

# TEMPORAL WORKLOAD ANALYSIS AND ITS APPLICATION TO POWER-AWARE SCHEDULING

Ye-In Seol<sup>1</sup>, Jeong-Uk Kim<sup>1</sup> and Young-Kuk Kim<sup>2</sup>.

<sup>1</sup>Green Energy Institute, Sangmyung University, Seoul, South Korea

<sup>2</sup>Dept. of Computer Sci. & Eng., Chungnam Nat'l University, Daejeon, South Korea

## ABSTRACT

*Power-aware scheduling reduces CPU energy consumption in hard real-time systems through dynamic voltage scaling(DVS). The basic idea of power-aware scheduling is to find slacks available to tasks and reduce CPU's frequency or lower its voltage using the found slacks. In this paper, we introduce temporal workload of a system which specifies how much busy its CPU is to complete the tasks at current time. Analyzing temporal workload provides a sufficient condition of schedulability of preemptive early-deadline first scheduling and an effective method to identify and distribute slacks generated by early completed tasks. The simulation results show that proposed algorithm reduces the energy consumption by 10-70% over the existing algorithm and its algorithm complexity is  $O(n)$ . So, practical on-line scheduler could be devised using the proposed algorithm.*

## KEYWORDS

*Power-aware Scheduling, Real-time Scheduling, Embedded Systems*

## 1. INTRODUCTION

Energy consumption issues are becoming more important for mobile or battery-operated embedded systems. Since the energy consumption of CMOS circuits, used in various microprocessors, has a quadratic dependency on the operating voltage( $E \propto V^2$ )[2], it is a very useful method for reducing energy consumption to lower the operating voltage of circuits. But, lowering the operating voltage also decreases its clock speed or frequency, so the execution times of tasks are prolonged. This makes problem more complex for hard real-time embedded systems where timing constraints of tasks should be met.

There has been significant research effort on Dynamic Voltage Scaling(DVS) for real-time systems to reduce energy consumption while satisfying the timing constraints[1,4-6,8-11,13]. The chance to lower its voltage occurs when there are slacks for the current executing real-time task. Generally there are two sources of slack, i.e., when the sum of worst case execution times of tasks is below the CPU's processing capacity and when a task completes early without consuming its worst case execution time. Main concern of DVS algorithms is how to identify those slacks and how to distribute them.

DVS algorithms also depend on the scheduling policy, task model, and processor architecture. In this paper, we adopt Early-Deadline First(EDF) scheduling policy, periodic or sporadic task model and uniprocessor system. EDF assigns dynamic priority for ready tasks and it is known as optimal for uniprocessor system[7]. Periodic task model assumes tasks are released periodically and their relative deadlines are the same as their respective periods. Sporadic task model allows

tasks are released randomly, but there is a restriction on the minimum inter-arrival time of the same task. In those task models, we require a priori knowledge of tasks, i.e., period, worst-case execution time, or minimum inter-arrival time, etc. Some works don't assume these kinds of information[5], or adopt aperiodic task model[11,12]. But, many hard real-time applications are classified as periodic or sporadic, so considering periodic and/or sporadic task model is practical.

In this paper, we introduce a notion of temporal workload which reflects how much busy the system is. It will be showed that analyzing temporal workload provides a useful method for real-time scheduling, especially EDF. We analyze the behaviors of EDF scheduling using temporal workload, and present some interesting results by which we understand more deeply the features of EDF scheduling.

Also we apply the analysis results into power-aware scheduling, and present an algorithm which adopts the results of CC-EDF[9] and temporal workload analysis. The simulation results show that the proposed algorithm with affordable algorithmic complexity reduces more energy consumption than previous work.

The rest of the paper is organized as follows. In section 2, we present the system model and notations adopted in this paper and introduce some previous works which motivate the work done in this paper. In section 3, we define the temporal workload and analyze EDF scheduling using it. In section 4, we present a power-aware scheduling algorithm derived from the temporal workload analysis. In section 5, simulation results will be provided and section 6 will conclude and discuss the future directions of this paper.

## 2. MOTIVATION

In this section we present the system model and introduce the result of the related work.

### 2.1. System Model

We consider preemptive hard real-time system in which all tasks are periodic or sporadic and mutually independent. The target processor is DVS enabled uniprocessor and its supply voltage and frequency are varied continuously between  $[v_{\min}, v_{\max}]$  and  $[f_{\min}, f_{\max}]$ , respectively. Let  $T = \{T_1, T_2, \dots, T_n\}$  be a set of periodic or sporadic tasks. Each task is represented as  $T_i = (P_i, C_i, D_i)$  where

- $P_i$  is period for periodic task or minimum inter-arrival time for sporadic task;
- $C_i$  is work-case computation time for task  $T_i$  at the maximum frequency;
- $D_i$  is relative deadline of a task  $T_i$ .  
If a instance or job of task  $T_i$  released at  $R_i$ , then its absolute deadline( $d_i$ ) is  $R_i + D_i$ . We will consider tasks only with  $D_i = P_i$ , so task  $T_i$  could be represented as  $(P_i, C_i)$ . Also the following notations will be used.
- $U_i$  or  $TU_i$  : the worst-case utilization of task  $T_i$  at the maximum frequency, i.e.,  $U_i = C_i/P_i$ .
- $U$  or  $TU$ : the worst-case total utilization of all tasks in the system, i.e.,  $U = \sum_i U_i$ .
- $CC_i$  : task's actual computation time which should be less than  $C_i$ . For uncompleted tasks,  $CC_i = C_i$ .
- $CU_i$  : actual utilization of task, i.e.,  $CU_i = CC_i/P_i$ .
- $CU$  : actual total utilization of system, i.e.,  $CU = \sum_i CU_i$
- $RC_i$  : task's remaining computation time.
- $T_{i,j}$  : jth instance or job of task  $T_i$ .

- $\alpha$  : current frequency ratio, i.e.,  $f_{cur}/f_{max}$
- $E_i$  : task's execution time, i.e.,  $E_i = C_i \times 1/\alpha$ .
- $RE_i$  : task's remaining execution time, i.e.,  $RE_i = RC_i \times 1/\alpha$
- $R_i$ : release time of task  $T_i$

## 2.2. Related Work

EDF scheduling has been extensively investigated in the area of real-time and power-aware scheduling[1,3-5,7-12]. But, some dynamic natures of EDF were not fully exploited, for example dynamic density function introduced in [6]. Dynamic density of a job is defined as its remaining execution time divided by the time to deadline. They deal with the case of unit execution time and multi-processor system using dynamic density function. Temporal workload introduced in this paper is similar or identical to dynamic density function, so we can say that we extended their results into more general task model, but uniprocessor system.

While devising a new power-aware scheduling algorithm based on the temporal workload analysis, we especially considered the results presented by Pillai and Shin[9]. They introduced a cycle-conserving method to real-time DVS. This method reduces the operating frequency on each task completion and increases on each task release. When a task completes its current invocation after using  $CC_i$  computation time, they treat the task as if its worst-case execution time were  $CC_i$ . So processor speed could be set as the actual total utilization CU which is always less than or equals to worst-case total utilization TU.

