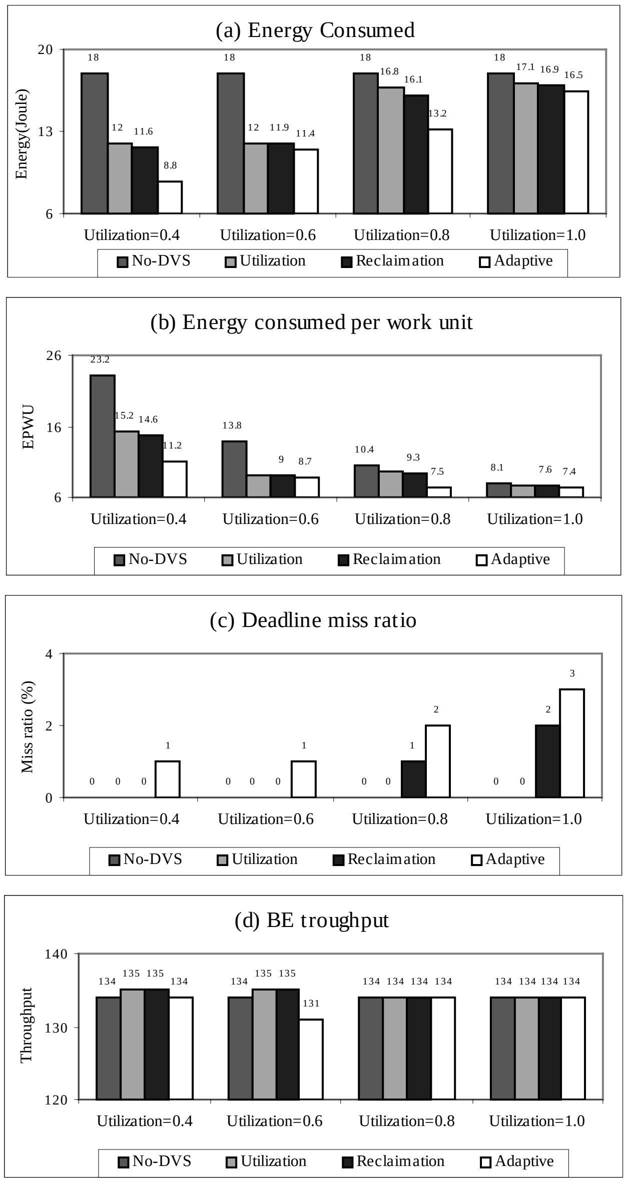

Figure 14. Effects by varying utilization boundary

Figure 14 shows the results when the utilization boundary is 0.4, 0.6, 0.8, and 1, respectively. Application performance varies slightly, 2% for deadline miss ratio and 3% for BE throughput, for different boundaries. However, with the increment of the utilization boundary, the energy consumption increases correspondingly while EPWU decreases, because the processor completes more work units with higher utilization. Another significant result is that the adaptive SRT-DVS algorithm saves more energy relative to the utilization-based algorithm when the utilization boundary is lower (27% and 4% for boundary 0.4 and 1, respectively). The reason is that there is more portion of BE workload when boundary is lower (50% and 20% of BE workload for boundary 0.4 and 1, respectively), thus leaving more energy saving potential for the adaptive algorithm.

5. RELATED WORK

Our proposed VS-CBS algorithm is built on the CBS algorithm [1], which, in turn, is inspired by dynamic sporadic server (DSS) [22], constant utilization server (CUS) [6], and total bandwidth server (TBS) [22]. All these servers can provide temporal isolation between tasks through the rules of budget replenishment and consumption. CUS and TBS, however, cannot be directly used to support multimedia applications, because their correct behavior relies on a precise estimation of WCETs, which is difficult to achieve in an open system. DSS and CBS remove the need of knowing WCETs and provide overrun protection and handling, thereby suitable for multimedia applications. But they may result in unused bandwidth. Several algorithms have been proposed to reclaim the unused bandwidth. In [12], the authors present a greedy reclamation algorithm reduce the total number of preemption. In [2], a reclamation algorithm is proposed to control overrun by sharing the residual bandwidth. In addition to these server mechanisms, resource reservation is also a common mechanism to provide isolation between soft real-time and best-effort applications in an open system [3][11][15][16][21][23][26]. The reservation mechanism, typically combined with admission control and real-time scheduling, allows an application to reserve processor resource and guarantees resource availability to admitted applications.

Most above approaches, however, assume a constant processor speed, and hence do not address the power management problem. In contrast, our VS-CBS algorithm delivers soft real-time guarantees to multimedia applications in a variable speed mobile environment. It enables applications to make optimistic resource reservation based on their average execution time, and ensures the correctness of reservation admission and enforcement with dynamic speed adjustment. The VS-CBS server protects overruns resulted from the average-based reservation and handles them in a predicted bounded time. The unused bandwidth of a VS-CBS server is efficiently reclaimed for overrun handling or/and energy saving.

