Energy-Aware Real-Time Task Scheduling Exploiting Temporal Locality**

Yong-Hee KIM¹, Myoung-Jo JUNG¹, Nonmembers, and Cheol-Hoon LEE¹⁺⁻, Member

SUMMARY We propose a dynamic voltage scaling algorithm to exploit the temporal locality called TLDVS (Temporal Locality DVS) that can achieve significant energy savings while simultaneously preserving timeliness guarantees made by real-time scheduling. Traditionally hard real-time scheduling algorithms assume that the actual computation requirement of tasks would be varied continuously from time to time, but most real-time tasks have a limited number of operational modes changing with temporal locality. Such temporal locality can be exploited for energy savings by scaling down the operating frequency and the supply voltage accordingly. The proposed algorithm does not assume task periodicity, and requires only previous execution time among a priori information on the task set to schedule. Simulation results show that TLDVS achieves up to 25% energy savings compared with OLDS, and up to 42% over the non-DVS scheduling. key words: dynamic voltage scaling, low-power, real-time scheduling, temporal locality

1. Introduction

Energy consumption has become a key design requirement in real-time systems recently. This is especially important for battery-operated systems, such as cellular phones, unmanned robots, and ubiquitous sensor network nodes, because low energy consumption extends their limited battery life.

Traditionally, systems have been designed to operate at a fixed supply voltage with a fixed clock frequency. Recent advances in power supply and circuit design technologies allow the implementation of a microprocessor system that can adjust the operating voltage (and thus the clock frequency) at run time. The dynamic voltage scaling (DVS) technique takes advantage of the fact that lowering voltage can reduce power quadratically \(E \propto V^2\) to reduce the energy consumption [2]. Voltage is scaled down to an appropriate level whenever possible. Such variable-voltage systems can achieve extremely low power/energy consumption compared to the standard systems with fixed supply voltage. DVS has been a key technique for exploiting the hardware characteristics of processors to reduce energy dissipation by lowering the operating frequency and supply voltage. Also, DVS is coupled with the underlying operating system task management mechanism and real-time scheduler, and it can achieve significant energy savings, while simultaneously preserving timeliness guarantees made by real-time scheduling.

In the past decade, energy-efficient scheduling for real-time tasks on DVS systems has been widely explored by many researchers. Most of the DVS algorithms are offline scheduling algorithm, assuming that the systems provides some fixed set \(s\) of functions and that the release times and processor-time/resource demands of all its jobs are known in advance [1], [3], [5], [7], [10]–[12], [14], [16], [18]–[20]. Besides, researchers in [4], [6], [8], [13], [15], [17] consider on-line scheduling algorithms for a system whose future workload is unpredictable.

In this paper, we propose a DVS algorithm to exploit the temporal locality called TLDVS (Temporal Locality DVS). Traditionally, hard real-time scheduling algorithms assume that the actual computation requirement would be varied continuously from time to time. However, most real-time tasks have a limited number of operational modes changing with temporal continuity. For example, consider a cellular phone. If the phone receives or transmits calls, it is in a busy state. Otherwise, it is in an idle state. In other words, once it is in one of those states, it may probably still remain in the same mode at the next time instance. Thus, by observing the previous state, the current as well as future state can be predicted with a very high hit ratio. Such temporal locality can be exploited for energy savings by scaling down operating frequency and the supply voltage accordingly. However, since the state may alter at a random time instance, the DVS algorithm must be prepared to adapt to the change.

The key idea of the proposed algorithm is that each task executes at the lowest attainable frequency/voltage level by using the predicted computation time based on temporal locality, while guaranteeing feasible execution of all the upcoming tasks under the worst-case scenario. The TLDVS algorithm is based on our previous works [6], [8]. While most existing DVS algorithms focus on periodic tasks only, the proposed algorithm does not assume the periodicity of tasks, and nor requires any a priori information on the task sets to schedule. Also it requires only \(O(1)\) computation on each task context switch. This is very important to utilize temporal locality in real-time task scheduling.

The paper is organized as follows. In the next section, we present the system model considered in this paper and introduce the OLDS algorithm. Section 3 presents details of our TLDVS algorithm and illustrates how it works. The
simulation results are presented in Sect. 4, and Sect. 5 concludes the paper with a summary and discussion of future work.