Mei et al.[8] integrated the above cycle-conserving method and the result of Qadi et al.[10] for sporadic task set. But, these method doesn't fully utilize the slack generated. Let's see the following figure. If a task completed at  $t_c$ , then during the time interval  $[R_i, t_c]$  the system operated at higher frequency than required. This observation provides a clue to more slow down the processor when a task completes. We will show later that the amount of slack which could be used for lowering processor frequency is related with temporal workload of the completed task.

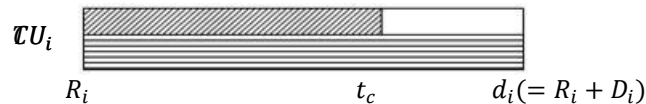


Figure 1. CC-EDF[9] or CC-DVSST[8] Schedule

## 3. TEMPORAL WORKLOAD ANALYSIS

In this section, we introduce some definitions and analyze behaviors of EDF scheduling. This section will provide a concrete theoretical basis for the power-aware scheduling algorithm presented in the next section.

### 3.1. Temporal Workload

*Definition 1:* Temporal workload of a task  $T_i$  at time  $t$ ,  $TW_i(t)$  or  $TW_i$  if not confusing, is defined as  $RC_i/(D_i - t)$  or 0 if it is completed before or at  $t$ . Temporal workload of a system at time  $t$ ,  $TW(t)$  or  $TW$ , is the sum of temporal workloads of all tasks in the system at time  $t$ .

Temporal workload of a task is similar or identical to dynamic density of a job[6], but we introduce new notion of temporal workload to distinguish remaining execution time and

remaining computation time, so more applicable to power-aware scheduling. Following notations are also used throughout this paper.

$TW^A$  or  $TW^A(t)$  : temporal workload of task set A

$TW_i^A$  or  $TW_i^A(t)$  : temporal workload of a task  $T_i$  in task set A

$TW(t_s, (T_i, \delta_i), (T_j, \delta_j))$  :  $TW(t_s + \delta_i + \delta_j)$  when executes  $T_i$  during  $[t_s, t_s + \delta_i)$ , and executes  $T_j$  during  $[t_s + \delta_i, t_s + \delta_i + \delta_j)$ .

For the time being,  $RC_i = RE_i$  because we schedule tasks with full speed of CPU, i.e.,  $\alpha = 1$ .

*Lemma 1:*  $TW$  is monotonically decreasing during a time interval  $[t, t + \delta)$  if no task was released during that interval,  $TW(t) \leq 1$ , and we schedule with EDF.

*Proof:* Let task  $t_i$  was executed during  $[t, t + \delta)$ , then by definition of temporal workload,

$$\begin{aligned} TW(t) &= \frac{RC_1}{d_1-t} + \dots + \frac{RC_i}{d_i-t} + \dots + \frac{RC_n}{d_n-t}, TW(t+\delta) = \frac{RC_1}{d_1-t-\delta} + \dots + \frac{RC_i-\delta}{d_i-t-\delta} + \dots + \frac{RC_n}{d_n-t-\delta}. \\ TW(t) - TW(t+\delta) &= RC_1 \left( \frac{1}{d_1-t} - \frac{1}{d_1-t-\delta} \right) + \dots + \left[ \frac{RC_i}{d_i-t} - \frac{RC_i-\delta}{d_i-t-\delta} \right] + \dots + RC_n \left( \frac{1}{d_n-t} - \frac{1}{d_n-t-\delta} \right) \\ &= \frac{-RC_1\delta}{(d_1-t)(d_1-t-\delta)} + \dots + \frac{-\delta RC_i + \delta(d_i-t)}{(d_i-t)(d_i-t-\delta)} + \dots + \frac{-RC_n\delta}{(d_n-t)(d_n-t-\delta)} \\ &= \frac{\delta}{d_i-t-\delta} - \sum_j \frac{\delta RC_j}{(d_j-t)(d_j-t-\delta)} = \frac{\delta}{d_i-t-\delta} \left( 1 - \sum_j \left( \frac{d_i-t-\delta}{d_j-t-\delta} \times \frac{RC_j}{d_j-t} \right) \right) \end{aligned} \quad (1)$$

By assumption,  $TW(t) = \sum_j \frac{RC_j}{d_j-t} \leq 1$ , and for we schedule with EDF,  $\frac{d_i-t-\delta}{d_j-t-\delta}$  is always less than or equals to 1, so Equation (1) is always larger than or equals to 0 which implies  $TW$  is monotonically decreasing.

Now we consider which task we schedule at time  $t$  effects on temporal workload of a system.

*Lemma 2:*  $TW(t, (T_i, \delta)) < TW(t, (T_j, \delta))$  if  $d_i < d_j$ .

*Proof:*  $TW(t, (T_i, \delta))$  and  $TW(t, (T_j, \delta))$  are the temporal workloads of a system at time  $t + \delta$  when we schedule  $T_i$  and  $T_j$  respectively at time  $t$ . So,

$$\begin{aligned} TW(t, (T_i, \delta)) &= \frac{RC_1}{d_1-t-\delta} + \dots + \frac{RC_i-\delta}{d_i-t-\delta} + \dots + \frac{RC_j}{d_j-t-\delta} + \dots + \frac{RC_n}{d_n-t-\delta} \\ TW(t, (T_j, \delta)) &= \frac{RC_1}{d_1-t-\delta} + \dots + \frac{RC_i}{d_i-t-\delta} + \dots + \frac{RC_j-\delta}{d_j-t-\delta} + \dots + \frac{RC_n}{d_n-t-\delta} \\ TW(t, (T_i, \delta)) - TW(t, (T_j, \delta)) &= \frac{RC_i-\delta}{d_i-t-\delta} - \frac{RC_i}{d_i-t-\delta} + \frac{RC_j}{d_i-t-\delta} - \frac{RC_j-\delta}{d_i-t-\delta} = \delta \left( \frac{1}{d_j-t-\delta} - \frac{1}{d_i-t-\delta} \right) \end{aligned} \quad (2)$$

By assumption, Eq. 2 is less than 0 which implies Lemma 2 is true.

Lemma 2 provides another clue that EDF scheduling policy is optimal in preemptive uniprocessor scheduling.

*Corollary 1:*  $TW(t, (T_i, \delta)) = TW(t, (T_j, \delta))$  if  $d_i = d_j$ .

Corollary 1 states that temporal workload of a system doesn't depend on which task execute at time  $t$  when tasks' deadlines are the same.

*Lemma 3:*  $TW(t_s, (T_i, \delta_i), (T_j, \delta_j)) = TW(t_s, (T_j, \delta_j), (T_i, \delta_i))$  if  $d_i, d_j < \delta_i + \delta_j$ .