Dynamic voltage scaling is usually used to reduce the processor energy consumption with tradeoff between performance and power consumption [7][10][18][19][24][25]. Recently, DVS has been investigated in two main areas, general-purpose operating systems (GPOS) and real-time operating systems (RTOS). Algorithms in a GPOS predict the workload using some heuristics. In [24], the authors proposed and evaluated several algorithms, which are interval-based and adjust speed according to the processor utilization in previous intervals. These algorithms have been extended by [8] and [14]. [17] and [10] proposed implementable algorithms, removing the requirement of the future knowledge in [24] and [8]. Several approaches, including estimation of workload distributions [13] and prediction of episodic interaction [7], have been proposed to maintain good interactive performance under DVS in a GPOS. The heuristic prediction in these algorithms is typically inaccurate, without understanding of application resource requirements, so they may violate the timing constraints of multimedia applications.

Algorithms in RTOS, typically integrated with the OS scheduler, deliver workload from information, for example, WCETs, specified by real-time applications. [25] presents a formal model on how to compute optimal speed-setting policy by constructing an EDF scheduling. [20] implements a heuristic energy priority scheduling algorithm in a StrongArm-based platform to apply DVS to real-time applications. Several papers combine offline analysis with online adjustment to achieve more energy. For example, in [9], the CPU speed is adjusted according to a probability distribution of actual execution time after an offline WCET-based analysis. In contrast to adjusting the speed to reclaim the slack, the *Look-Ahead* algorithm presented in [19] tries to initially set a lower speed to avoid the slack by deferring as much work as possible without missing deadlines.

Most of the above DVS algorithms in RTOS assume an embedded hard real-time environment, where the workload consists of only real-time applications and the WCETs of applications are achievable or deliverable. In contrast, our SRT-DVS algorithms target a typical open mobile environment with mix of multimedia and best-effort applications, where it is difficult to precisely estimate the WCETs of applications. Moreover, our approaches support flexibility to tradeoff between energy saving and total workload completion. For example, there are two options, slowing down the processor or admitting more applications, after reducing the bandwidth of BE tasks.

6. CONCLUSION

Battery-powered mobile devices are becoming increasingly important platforms for the mix of soft real-time multimedia and best-effort applications. Such open mobile systems present two major design concerns. To support the high resource demand from a dynamic application workload, they need powerful processors with high energy consumption. For the sake of mobility, however, they require low energy consumption to maximize the battery lifetime. In this paper, we have presented a framework to integrate dynamic voltage scaling into soft real-time scheduling, achieving energy saving of DVS while preserving resource guarantees of SRT scheduling. The SRT scheduler allows multimedia applications to reserve resource based on their average resource usage, and ensures the correctness of reservation admission and enforcement in a variable speed context through the VS-CBS algorithm. Given the workload and resource usage information provided by the scheduler, the DVS manager dynamically adjusts the processor speed to save energy while meeting application resource requirements. In particular, we have proposed three SRT-DVS algorithms, which reduce the processor energy consumption by exploiting the unallocated resource and unused resource, as well as adapting resource allocation to avoid (or reduce) the unused resource.

We have simulated the SRT-DVS algorithms and performed extensive experiments to evaluate them according to metrics of energy saving and application performance guarantee. The results demonstrate that these algorithms are able to save about 4% to 32% energy, while slightly affecting the SRT deadline miss ratio and BE throughput. In the future, we will implement these algorithms to actually adjust the processor speed through the Advanced Configuration and Power Interface (ACPI) [5].

7. ACKNOWLEDGMENTS

This work was supported by the Air Force grant F30602-97-2-0121, the NSF CCR 96-23867, the NSF CISE EIA 99-72884, and the NASA grant under NASA NAG 2-1250. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, NASA, or the U.S. government.

8. REFERENCES

[1] L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” *Proc. of the 19th IEEE Real-Time Systems Symposium*, Phoenix, AZ, 4-13, Dec 1998.

[2] M. Caccamo, G. Buttazzo, and L. Sha, “Capacity Sharing for Overrun Control,” *Proc. of the 21th IEEE Real-Time Systems Symposium*, Orlando, FL, Dec 2000.

[3] H. H. Chu and K. Nahrstedt, “CPU Service Classes for Multimedia Applications,” *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems (ICMCS'99)*, Florence, Italy, June 1999.

[4] E. G. Coffman, Jr., R. R. Muntz, and H. Trotter, “Waiting time distributions for processor-sharing systems,” *Journal of the ACM*, 17(1):123-130, Jan 1970.

[5] Compaq, Intel, Microsoft, Phoenix and Toshiba, *Advanced Configuration and Power Interface Specification*, [online] http://www.teleport.com/~acpi/spec.htm, July 2000.

