Sustainable Computing: Informatics and Systems 18 (2018) 137–148

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Energy-efficient scheduling with reliability guarantee in embedded real-time systems

Hongzhi Xu [a,b,*], Renfa Li [a], Lining Zeng [a], Keqin Li [a,c], Chen Pan [d]

[a] College of Information Science and Engineering, Hunan University, Changsha 410082, China
[b] College of Software and Service Outsourcing, Jishou University, Zhangjiajie 427000, China
[c] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
[d] School of Electrical and Computer Engineering, Oklahoma State University, OK 74078, USA

ARTICLE INFO

Article history:
Received 24 January 2016
Received in revised form 16 January 2018
Accepted 22 January 2018
Available online 31 January 2018

Keywords:
Energy efficiency
Processor utilization
Real-time system
Reliability
Task scheduling

ABSTRACT

Embedded systems have been widely applied in many areas such as aerospace, intelligent transportation, smart grid, and, medical care. However, limited energy supply and runtime faults are two challenges that hinder the embedded system to be even more pervasively applied. In this paper a greedy energy-efficient (GEE) algorithm is proposed to improve the energy efficiency under the constraint of reliability. As GEE may lead to more preemptions with reduced execution frequency, this paper analyzes the preemption during the execution and proposes a series of constraints to reduce the frequency of preemption. This reduces both the energy consumption and execution time. To further balance the execution frequency, two energy-efficient algorithms are presented based the processor utilization. Finally, considering the fact that the actual execution time (AET) of many tasks may be less than the worst-case execution time (WCET), a global dynamic scheduling algorithm is proposed to maximize the energy efficiency while ensuring the reliability. The experimental results show that the proposed techniques have significant improvements of energy efficiency with guaranteed system reliability.

1. Introduction

Embedded systems have been widely applied in many areas due to its promising features such as high speed, low cost, or easy of control [1–3]. These areas include, aerospace, intelligent transportation, smart grid, smart home, medical care, and health monitoring of large buildings. Embedded system greatly facilitates the computational, communicational, and commercial requirements in these fields.

Embedded system, however, still have two challenges that prevent it from being even more pervasively applied [4,5]. First, the power sources of embedded system are generally power constrained such as battery and energy harvester [6]. Second, faults are likely to occur on embedded system due to unstable power or outside interferences. These two challenges become even stronger in recently popularized cyber-physical system (CPS) [7] where embedded system, as a core component, requires higher precision, more reliable communication, and longer lifetime. Therefore, in order for embedded system to be well applicable in CPS, these two challenges need to be addressed.

To overcome the challenge of power constraint, many works have targeted on improving energy efficiency of embedded system. Many technology, such as dynamic power management (DPM) and dynamic voltages frequency scaling (DVFS), which reduce the system energy consumption by adjusting the voltage and frequency in runtime, have been researched by some hardware producers and researchers. They provide viable methods to efficiently reduce the system energy consumption by adjusting the runtime voltage and frequency. Jejurikar et al. [8] presented a parameter critical speed to determine how to adjust the processor voltage. Kavousianos et al. [9] considered the test application of multicore processor, present a fast heuristic test scheduling technique to optimize the voltage setting. Tokarnia et al. [10] taken soft delay deadline strategy, present some path-based dynamic voltage and frequency scaling algorithms. Kim et al. [11] used voltage scaling manage the resource dynamically in heterogeneous ad-hoc grid, optimize the energy consumption and system performance. Li et al. [12] considered

* Corresponding author at: College of Information Science and Engineering, Hunan University, Changsha 410082, China.
E-mail addresses: xuhongzhi9@163.com (H. Xu), lirenfa@vip.sina.com (R. Li), will120120@126.com (L. Zeng), lik@newpaltz.edu (K. Li), chen.pan@okstate.edu (C. Pan).