*Proof:*  $TW(t_s, (T_i, \delta_i), (T_j, \delta_j))$  is the temporal workload of a system when we schedule  $T_i$  at  $[t_s, t_s + \delta_i)$  and  $T_j$  at  $[t_s + \delta_i, t_s + \delta_i + \delta_j)$  and  $TW(t_s, (T_j, \delta_j), (T_i, \delta_i))$  is the temporal workload of a system when we schedule  $T_j$  at  $[t_s, t_s + \delta_j)$  and  $T_i$  at  $[t_s + \delta_j, t_s + \delta_j + \delta_i)$ , but at time  $t = (t_s + \delta_i + \delta_j)$ ,  $T_i$  has the same  $TW_i$  because  $RC_i$  is the same for the two cases, and  $T_j$  also. So, Lemma 3 holds.

Lemma 3 implies that the order of execution has no effect on the last temporal workload of a system if there is no deadline miss. Which task executed and how much it executed during a time interval concern only the calculation of the last temporal workload of a system.

At Lemma 1, we proved the monotonic decreasing property of temporal workload under EDF scheduling. More investigation shows that when a task's deadline expires, then the temporal workload of a system decreases as least as the temporal workload of the task if there is no task release during the time interval. Following Lemma proves the above discussion.

*Lemma 4:*  $TW(t + D_i) \leq TW(t) - TW_i(t)$  if there is no task release during  $(t + D_i)$ ,  $TW(t) \leq 1$  and we schedule using EDF.

*Proof:* Let's consider a ideal CPU that executes every tasks concurrently proportional to their initial temporal workload. This could be accomplished by minimizing  $\delta$  as small as possible in Fig. 2.

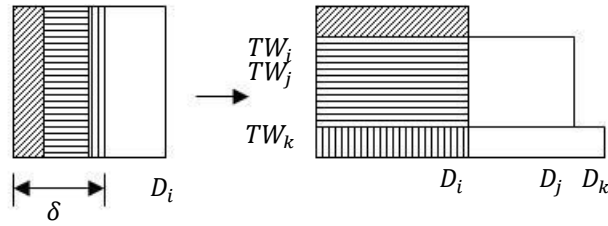


Figure 2. Ideal CPU execution

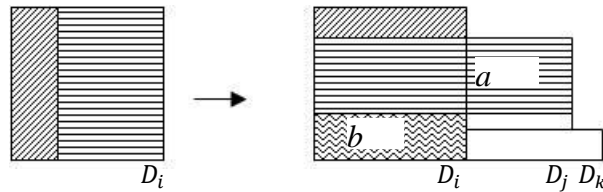


Figure 3. Real CPU execution

The execution of ideal CPU could be relocated as the right side of Fig. 2. Because only the amount of execution time contributes to last temporal workload by Lemma 3, temporal workload of both side of Fig. 2 is the same. Also, the execution of a real CPU could be relocated as the right side of Fig. 3. At the right side of Fig. 3, the areas of 'a' and 'b' equal for total computation time could not be changed. By Lemma 2, temporal workload of the real CPU is less than or equals to that of the ideal CPU. The temporal workloads of tasks in ideal CPU don't change during the time interval  $[t, t + D_i)$  and the temporal workload of the task whose deadline is  $t + D_i$  sets to zero at  $t + D_i$ . So,

$$\begin{aligned}
 TW(t + D_i) \text{ of the real CPU} &\leq TW(t + D_i) \text{ of the ideal CPU} \\
 &= TW(t) - TW_i(t)
 \end{aligned}
 \tag{3}$$

In the process of proving Lemma 4, we presented a useful high limit of temporal workload under EDF scheduling. It is that temporal workload of ideal CPU system provides an intuitive high limit as Eq. (3) states.

### 3.2. Temporal Workload Isomorphic

Now we introduce new notion to compare temporal workloads of different systems.

*Definition 2:* Task systems, A and B, are temporal-workload-isomorphic if  $\exists \alpha$  such that  $TW^A(t) = \alpha TW^B(t)$  for all time t.

If temporal workload value of system A is always the same or constant multiples of that of system B, then we can easily assume that schedulability conditions of two systems are very close or the same and schedule behaviors are very similar to each other.

*Lemma 5:* Following two periodic task systems, A and B are temporal-workload-isomorphic under worst-case execution scenarios.

$A = \{T_i | T_i = (P_i, C_i), \text{ where } \{P_i\} \text{ could be multiset. EDF scheduling}\}$

$B = \{T'_j | T'_j = (P'_i, \sum_k C_k) \cdot \{P'_i\} \text{ is distinct set of } \{P_i\}, \text{ i.e., } P'_i \neq P'_j \text{ if } i' \neq j'. C_k \text{ is for all } k \text{ of } A \text{ such that } P'_i = P_k. \text{ EDF scheduling}\}.$

*Proof:* Let's consider  $T'_j$  in B and its cousin tasks  $\{T_i\}$  in A, i.e.,  $P'_j = P_i$  for  $\forall i$ . Then release times and deadlines of all above tasks are the same. Under EDF scheduling, deadline of task is the only criterion of scheduling, then we can make the following condition hold that if we schedule a job of  $T'_j$ , then a job of some corresponding  $T_i$ 's is scheduled. If not, then there could be one or more jobs whose deadline is coincided with  $T'_j$ . But, despite the existence of another job of the same deadline and a different schedule could be done at that moment,  $TWA = TWB$  by Lemma 3 and Corollary 1. Also, the last time at which all jobs of the same deadline are completed is the same for A and B. So, Lemma 5 holds.

Now, we consider the schedulability condition of task system using temporal workload. Following Lemma is directly implied from the definition of temporal workload because temporal workload identifies the amount of work to be done until deadline of each task.

*Lemma 6:* If two finite task system A and B are temporal-workload-isomorphic, then schedulability conditions of two systems are identical.

*Proof:* We prove this Lemma by contradiction. If task system A has violated its timing constraints and B not, then there exists a task  $T_i$  of A such that

$$\lim_{t \rightarrow D_i} \frac{RC_i}{D_i - t} = \lim_{t \rightarrow D_i} \frac{D_i - t + \delta}{D_i - t} = \infty.$$

But temporal workload of system B could not reach infinite because it has finite tasks and each task of system B has finite value of temporal workload. This contradicts with assumption of this Lemma.

### 3.3. Upper Bound of Temporal Workload

*Definition 3:* Task system A is temporal-workload-upper-bounded if  $TWA(t) \leq \beta$  for all time t.  $TWUB^A$  will denote the upper bound  $\beta$ .

Because temporal workload of a system denotes the ratio of overhead to CPU capacity, we may schedule all the tasks of the system without violating its timing constraints if TWUB is below or the same as 1. Following theorem shows above discussion is true.

*Theorem 1:* If  $TW^A(t) \leq 1$  for all time  $t$ , i.e.,  $TWUB^A = 1$ , then task set  $A$  is schedulable using EDF.