2. Preliminaries

2.1 System Model

We focus on a preemptive hard real-time system in which real-time tasks are scheduled under the earliest-deadline-first (EDF) algorithm (i.e., the task with the earliest deadline is given the highest priority) [9]. A set of $n$ tasks is denoted by $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$, where tasks are assumed to be independent. Three parameters $\{r_i, C_i, d_i\}$ are used to represent each task $\tau_i$, where $r_i$ is the arbitrary release time, $C_i$ is the worst-case execution time (WCET) of the task at the maximum processor speed, and $d_i$ is the deadline.

We also define the start time, denoted by $s_i$, as the time at which task $\tau_i$ gets the control of CPU and starts to execute under worst-case scenario (all tasks execute in WCET at all times). For each task $\tau_i$, the actual execution time $e_i$ is defined as the time spent by CPU to complete execution of the task when operating at the maximum frequency. In many cases, the possibility of a task running at its WCET is very low. That is, although the real-time tasks are specified with worst-case execution requirements, they generally use much less than the worst-case (i.e., $e_i \leq C_i$). We assume that a constant amount of energy is required for each cycle of operation at a given supply voltage. Only the energy consumed by the processor is considered, variations as different types of instructions executed are not taken into account. Also we assume that energy isn’t consumed during idle time.

In current processors supporting the dynamic voltage scaling, the voltage level cannot be varied continuously. Instead, a limited number of voltage/frequency operating settings are usually available, causing the processor to run at a speed selectable within a discrete range. Therefore, we assume that the target processor can operate at several discrete frequency/voltage levels. The target variable voltage processor has discrete operating frequency and supply voltage levels and it is assumed to be able to scale its discrete operating frequency and supply voltage levels within its operational ranges, $[f_{\min}, f_{\max}]$ and $[v_{\min}, v_{\max}]$, respectively.

Let $\alpha(t)$ be a frequency scaling function such that the frequency is scaled down to $\alpha(t) \cdot f_{\max}$ at time $t$ ($0 \leq \alpha(t) \leq 1, \forall t$). Also, let $\alpha_i = f_i / f_{\max}$ be the current scaling factor which is the fraction of the current processor frequency $f_i$ over the maximum processor frequency $f_{\max}$. If the frequency is scaled down, the requested CPU time to complete each task will be increased. Therefore, we define the effective execution time, denoted by $e_i'$, as the time actually spent by CPU to complete $\tau_i$ under a frequency scaling $\alpha(t)$ (obviously, $e_i' \geq e_i$). We also assume that the voltage switching overhead is included into the WCET of each task.

2.2 The algorithm OLDVS

Consider a set $\Psi$ of real-time tasks whose loading factor $u_\Psi$ which is the maximum of the fraction of processor time possibly demanded by the task set in any interval of time, is 1. Since $u_\Psi = 1$, there is no slack time available on the schedule by EDF under the worst-case execution scenario in which every task takes the WCET for its execution as shown in Fig. 1.

Consider task $\tau_i$ in the example schedule, where $s_i'$ and $c_i'$ are the expected start time and the expected completion time for $\tau_i$ under the worst-case scenario, respectively. Since we assume preemptive scheduling, if $\tau_i$ is released before $s_i'$ (i.e., $r_i < s_i'$), task $\tau_i$ has higher priority than $\tau_i$ and is completed at $s_i'$ (i.e., $d_i \leq d_i'$ and $c_i' = s_i'$). If $r_i = s_i'$, $\tau_i$ is completed at $s_i'$, $\tau_i$ is a lower priority task and preempted by $\tau_i$ at $s_i'$. Also, tasks $\tau_j$ and $\tau_k$ in Fig. 1 are all higher-priority tasks and preempt $\tau_i$ at the times they release, respectively. So, the expected completion time $c_i'$ for each task $\tau_i$ under the worst-case scenario is calculated as follows:

$$c_i' = s_i' + C_i + \sum_{s_i' < j < s_i'} C_j.$$  \hspace{1cm} (1)

To guarantee the feasible execution of all the upcoming tasks under the worst-case scenario, each task $\tau_i$ must complete before or at $c_i'$. To do this, each task $\tau_i$ must also start before or at $s_i'$. We define the worst-case completion time of task $\tau_i$, denoted by $D_i$, as the latest time to complete, that guarantees the feasible execution of all the upcoming tasks. If an on-line algorithm estimates the worst-case completion time of each task to be less than or equal to its expected completion time under the worst-case scenario (i.e., $D_i \leq c_i'$, $\forall i$) and schedules it to complete before that, the algorithm guarantees the feasible execution. If task $\tau_i$ completes earlier than $c_i'$, the next task $\tau_j$ can use the unused execution time, effectively moving its start time earlier. These slack times can be exploited for energy saving by scaling down the operating frequency and the supply voltage accordingly.