[6] Z. Deng and J. S. Liu, “Scheduling of Real-Time Applications in an Open Environment,” *Proc. of 18th IEEE Real-time Systems Symposium*, San Francisco, CA, 308-319, Dec 1997.

[7] K. Flautner, S. Reinhardt, and T. Mudge, “Automatic performance setting for dynamic voltage scaling,” *Proc. of the 17th conference on Mobile computing and networking MOBICOM’01*, 260-271, Rome, Italy, July 2001.

[8] K. Govil, E. Chan, and H. Wasserman, “Comparing algorithms for dynamic speed-setting of a low-power CPU,” *Proc. ACM Int'l Conf. on Mobile Computing and Networking*, Berkeley, CA, 13-25, Nov 1995.

[9] F. Gruian, “Hard real-time scheduling for low energy using stochastic data and DVS processors,” *Proc. of the International Symposium on Low-Power Electronics and Design ISLPED’01*, Huntington Beach, CA, Aug 2001.

[10] D. Grunwald, P. Levis, C. Morrey III, M. Neufeld, and K. Farkas, “Policies for Dynamic Clock Scheduling,” *Proc. of the 4th Symposium on Operating System Design and Implementation*, San Diego, CA, Oct 2000.

[11] M. B. Jones and J. Regehr, “CPU Reservations and Time Constraints: Implementation Experience on Windows NT,” *Proc. of the 3rd USENIX Windows NT Symposium*, Seattle, WA, 93-102, July 1999.

[12] G.Lipari and S.K. Baruah “Greedy reclaimation of unused bandwidth in constant bandwidth servers,” *Proc. of the 12th Euromicro Conference on Real-Time Systems,* Stokholm, Sweden, June 2000.

[13] J. Lorch, and A. Smith, “Improving dynamic voltage scaling algorithms with PACE,” *Proc. of the ACM SIGMETRICS 2001 Conference,* Cambridge, MA, 50–61, June 2001.

[14] T. Martin, “Balancing batteries, power, and performance: system issues in CPU speed-setting for mobile computers,” PhD thesis, Carnegie Mellon University, 1999.

[15] C.W. Mercer, S. Savage and H. Tokuda, “Processor Capacity Reserves: Operating System Support for Multimedia applications,” *Proc. of the International Conference on Multimedia Computing and Systems*, 90-99, May 1994.

[16] J. Nieh and M. S. Lam, “The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications,” *Proc. of the 16th ACM Symposium on Operating Systems Principles,* St-Malo, France, 184-197, Oct 1997.

[17] T. Pering, T. Burd, and R. Brodersen, “The simulation and evaluation of dynamic voltage scaling algorithms,” *Proc. Int'l Symposium on Low Power Electronics and Design,* 76-81, Aug. 1998.

[18] T. Pering, T. Burd, and R, Brodersen, “Voltage scheduling in the lpARM Microprocessor System,” *Proc. of the Intl. Symposium on Low Power Electronics and Design,* Rapallo, Italy, July 2000.

[19] P. Pillai and K. G. Shin, “Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems,” *Proc. of 18th Symposium on Operating Systems Principles,* Banff, Canada, Oct, 2001.

[20] J. Pouwelse, K. Langendoen, and H. Sips, “Energy Priority Scheduling for Variable Voltage Processors,” *Proc. of Int. Symp. on Low Power Electronics and Design (ISLPED 2001)*, Huntington beach, CA, Aug 2001.

[21] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa, “Resource Kernels: A Resource-Centric Approach to Real-Time Systems,” *Proc. of the SPIE/ACM Conference on Multimedia Computing and Networking,* Jan 1998.

[22] M. Spuri and G. Buttazzo, “Scheduling Aperiodic Tasks in Dynamic Priority Systems,” *Real-Time Systems* 10(2): 179-210, 1996.

[23] V. Sundaram, A. Chandra, P. Goyal, P. Shenoy, J. Sahni and H.M.Vin, “Application Performance in the QLinux Multimedia Operating System,” *Proc. of the 8th ACM Conference on Multimedia,* Los Angeles, CA, Nov 2000.

[24] M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced CPU energy,” *Proc. of USENIX Symposium on Operating Systems Design and Implementation,* 13-23, Nov 1994.

[25] F. Yao, A. Demers, and S. Shenker, “A scheduling model for reduced CPU energy,” *Proc. of the 36th Annual Symposium on Foundations of Computer Science,* Milwaukee, WI, Oct 1995.

[26] W. Yuan, K. Nahrstedt, and K. Kim, “R-EDF: A Reservation-Based EDF Scheduling Algorithm for Multiple Multimedia Task Classes,” *Proc. of the 7th IEEE Real-Time Technology and Applications Symposium,* Taipei, Taiwan, May 2001.