*Proof:* We will prove it by induction on time  $t$ . Let's assume that this theorem holds until time  $t$ , i.e., we successfully scheduled the tasks until time  $t$ . At time  $t$ , we can still schedule the highest priority task, i.e., the shortest deadline task, without violating its deadline if there is no task release during  $(t, t + D_i)$  for  $1 \geq TW^A(t) \geq TW_i^A(t)$ . If there is a task release at time  $t + \delta$  where  $t + \delta < t + D_i$ , then we can insist that this theorem still hold until time  $t + \delta$  because there is no deadline during  $(t, t + \delta)$ . From the assumption, at  $t + \delta$ , the temporal workload of task set  $A$  should be less or equals to 1. So, we proved that this theorem still holds until  $t + D_i$  or  $t + \delta$  which are larger than  $t$  if it holds at  $t$ .

Theorem 1 states that if we preserve the upper bound of temporal workload below or equal to 1, then it is always schedulable. Also theorem 1 provides a sufficient condition for the schedulability of a system. Now we investigate the temporal workload upper bounds of periodic task systems.

*Theorem 2:* For a periodic or sporadic task system  $T = \{T_i | (P_i, C_i), EDF\}$ ,  $TWUB^T$  is  $\sum_i C_i/P_i$ .

*Proof:* We already said that ideal CPU with EDF scheduling policy provides an upper bound of temporal workload. For a periodic or sporadic task system of ideal CPU, clearly its temporal workload is always less than or equals to its total utilization. So theorem 2 holds.

### 3.3. Temporal Workload at Lower Processing Speed

Now, we consider the case of  $E_i \neq C_i$ , i.e., CPU's processing speed is not always 1.

*Lemma 7:* Following two periodic task systems,  $A$  and  $B$  are temporal-workload-isomorphic under worst-case execution scenarios.

$A = \{T_i | T_i = (P_i, C_i), \sum_i C_i/P_i = \alpha, \text{CPU's processing speed is } \alpha (< 1), EDF\}$ ,

$B = \{T_i | T_i = (P_i, C'_i), C'_i = C_i/\alpha, \text{CPU's processing speed is } 1, EDF\}$ .

*Proof:* Clearly at time  $t = 0$ ,  $TW^A(t) = TW^B(t)/\alpha$  by definition 1. And when we execute a task of system  $A$ ,  $T_i$  at time  $t$ , following holds.

$$\begin{aligned} C_i(t + \delta) &= C_i(t) - \alpha\delta. \\ C'_i(t + \delta) &= C'_i(t) - \delta = \frac{C_i(t)}{\alpha} - \delta = \frac{C_i(t) - \alpha\delta}{\alpha} = \frac{C_i(t + \delta)}{\alpha} \end{aligned} \quad (4)$$

Eq. (4) states that if we execute tasks of the same deadline at the same time for system  $A$  and  $B$ , then the ratio of temporal workloads of two systems are not changed. Because  $A$  and  $B$  use EDF scheduling policy and execution times(not computation times) and deadlines of  $T_i$  and its corresponding  $T'_i$  are identical under worst-case execution scenarios, Lemma 7 holds. There could be the cases when we execute  $T_i$  of  $A$  and  $T'_j$  of  $B$ , i.e., deadlines of two tasks are coincidentally exact, but by Lemma 3 and Corollary 1, Lemma 7 still holds as we insisted at Lemma 5.

*Corollary 2:* For a periodic task system  $T = \{T_i | T_i = (P_i, C_i), \sum_i C_i/P_i = \alpha\}$ , then we can schedule with constant CPU speed  $\alpha$ .

*Proof:* By Lemma 7, we can construct a temporal-workload-isomorphic system of T whose TWUB is 1 and its CPU processing speed is 1. Then by theorem 1, the newly constructed task set could be successfully scheduled with EDF. By Lemma 6, this shows that original task set could be successfully scheduled.

Using Lemma 7 and Corollary 2, we can safely lower processing speed for periodic real-time task system when its total utilization is below 1 as many previous works for power-aware scheduling said.

Following theorem is the repetition of Theorem 4 of DVSST paper[10]. DVSST algorithm scales up CPU whenever new task releases and scales down whenever its deadline expires exactly as much as utilization of that task.

*Theorem 3:* Sporadic task system T with  $\sum_i C_i/P_i$  and DVSST algorithm, it is schedulable using EDF if and only if  $\sum_i C_i/P_i \leq 1$ .

*Proof: "Only If" part:* As the assertion stated in [10], if  $U > 1$ , then EDF will not find a feasible schedule, therefore DVSST combined with EDF will not find a feasible schedule.

*"If" part:* As we stated at Lemma 4, ideal CPU can provide an upper bound of temporal workload. So, whenever a new task released, temporal workload of ideal CPU system increased as much as utilization of that task during  $[R_i, d_i]$  and whenever a deadline of a task expires it decreased also as much as that amount. Now suppose that there is neither new task release nor deadline expiration during  $[t_1, t_2]$  and  $\alpha$  is the temporal workload of ideal CPU during the time interval, then we can construct a new task system whose total utilization is 1 and each task  $T_i'$  has the same deadline of original corresponding task  $T_i$  and its computation time  $C_i'$  is  $C_i/\alpha$ .

Then by Theorem 1, we can schedule new task system using EDF and by Lemma 6, we can say that we can schedule original task system with EDF for two task systems are temporal workload isomorphic during the interval. And for two task systems have identical deadline distributions, we always make sure that  $(T_i, T_i')$  pair of tasks execute at the same time in each system without violating EDF policy. So by Lemma 7, above process could be repeated at every time interval during which neither new task releases nor deadline expires. This proves 'if' part.

Following theorem 4 states that temporal workload analysis provides another useful schedulability test.

*Theorem 4:* Let  $UBS = \{\text{periodic or sporadic task systems whose total utilizations are less than or equal to 1}\}$ ,  $TWUBS = \{\text{task systems whose TWUB are less than or equal to 1}\}$ , and  $LFBS = \{\text{task systems whose loading factors}[5](= \sup((\sum_{t_1 < r_i, d_i < t_2} C_i)/(t_2 - t_1))) \text{ are less than or equal to 1}\}$ , then

$$UBS \subset TWUBS \subset LFBS.$$

*Proof:* By Theorem 2,3 and Corollary 2, clearly  $UBS \subseteq TWUBS$ . But TWUBS doesn't assume neither minimum interval nor periodicity of tasks, so  $TWUBS \neq UBS$ .

LFBS is the maximum set of tasks which could be scheduled by EDF because loading factor test provides necessary and sufficient condition[5]. So,  $TWUBS \subseteq LFBS$ . But, for the following task set  $T = \{T_i = (R_i, C_i, d_i) | R_i = 0, C_i = 1, d_i = i, 1 \leq i \leq N\}$ ,  $T \in LFBS$ , but  $T \notin TWUBS$ . So, Theorem 4 holds.



The temporal workload of the task set  $T$  in Theorem 4 is infinite when  $N$  is infinite, so there is no upper limited value of  $TWUB$  which provides necessary condition for schedulability test.