In [8], we proved that each set of real-time tasks is feasibly schedulable by EDF if and only if $u_\Psi \leq 1$ under a certain frequency scaling. Where $u_\Psi$ is the absolute effective loading factor which is the maximum of all possible intervals. The main challenge in designing such algorithms is to ensure that deadline guarantees are not compromised when the operating frequencies are reduced. If the input task set $\Psi$ is feasible, we also proved that the OLDVS algorithm guarantees deadlines of all the tasks [8].

Figure 2 shows the OLDVS algorithm. For each context switch to task $\tau_i$, say at time $t$, the worst-case comple-
upon context switch to each task $\tau_i$ at time $t$:

$$\alpha_i = \text{calculate}_\alpha(t);$$
SetFrequency($\alpha_i$);

$\text{calculate}_\alpha(t)$
if $\tau_i$ preempted $\tau_j$ then /* $\tau_i = s_i = t$ and $d_i < d_j$ */
    $D_i = t + C_i$;
    $R_i = C_i$;
    $R_j = R_j - \alpha_i \cdot (t - l)$; /* $t$ : the previous context switch time */
else if $\tau_i$ resumes after some task $\tau_k$ then
    $D_i = D_i + D_k - t_p$; /* $\tau_i$ was preempted at $t_p$ */
else /* $\tau_i$ starts execution after some task $\tau_k$ */
    if ($d_k > d_i$ or $D_k < t$) then
        $D_i = t + C_i$;
    else $D_i = D_k + C_i$;
    $R_i = C_i$;
return SelectFrequency($R_i/(D_i - t)$);
/* $t$ : the return the scaling factor $\alpha_i$ */

SelectFrequency($\alpha$)
return min($f_1/f_m, \cdots, f_m/f_m | f_i/f_m \geq \alpha$)

SetFrequency($\alpha$)
$f = \alpha \cdot f_{max}$

Fig. 2  OLDVS algorithm.

ation time $D_i$ and the worst-case remaining time $R_i$ of task $\tau_i$ are calculated by $\text{calculate}_\alpha(t)$. As proved in [8], $D_i$ determined by $\text{calculate}_\alpha(t)$ is less than or equal to $C_i$. Thus, if each task $\tau_i$ completes before or at $D_i$, feasible execution of all the upcoming tasks are guaranteed. $R_i$ is the remaining CPU time to complete the task when operating at the maximum frequency under the worst-case scenario, also it is initially set to its WCET (i.e., $R_i = C_i$). When a task is preempted by another task, $R_i$ is reduced by the amount converted into maximum frequency CPU time from the previous context switch. The frequency scaling factor $\alpha_i$ is set as follows:

$$\alpha_i = \frac{R_i}{D_i - t}. \quad (2)$$

Then the operating frequency is selected to the smallest one larger than the scaling factor. The supply voltage, of course, is changed to match the operating frequency. By scaling the frequency above, each task completes before or at $D_i$, thus OLDVS guarantees the feasible execution of all the upcoming tasks under the worst-case scenario. No more than two operating frequency (and supply voltage) transitions can occur per task. Since the time complexity of the algorithm is $O(1)$, the scaling factor can be calculated at each context switch time with negligible overhead.

3. The Temporal Locality DVS

3.1 The Algorithm TLDVS

In the TLDVS algorithm, each task executes at the lowest attainable frequency/voltage level by using the expected computation time based on temporal locality. However, since the actual computation time may be larger than the expected computation time, the TLDVS algorithm must prepare to accommodate this situation. To do this, the proposed algorithm must schedule each task $\tau_i$ to complete before $D_i$, even if the worst case is exhibited.

In the TLDVS algorithm, we named $D_i$ of each task the given time budget. It is divided into two parts according to the transition of the frequency/voltage. Each task executes at the lowest attainable frequency/voltage level assuming $e_i \leq E_i$, where $E_i$ is the expected remaining time, in the first part. $E_i$ is set to the latest actual execution time $e_i$ of task $\tau_i$. At the end of the first part, a timer interrupt activates the real-time task scheduler. If the scheduler realizes $e_i > E_i$, then increases the frequency/voltage to complete task $\tau_i$ before $D_i$. Consequently, the length of the second part must be larger than or equal to the difference between the worst-case remaining time and the expected remaining time, namely $|R_i - E_i|$. The latest timer interrupt triggering time $t_{INT}$ and the scaling factor $\alpha_{i,1}$ for the first part are calculated as follows:

$$t_{INT} = D_i - (R_i - E_i) + t, \quad (3)$$

$$\alpha_{i,1} = \frac{E_i}{t_{INT} - t}. \quad (4)$$

Since we assume that the processor can operate at discrete voltage/frequency levels, the actual scaling factor $\alpha'_{i,1}$ for the first part is selected to the smallest one larger than the scaling factor (i.e., $\alpha'_{i,1} \geq \alpha_{i,1}$). Thus the actual timer interrupt triggering time $t'_{INT}$ is less than or equal to $t_{INT}$ (i.e., $t'_{INT} \geq t_{INT}$). It is calculated as follows:

$$t'_{INT} = \frac{E_i}{\alpha'_{i,1}} + t. \quad (5)$$

If $\tau_i$ does not complete until $t'_{INT}$, the scheduler is triggered by the timer interrupt. The scheduler increases the frequency to complete the remaining computation load until $D_i$. The scaling factor $\alpha_{i,2}$ for the second part is calculated as follows:

$$\alpha_{i,2} = \frac{R_i - E_i}{D_i - t'_{INT} + t}. \quad (6)$$

The actual scaling factor $\alpha'_{i,2}$ for the second part is calculated in the same way as $\alpha'_{i,1}$. Thus $t''_{INT}$, the actual finished time of task $\tau_i$, is calculated as follows:

$$t''_{INT} = \frac{R_i - E_i}{\alpha'_{i,2}} + t'_{INT}. \quad (7)$$

For simplicity, we illustrate the TLDVS algorithm using the examples as shown in Fig. 3. Consider, at time $t = 0$, task $\tau_i$ starts to execute with $C_i = 5$, $D_i = 10$, and $e_i = 1$.

The OLDVS algorithm fully utilizes the given time budget as shown in Fig. 3(a). It reduces power consumption by lowering the operating frequency/voltage to 50%. In spite of this good performance under the worst-case scenario,
OLDVS does not totally take advantage of the given time budgets since the task completes at time $t = 2$. However, in the case of TLDVS (see Fig. 3(b)), $t'_{\text{INT}} = 10 - 5 + 1 = 6$, $\alpha_{i,1} = 1/6 = 0.167$, $\alpha'_{i,1} = 0.25$, $t''_{\text{INT}} = 1/0.25 = 4$, $\alpha_{i,2} = (5 - 1)/(10 - 4) = 2/3$, $\alpha'_{i,2} = 0.75$, and $t''_{\text{INT}} = (5 - 1)/0.75 + 4 = 9.33$. The completion time of task $\tau_i$ is 4, since $e_i = 1$. Figure 3(b) shows obviously that the completion time of task $\tau_i$ less than its the worst-case completion time $D_i$.

Figure 4 shows the TLDVS algorithm. It is fairly easy to incorporate into a real-time operating system since the algorithm requires $O(1)$ computation.

3.2 The Illustration for TLDVS

We detail the TLDVS algorithm using the timing diagram as follows:

(a) If $\tau_i$ preempts some task $\tau_j$ (it means that $\tau_i$ has a higher priority than $\tau_j$), and $D_j$ is set to $t + C_j$. Also, $R_j$ and $E_j$ for task $\tau_j$ are recalculated as follows:

\[
R_j = C_j, \quad E_j = C_j - \alpha_j \times (t-l)
\]

(b) Else if $\tau_i$ resumes after some task $\tau_k$ (it means that $\tau_i$ was previously preempted at time $t_p$), then $D_i$ is set to $D_i + D_k - t_p$.

(c) and (d) Else if $D_k < t$ or $(d_i < d_k$ and $D_k \geq t)$, then $D_i$ is set to $t + C_i$.

(e) Else, $D_i$ is set to $D_i + C_i$ (i.e., $d_k \leq d_i$ and $D_k \geq t$).

Fig. 3 Examples scheduled by OLDVS and TLDVS.