## 4. POWER-AWARE SCHEDULING ALGORITHM

### 4.1. An On-Line Algorithm

In section 3, we analyzed scheduling behaviours of EDF using temporal workload. Now we apply the results into power-aware scheduling. As we stated at section 2, previous algorithms such as CC-EDF, DVSST and CC-DVSST don't fully utilize the slacks generated by early completed tasks. But, based on the temporal workload analysis, we can find more slacks to slow down the CPU speed. Before presenting more discussion, let's introduce new definition.

*Definition 4:* Temporal idleness  $TI_i(t)$  of a task  $T_i$  at time  $t$  is defined as following.

$$\left\{ \begin{array}{l} 0 \text{ until its completion and after its deadline.} \\ RC_i/(d_i - t_c) \text{ if it was completed at time } t_c \text{ and } t = t_c. \\ TI_i(t) \pm \gamma \text{ if } t < t_c. \end{array} \right.$$

The real value of  $\gamma$  depends on the status of system and how to calculate will be presented later. Likewise the definition of temporal workload of a system, temporal idleness of a system,  $TI(t)$  or  $TI$ , is defined as the sum of  $TI_i(t)$ . Note that  $TI_i(t_c)$  is the same as  $TW_i(t_c)$  of uncompleted case. Now, consider the amount of computation time until its completion. At CC-EDF and CC-DVSST, they lower the processing speed of task  $T_i$  when it is completed. This means that its pace of computation is faster than actual execution needed as stated at section 2. Following figure shows the situation.

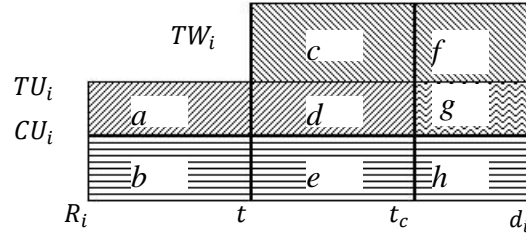


Figure 4. Computation time comparison

At time  $t_c$ , task  $T_i$  was completed, so during  $[R_i, t_c]$ , its computation time(= $a + b + d + e$ ) is larger than needed ( $= b + e$ ) as amount of  $(a + d)$ . If we assume that task  $T_i$  executed with its temporal workload during  $[t, t_c]$ , then  $TW_i(t_c) \times (t_c - t)$  equals to  $CC_i(= c + d + e)$  and  $TW_i(t_c) \times (d_i - t)$  equals to  $RC_i(= f + g + h)$ . But,  $CU_i \times P_i = CC_i(= b + e + h)$ ,  $(TU_i - CU_i) \times P_i = RC_i(= a + d + g)$ . So,  $RC_i = f + g + h = a + d + g$ , i.e.,  $f + h = a + d$ . It is that the amount of slack could be used until its deadline is  $(a + d) + g = (f + h) + g = f + g + h$ , i.e.,  $TI_i \times (d_i - t)$ . Following lemma shows it formally.

*Lemma 8:* At CC-EDF or CC-DVSST scheduling, the amount of computation time exceeding its actual pace until its completion time(= $t_c$ ) is the same as  $[TI_i(t_c) - (TU_i - CU_i)] \times (d_i - t_c)$ .

*Proof:*  $[TI_i(t_c) - (TU_i - CU_i)] \times (d_i - t_c)$

$$= \left[ \frac{RC_i}{d_i - t_c} - \frac{C_i}{D_i} + \frac{CC_i}{D_i} \right] \times (d_i - t_c) = RC_i - \frac{C_i}{D_i} \times (D_i + R_i - t_c) + \frac{CC_i}{D_i} \times (D_i + R_i - t_c)$$

$$= [RC_i D_i - C_i D_i - C_i R_i + C_i t_c + CC_i D_i + CC_i R_i - CC_i t_c] / D_i \quad (5)$$

But, by definition,  $C_i = CC_i + RC_i$ , so,

$$(5) = [C_i(t_c - R_i) + CC_i(t_c - R_i)] / D_i = [C_i / D_i - CC_i / D_i] \times (t_c - R_i) \\ = (TU_i - CU_i) \times (t_c - R_i)$$

Using Lemma 8, if a task  $T_i$  completed at time  $t$ , then we can slow down processing speed by amount of  $TI_i$  when executing lower priority tasks than  $T_i$  until its deadline, because CC-EDF or CC-DVSST slows down processing speed by the amount of  $(TU_i - CU_i)$ . The proposed algorithm tries to use slacks of already completed higher priority tasks which are necessarily generated by assuming worst-case execution scenarios and follows the result of CC-EDF when running task's priority is higher than those of already completed tasks. Also, if we cannot utilize those slacks by some reasons, i.e., when executing higher priority tasks or when slacks are too much to fully utilize, then we evenly distribute those unused slacks until corresponding deadlines.

One more consideration occurs when there is idle period. Let's consider periodic task set  $T = \{T_i = (P_i, C_i) | T_1 = (3,1), T_2 = (3,1), T_3 = (6,2)\}$ . If  $T_{1,1}$  and  $T_{2,1}$  complete at  $t=1$  and  $t=2$ , respectively, and  $T_{3,1}$  completes early at  $t=2.5$ , then  $TI_3(2.5) = 1.5/4.5 = 3/7$ . If we lower the processing speed as much as  $TI_3(2.5)$ , then actual processing capacity during  $t=[3,6]$  is  $(1 - 3/7) * 3 = 12/7$  which is less than sum of WCET of  $T_{1,2}$  and  $T_{2,2}$ . Deadline miss occurs because there is idle period during  $t=[2.5,3]$ . During the idle period, the total processing capacity which should be processed under the actual execution scenarios is larger than sum of slack used. So, we should reduce future slacks to compensate larger processing capacity. Following figure shows it.