Fig. 4 TLDVS algorithm.

upon context switch to each task $\tau_i$ at time $t$:

\[\alpha_i = \text{calculate}\_\alpha(t);\]
\[\text{SetFrequency}(\alpha_i);\]
\[\text{SetTimer}(E_i/\alpha_i);\]

\[\text{calculate}\_\alpha(t)\]
if $\tau_i$ preempted $\tau_j$ then
\[D_i = t + C_i;\]
\[R_i = C_i;\]
\[R_j = R_j - \alpha_j \times (t-l);\]
/* l : the previous context switch time */
\[E_j = E_j - \alpha_j \times (t-l);\]
/* update the expected remaining computation time */
else if $\tau_i$ resumes after some task $\tau_k$ then
\[D_i = D_i + D_k - t_p;\]
/* $\tau_i$ was preempted at $t_p$ */
else if $(d_k > d_i$ or $D_k < t$ ) then
\[D_i = t + C_i;\]
else
\[D_i = D_k + C_i;\]
\[R_i = C_i;\]
return \[\text{SelectFrequency}(E_i/(D_i - R_i + E_i - t));\]
\[\text{SetTimer}(t);\]
Set timer interrupt triggered at time $t_i$

\[\text{InterruptHandler}();\]
\[\text{SelectFrequency}((R_i - E_i)/(D_i - t))\]
As stated above, if the worst-case completion time for the task $\tau_i$ is less than or equal to its expected completion time ($i.e., C_i \leq c'_i$), the next task can use this unused execution time. These slack times can be exploited effectively for energy savings by lowering the supply voltage accordingly.

4. Simulation Results

We performed some simulation by using RTSIM [22] which is a real-time system simulator to evaluate the potential energy savings from voltage scaling in a real-time scheduling system. We show some simulation results and provide the most significant system parameters affecting energy consumption. The input data for simulation is a randomly generated task set specified with the release time, deadline, WCET and the actual computation workload for each task. The simulation assumes that a constant amount of energy is required for each cycle of operation at a given voltage. Only the energy consumed by the processor is computed, and variations due to different types of instructions executed are not taken into account.

The real-time task sets are specified using its period and worst-case computation time of each task. The periodic and aperiodic real-time task sets are generated randomly, each task has an equal probability of having a short (1~10 ms), medium (10~100 ms), or long (100~1000 ms) period. Task periods are uniformly distributed within each range.

We compared TLDVS to OLDVS, non-DVS, ccEDF, and laEDF [12]. For a fair and efficient comparison, for each test with ccEDF and laEDF, all aperiodic tasks are converted into a single periodic task such that its period and worst-case execution time are chosen so as to minimize to processor utilization while guaranteeing the deadlines of all tasks in each period. We call the converted task a periodic server that behaves like a periodic task and is created for the purpose of executing aperiodic tasks [8]. The computation requirements of the tasks are chosen such that the total processor utilization (including the periodic server for aperiodic tasks) becomes 1.

The practical processors considered are Intel PXA250 [21] and Transmeta TM5800 [23] which have discrete operating frequency and supply voltage levels. The following summarizes the hardware frequency and voltage levels, where each two-tuple $(f_i, v_i)$ consists of the frequency and the corresponding processor voltage:

- PXA250: (100,0.85), (200,1.0), (300,1.1), (400,1.3)
- TM5800: (300,0.8), (433,0.875), (533,0.95), (667,1.05), (800,1.15), (900,1.25), (1000,1.3).

The performance of the TLDVS algorithm is severely affected by how we can accurately predict the actual computation requirement of each task. There may be a lot of prediction methods. In this paper, we employ the Least Recently Used (LRU) scheme. We predict that the actual computation requirement of a task is equal to that of its previous instance.

Each task has strictly two distinct operation modes: IDLE and BUSY. The actual computation time of the IDLE mode is the best-case execution time (BCET), while that of the BUSY mode is the worst-case execution time (WCET). The modes are parameterized with the load ratio, BCET/WCET, and the transition probability. To determine the effects of temporal locality, we performed simulations varying the probabilities of both BUSY-to-IDLE transitions and IDLE-to-BUSY transitions.

Figure 5 shows the normalized energy consumption for task sets with 15 tasks for OLDVS and TLDVS. All of the simulations assume that the load ratio of task set is 10%. Also, we assume that the probability of IDLE-to-BUSY
transitions is 1% as shown in Fig. 5 (a) and the probability of BUSY-to-IDLE transitions is 50% as shown in Fig. 5 (b).