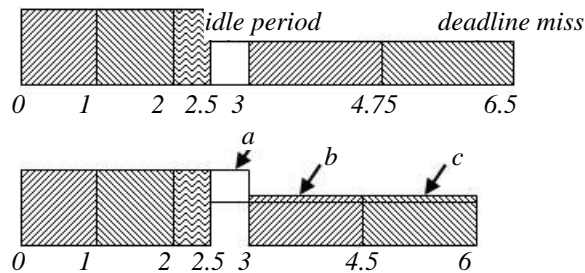


Figure 5. Slack reduction example

The amount of slack which should be reduced is  $TU - TI$ , and is the same as  $[CU - (TI - (TU - CU))]$ . At the above figure, the area 'a' is the same as 'b'+c'. And if  $TI$  exceeds  $TU$ , we could save some slacks because only  $TU \times (\text{length of idle period})$  should be processed by CPU.

```

global variables
last_idle_time
last_cpu_speed
last_calculation_time
ex_ratio, ex_flag, ex_task
 $\beta$  // current total utilization, i.e.,  $\sum_i CC_i/P_i + \sum_i C_i/P_i$ 
TC // completed task set sorted by deadline

compute_cpu_speed( $T_k$ ) //  $T_k$  is the highest priority task or NULL if no ready task
cpu_speed =  $\beta$ 
for all tasks  $t_i$  at TC
  if  $d_i \leq d_k$  or  $T_k$  is NULL
    if (cpu_speed  $\geq (TI_i - (TU_i - CU_i))$ )
      cpu_speed -=  $(TI_i - (TU_i - CU_i))$ 
    else
      ex_ratio =  $(TI_i - (TU_i - CU_i)) - \text{cpu\_speed}$ 
      ex_flag = 1, ex_task =  $T_i$ 
      cpu_speed = 0
      break
    else if (ex_flag == 0 &&  $(TI_i - (TU_i - CU_i)) \neq 0$ )
      ex_ratio =  $(TI_i - (TU_i - CU_i))$ 
      ex_flag = 1, ex_task =  $TI_i$ 
      break
last_calculation_time =  $t_{cur}$ 
return cpu_speed

decrease_temporal_idleness()
idle_time = last_idle_time -  $t_{cur}$ 
idle_work = idle_time * last_cpu_speed
for all tasks  $t_i$  at TC
  if (idle_work == 0)
    break
   $RI_i = (TI_i - (TU_i - CU_i)) \times (d_i - t_{cur})$ 
  if ( $RI_i \geq \text{idle\_work}$ )
     $RI_i -= \text{idle\_work}$ 
     $TI_i = RI_i / (d_i - t_{cur}) + (TU_i - CU_i)$ 
    break
  else
    idle_work = idle_work -  $RI_i$ 
     $TI_i = (TU_i - CU_i)$ 

increase_temporal_idleness()
for all tasks  $T_i$  whose priority is less than or equals to that of ex_task at TC
  if (ex_flag == 1)
    busy_ratio = ex_ratio
    ex_flag = 0
  else
    busy_ratio =  $(TI_i - (TU_i - CU_i))$ 
    busy_time = last_calculation_time -  $t_{cur}$ 
    busy_work = busy_time  $\times$  busy_ratio
  if ( $d_i > t_{cur}$ )
     $TI_i += (\text{busy\_work}) / (d_i - t_{cur})$ 

```

Figure 6. Temporal idleness management

```

when task  $T_i$  arrived
  if ( exceed_flag)
    increase_temporal_idleness()
  else if ( cpu was idle)
    decrease_temporal_idleness()
   $\beta \ += C_i/P_i$ 
  last_cpu_speed = compute_cpu_speed( $T_{cur}$ )
  set cpu speed as last_cpu_speed

when task  $T_i$  completed
   $TI_i = TW_i$ 
  if ( exceed_flag)
    increase_temporal_idleness()
  insert  $T_i$  into TC
  last_cpu_speed = compute_cpu_speed( $T_{cur}$ )
  if there is no task to execute
    last_idle_time =  $t_{cur}$ 
    set cpu as idle
  else
    set cpu speed as last_cpu_speed

on deadline of task  $T_i$ 
  if ( exceed_flag)
    increase_temporal_idleness()
  else if ( cpu was idle)
    decrease_temporal_idleness()
   $\beta \ -= C_i/P_i$ 
  delete  $T_i$  from TC
  last_cpu_speed = compute_cpu_speed( $T_{cur}$ )
  if there is no task to execute
    last_idle_time =  $t_{cur}$ 
    set cpu as idle
  else
    set cpu speed as last_cpu_speed

```

cf.  $t_{cur}$  : current time,

$T_{cur}$  : current ready task of highest priority or NULL if no ready task.

Figure 7. Power-aware scheduling algorithm

Now, following theorem proves the correctness of our algorithm.

*Theorem 5:* The algorithm presented at Fig. 3, schedules every periodic or sporadic task set if and only if  $\sum_i C_i/P_i \leq 1$ .

*Proof: "Only If" part:* If  $U > 1$ , then EDF will not find a feasible schedule, therefore our algorithm will not find a feasible schedule because our algorithm is the same as EDF or DVSST when tasks execute always at worst-case execution scenarios, i.e.,  $TI_i(t) = 0$  and  $TU_i = CU_i$  for  $\forall t$  and  $i$ .

*"If" part:* We will show that temporal workload of a system adopting our algorithm is always less than or equals to that of another ideal system which schedules tasks without violating their timing constraints. Then by lemma 6 and theorem 1, we can insist that our scheduling algorithm also satisfy real-time constraints.

Let's consider ideal CPU of ideal system which can anticipate task's actual computation time when released and can execute active tasks with concurrently with the speed of CU. This ideal system of figure 8 resembles the system of figure 2.

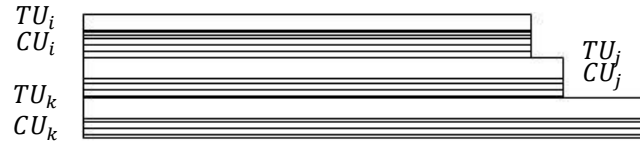


Figure 8. CC-EDF schedule : ideal CPU and ideal execution

Our real system starts to execute with TU speed and slows down with  $(TU - TI)$  when some tasks are completed. And we already said that  $[TI_i - (TU_i - CU_i)] \times (d_i - t_c) = B$  is exactly the same as  $(TU_i - CU_i) \times (t_c - R_i) = A$ . Therefore we can apply following operation to scheduling result of our real system which replaces some areas of A by the same areas of B. Following figures shows it.

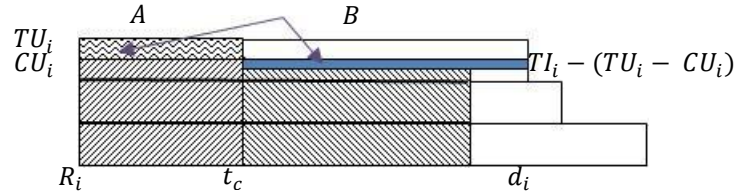


Figure 9. Slack exchange process

During the time interval  $[t_c, d_i]$  which has no idle period, we can apply the above operation for all areas of A. Then temporal workload of the last applied result is less than or equals to that of ideal system because the total computing capacity of the former is larger than or equals to that of the latter and the computing capacity of the former is always exhausted by tasks which have shorter or the same deadlines of tasks of the latter.

During idle period, there exist some areas of ideal system which cannot find counterparts of real system. Those areas could be filled up or substituted with the areas of future slacks of already completed tasks as was illustrated at figure 5 and above exchange operation could be also applied for them. The temporal workload of substitution and exchange result is also less than or equals to that of ideal system for the same reason.

Now temporal workload of our real system is always less than or equals to that of substitution and exchange result because the total computing capacity of the former is always larger than or the same as that of the latter and the same tasks are executed. This proves this theorem.

#### 4.2. An Illustrative Example

Let's consider an illustrative example, a periodic task system  $A = \{(P_i, C_i) | (2,1), (3,1), (7,7/6)\}$ . Total utilization of task system A is 1 and  $TWUB^A$  is also 1. Suppose that actual computation times of tasks  $T_1, T_2$  and  $T_3$  are  $1/2, 1/2,$  and  $1/3$  respectively. Then following figure shows the result of our scheduling algorithm.