The simulation results show that TLDVS outperforms OLDVS. The performance gap becomes bigger as the probability of BUSY-to-IDLE transitions increases or the probability of IDLE-to-BUSY transitions decreases. However, when the probability of IDLE-to-BUSY transitions passes over a certain point (around 0.42~0.45 in Fig. 5 (b), the OLDVS slightly outperforms the TLDVS. This is caused by the prediction miss overhead of TLDVS.

In this paper, the proposed algorithm provides up to 25% energy savings compared with OLDVS, and up to 42% over the non-DVS scheduling algorithm. The performance of TLDVS becomes much larger as the temporal locality increases (see Fig. 5).

To see how the load ratio affects the performance of TLDVS, another simulation was performed assuming that the probabilities to IDLE-to-BUSY transitions and BUSY-to-IDLE transitions are fixed to 10% and 50%, respectively, while varying the load ratio. Figure 6 shows the result. As shown in Fig. 6, the difference of the performance between OLDVS and TLDVS becomes much larger as the load ratio decreases.

To compare TLDVS with ccEDF and laEDF, we performed the third simulation. In this simulation, each task set is composed of 10 tasks with the aperiodic factor of 0.5 while varying the load ratio. Aperiodic factor is the ratio of the computation requirement of the aperiodic tasks to the total computation requirement. Figure 7 shows the resulting normalized energy consumptions for the OLDVS, TLDVS, ccEDF, and laEDF algorithms. As shown in this figure, TLDVS outperforms ccEDF and laEDF as well as OLDVS in most cases. In this simulation, ccEDF and laEDF show a great increase in energy consumption compared with other two algorithms as the load ratio increases. With further simulations, we could see that the energy efficiency of ccEDF and laEDF deteriorates as the aperiodic factor increases. That is because the ccEDF and laEDF algorithms are focused only on periodic tasks. In particular, since laEDF tries to defer as much work as possible, they need to run at high frequencies later in order to complete all of the deferred work in time. On the other hand, since TLDVS does not assume the periodicity of real-time tasks, the aperiodic factor does not affect its energy efficiency.

5. Conclusions

In this paper, we presented a novel energy-aware real-time task scheduling algorithm exploiting temporal locality of the operation mode. While most existing DVS algorithms focus on periodic tasks only, the proposed algorithm does not assume the periodicity of tasks, and nor requires any a priori information on the task sets to schedule. Also the proposed algorithm requires only $O(1)$ computation on each context switch, so it is fairly easy to incorporate into a real-time operating system. Simulation results show that the TLDVS algorithm achieves up to 25% energy savings compared with the OLDVS algorithm, and up to 42% over the non-DVS scheduling. The performance gap becomes bigger as the temporal locality increases.

In the future, we could like to expand this work beyond the deterministic/absolute real-time paradigm presented here. In particular, we would like to investigate DVS with probabilistic or statistical deadline guarantees. We will also explore integration with other energy-conserving mechanisms, including application energy adaption and energy-
adaptive communication (both real-time and best-effort).

References


Yong-Hee Kim received the M.S. degree in computer engineering from Chungnam National University, Daejeon, Korea in 2003. Since 2003, she has been working toward the Ph.D. degree in System Software Laboratory in Chungnam National University. Her research interests include embedded systems, fault-tolerant system, real-time scheduling, and real-time operating systems.

Myoung-jo Jung received the M.S. degree in computer engineering from Chungnam National University, Daejeon, Korea in 2003. Since 2003, he has been working toward the Ph.D. degree in System software Laboratory in Chungnam National University. His research interests include java virtual machine, real-time scheduling, and real-time operating systems.

Cheol-Hoon Lee received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1983 and the M.S. and Ph.D. degrees in computer engineering from KAIST, Daejeon, Korea, in 1988 and 1992, respectively. From 1983 to 1994, he worked for Samsung Electronics Company in Seoul, Korea, as a researcher. From 1994 to 1995 and from 2004 to 2005, he was with the University of Michigan, Ann Arbor, as a research scientist at the Real-time Computing Laboratory. Since 1995, he has been a professor in the Department of Computer Engineering, Chungnam National University, Daejeon, Korea. His research interests include parallel processing, operating system, real-time system, and fault tolerant computing.