At time  $t=1/2$ ,  $T_{1,1}$  completes its execution and we can set processing speed as  $\beta - (TI_{1,1} - (TU_{1,1} - CU_{1,1})) = TU - TI_{1,1} = 1 - 0.5/1.5 = 2/3$ .

At time  $t=1/2+3/4=5/4$ ,  $T_{2,1}$  completes its execution, the processing speed should be  $TU - TI_{1,1} - TI_{2,1}=1-1/3-0.5/(7/4)=8/21$ . During  $[5/4,2]$ , computation time of  $T_{3,1}$  is  $8/21*3/4=2/7$ . Deadline of  $T_{1,1}$  is 2 and  $T_{1,2}$  is released at that time, so  $TU - TI_{2,1}=1-2/7=5/7$ .

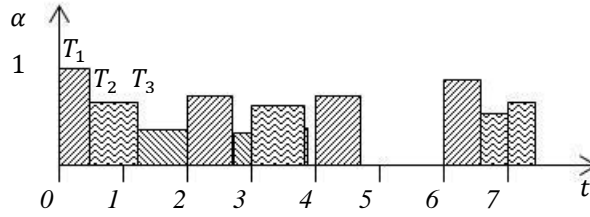


Figure 10. Scheduling example

At  $t=2+7/10$ ,  $T_{1,2}$  completes its execution and  $TU - TI_{1,2} - TI_{2,1}=1-0.5/(13/10)-2/7=30/91$ . During  $[2+7/10,3]$ ,  $T_{3,1}$  will execute with speed of  $30/91$  and its computation time is  $30/91*3/10=9/91$ , so total computation time of  $T_{3,1}$  until released is  $2/7+9/91=245/637$ .

At  $t=3$ ,  $T_{2,2}$  releases, and its deadline is less than  $T_{3,1}$ . So,  $T_{2,2}$  will execute with speed of  $\beta - (TI_{1,2} - (TU_{1,2} - CU_{1,2})) = TU - TI_{1,2}=1-5/13=8/13$ .

At time  $t=3+13/16$ ,  $T_{2,2}$  will be completed and  $T_{3,1}$  will execute with speed of  $TU - TI_{1,2} - TI_{2,2} \approx 0.387$ , and it will complete at  $t \approx 3.823$ . Then, during  $[3.823,4]$ , there is no ready task, so slack reduction process should be done.

At  $t=4$ ,  $T_{1,3}$  will be released and it will be executed with speed of  $\alpha \approx 0.722$  until  $t \approx 4.693$ . At  $t \approx 4.693$ , there is no ready task, so slack reduction process should be done at  $t=6$  when  $T_{1,4}$  and  $T_{2,3}$  are released. At  $t=6$ ,  $T_{1,4}$  will be executed with speed of  $\alpha \approx 0.889$ , and so on.

## 5. EXPERIMENTAL RESULTS

We evaluated our proposed algorithm using RTSIM[14] which is a real-time simulator. RTSIM can simulate the behaviors of dynamic voltage scaling algorithms as well as traditional real-time scheduling algorithms. In this simulation, it is assumed that a constant amount of energy is required for each cycle of operation at a given voltage. This quantum is scaled by the square of the operating voltage, consistent with energy dissipation in CMOS circuits ( $E \propto V^2$ )[2,5]. Only the energy consumed by CPU was computed and any other source of energy consumption was ignored. Also we do not consider preemption overheads, task switch overheads, and operating frequency change overheads. It is also assumed that the CPU consumes no energy during idle period and its operating frequency range is continuous at  $[f_{min}=0, f_{max}=1]$ .

We compared our proposed algorithm with CC-EDF[9], DVSST[10], and CC-DVSST[8]. CC-EDF assumes periodic task model, so we compared it at periodic task system. DVSST and CC-DVSST assume sporadic task model, so we compared them at sporadic task system.

To evaluate the effect of number of tasks in the system, we generated 10 or 20 tasks for each comparison. Their periods or minimum inter arrival times are chosen randomly in the interval  $[1-1000]$ ms. We divided task set into three groups to reflect more real environments. One group of tasks have short period in the interval  $[1-10]$ ms, another group of tasks have medium period in the interval  $[10-100]$ ms, and the last group of tasks have long period in the interval  $[100-1000]$ ms.

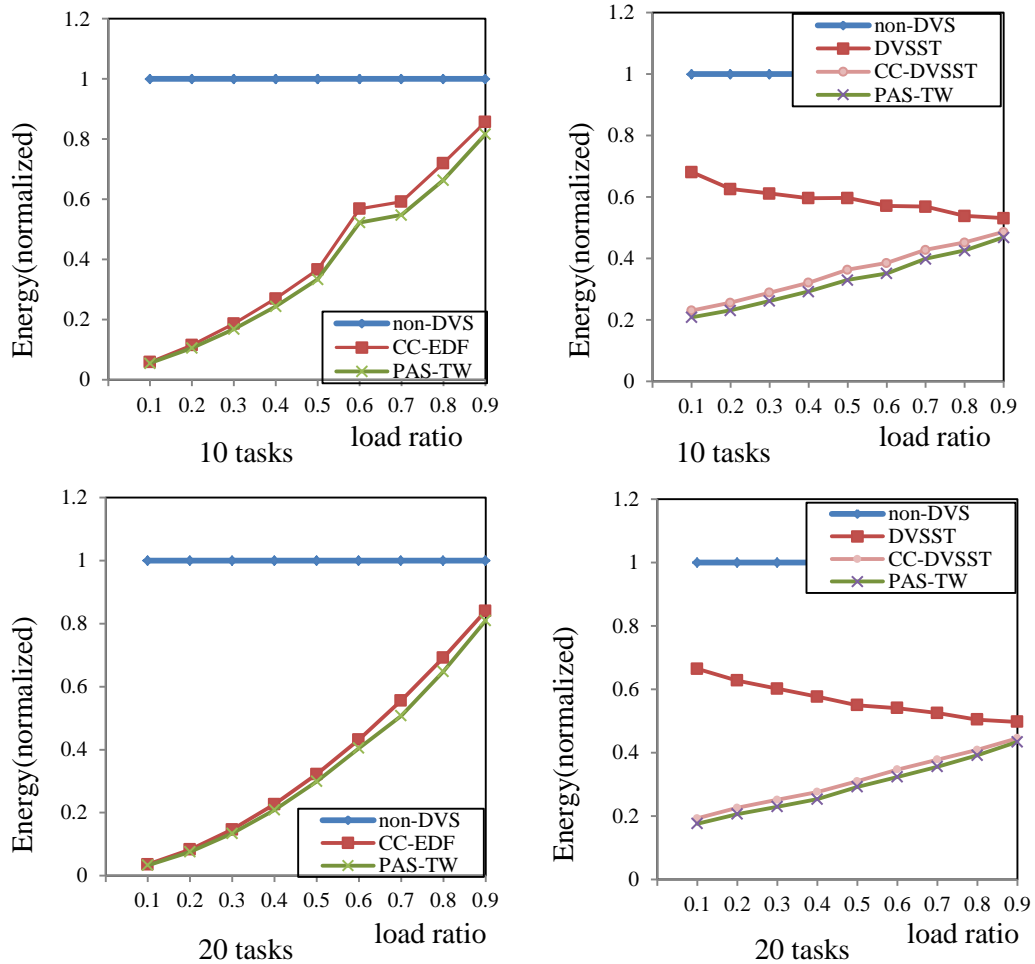


Figure 11. Simulation results : for periodic tasks(left) and for sporadic tasks(right)

The simulation was performed also by varying the load ratio of tasks, i.e., the ratio of the actual computation time to the worst case computation time. For all simulations, the worst-case total utilization of system is always 1, i.e.,  $TU = 1$ . Above figures show the simulation results.

Figure 11(left) shows the result for periodic task system. In this case, our proposed algorithm(PWA-TW) always outperforms CC-EDF, and the ratio of energy saving is up to 10%. The effect of number of tasks in the system could be neglected on the simulation result as the figures show. Also the number of CPU frequency changes was almost the same for the two algorithms, so more energy saving could be acquired at real environments.

Figure 11(right) shows the result for sporadic task system. In this case, our proposed algorithm also outperforms both DVSST and CC-DVSST. For DVSST, the ratio of energy saving is up to 70% and for CC-DVSST, up to 10%. The effect of number of tasks in the system could be also neglected, but the number of CPU frequency change of our algorithm was larger than that of DVSST and almost the same as that of CC-DVSST. So, the ratio of energy saving to DVSST could be shrunk. But our algorithm has huge performance gain to DVSST, so in spite of frequency change overheads, it is expected that our algorithm still outperforms to DVSST at real environments.

## 6. CONCLUSION

In this paper, we analyzed the behaviors of EDF scheduling and presented a power-aware scheduling algorithm for periodic and sporadic tasks. Temporal workload analysis provides another sufficient condition for schedulability of preemptive real-time task scheduling and another formal method to prove the correctness of power-aware scheduling algorithms. The proposed algorithm also adopts the results of cycle conserving method(CC-EDF) and sporadic task scheduling(DVSST). The simulation results show that the proposed algorithm outperforms existing algorithms up to 10-70 % with respect to CPU energy saving.

In the future we would like to improve the proposed algorithm. This could be done if we assign all slacks generated by early completed higher priority tasks into the task of highest priority among uncompleted ready tasks instead of evenly distributing them until the ends of deadlines. This method may lower processor frequency much more than the proposed algorithm. Also we would like to apply the temporal workload analysis into another area of real-time task scheduling, for example, aperiodic task acceptance problem. If we could maintain the temporal workload of a system below or equal to 1, then real-time constraints of all tasks in the system are met. So this may provide an effective method for aperiodic task scheduling.

## ACKNOWLEDGEMENTS

This work was supported by the Industrial Strategic Technology Development Program(10041740, Development of a software that provides customized real-time optimal control monitoring services by integrating equipments in buildings with web service) funded by the Ministry of Trade, Industry and Energy(MOTIE, Korea).

## REFERENCES

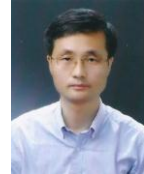
- [1] H. Aydin, R. Melhem, D. Mosse and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Trans. on Computers*, Vol. 53, pp 584-600, 2004.
- [2] T. D. Burd and R. W. Brodersen, "Energy efficient CMOS microprocessor design," In Proc. of Twenty-Eighth Hawaii Int'l Conf. on System Sciences, Vol. 1, 1995.
- [3] H. Chetto and M. Chetto. "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, Vol.15(10),pp.1261-1269, 1989.
- [4] W. Kim, J. Kim, and S. L. Min, "A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis," In Proc. of Design, Automation and Test in Europe, pp. 788-794, 2002.
- [5] C. H. Lee and K. G. Shin, "On-line dynamic voltage scaling for hard real-time systems using the EDF algorithm," In Proc. of IEEE Int'l Real-Time Systems Symposium, pp. 319-327, 2004.
- [6] J. Lee, A. Easwaran, I. Shin, and I. Lee, "Multiprocessor real-time scheduling considering concurrency and urgency," *ACM SIGBED Review*, Vol. 7(1), 2010.
- [7] C. L. Liu and J.W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *J. ACM* Vol.20(1), pp 46-61, 1973.
- [8] J. Mei, K. Li, J. Hu, S. Yin, and E. H-M Sha, "Energy-aware preemptive scheduling algorithm for sporadic tasks on DVS platform," *Microprocessors & Microsystems*, Vol.37, pp. 99-112, 2013.
- [9] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," *ACM SIGOPS Operating System Review*, Vol.35(5), pp. 89-102, 2001.
- [10] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," In Proc. of IEEE Int'l Real-Time Systems Symposium, pp 52-62, 2003.
- [11] D. Shin and J. Kim, "Dynamic voltage scaling of periodic and aperiodic tasks in priority-driven systems," In Proc. of the Asia and South Pacific Design Automation Conference, pp 653-658, 2004.
- [12] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Real-Time Systems*, Vol.10(2),pp.179-210, 1996.



- [13] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," In Proc. of the IEEE Foundations of Computer Science, pp.374–382, 1995.
- [14] RTSIM:Real-time system simulator, <http://rtsim.sssup.it>.

## AUTHORS

**Ye-In Seol** received his B.S. degree in Nuclear Engineering from Seoul National University, Korea in 1992, M.S. degree in Computer Science and Statistics from Seoul National University in 1994. He is a chief researcher in Green Energy Institute of SangMyung University in Seoul. His research interests include embedded system, real-time system and building automation system.



**Jeong-Uk Kim** received his B.S. degree in Control and Instrumentation Engineering from Seoul National University, Korea in 1987, M.S. and Ph.D. degrees in Electrical Engineering from Korea Advanced Institute of Science and Technology in 1989, and 1993, respectively. He is a professor in SangMyung University in Seoul. His research interests include smart grid demand response, building automation system, and renewable energy.



**Young-Kuk Kim** received the B.S. and M.S. degrees in Computer Science and Statistics from Seoul National University, Korea in 1985 and 1987 respectively and the Ph.D. degree in Computer Science from University of Virginia, Charlottesville, Virginia in 1995. After his Ph.D., he visited VTT Information Technology, Finland and SINTEF Telecom & Informatics, Norway as an ERCIM research fellow during 1995-1996. He joined the Chungnam National University as a faculty member of the Computer Science Department in March 1996. From August 2002 to July 2003, he visited Computer Science Department at University of California, Davis as an exchange scholar. His research interests include real-time systems, database systems, multimedia and mobile information systems