https://doi.org/10.1016/j.suscom.2018.01.005
2210-5379/

the nonpreemptive situation, design nonpreemptive scheduling algorithms that ensure timing correctness and optimize energy consumption on a processor with variable speeds. Mishra et al. [13] presented energy-efficient load scheduling algorithm (EELSA) to minimize the energy consumption for a given computational load that has to be completed within a given deadline. Lee [14] proved that the best energy efficiency scheduling under the deadline restrain in multi-core system is NP-hard. Li et al. [15] studied the scheduling a bag-of-tasks application, balance the constrain of energy consumption and real-time constrain for stochastic tasks on heterogeneous computing system. Ding et al. [16] presented an energy-efficient scheduling algorithm of virtual machines (VMs) in cloud considering the deadline constraint. The algorithm computed the optimal frequency of each active core according to the sum of the required resources of the VMs on it. Aldahari et al. [17] calculated and tuned to the best CPU frequency for each running task combined with round robin and first fit scheduling algorithms, which reduced CPU frequency without affecting the Quality of Service. However, energy efficiency techniques in most of their works require to recycle the slack to lower the frequency of tasks, especially the uncritical tasks, or to close some processors. This may likely to increase unreliability of embedded system [18].

Since energy efficiency techniques may cause unreliable issues to embedded system, many researches have focused on improving energy efficiency while at the same time guaranteed the embedded system reliability. Zhu et al. [19] proposed the reliability-aware power management (RA-PM) for real-time periodic tasks by lowering the execution efficiency in the slack time. This reduced the energy consumption. Zhu et al. in [20,21] also used check pointing technology to divide the tasks with long execution time into several short tasks. This both guarantees the reliability and optimizes the energy. Wei et al. [22] presented a reliability-driven task scheduling scheme for real-time embedded systems by considering both the dynamic power and the leakage power. The scheme was able to optimize system energy consumption under stochastic fault occurrences. Lin et al. [23] presented the leakage control dynamic (SUF-DL) algorithm to guarantee reliability at least as high as that of without speed scaling. Li et al. [24] minimized the energy consumption and guarantee the reliability of real-time task set based on frame. Zhao et al. [25] presented SHR algorithm using shared recovery technique based on reliability-aware power management schemes. SHR considered a set of frame-based real-time tasks with individual deadlines and a common period where the precedence constraints are represented by a directed acyclic graph (DAG). Zhang et al. [26] present reliability maximization with energy constraint (RMEC) algorithm that maximizes the reliability with the restrain of energy efficiency based on the DAG, which includes three stages of priority confirm, frequency selection and processor distribution. Xiang et al. [27] considered variations in solar radiance, execution time, and transient faults, and proposed a hybrid design-time and run-time framework for resource allocation for solar energy harvesting system. While these works provide reliability ensured power efficient techniques through voltage and frequency adjustment, it is equally important to consider about the task preemption which may also cause embedded system to become unreliable.

The preemption of tasks is actually common in application. In fact, it consumes time and energy during the preemption caused by context switches and cache disable [28,29]. This may cause some tasks miss the constraint of deadline and consume more energy [30]. Therefore it is equally important to improve energy efficiency during task preemption while at the same time ensure the reliability of embedded system. This paper targets on task scheduling for a set of periodic tasks. Based on EDF strategy, the proposed techniques are able to maximize the energy efficiency with a guarantee of system reliability.

The major contributions of this paper include the following.

- This paper collects the slack time by defining a periodic virtual task. Based on the slack time, a greedy energy-efficient algorithm is proposed with the guaranteed reliability.
- This paper designs two kinds of energy-efficient algorithm based on processor utilization in order to balance the execution frequency of released jobs.
- This paper reduces the number of preemption, which optimizes the execution time and energy consumption.
- With the consideration that the actual execution time of many tasks may be less than the WCET, this paper design a global dynamic energy efficient algorithm based on the processor utilization.

The experimental results show that the proposed techniques guarantee system reliability and have significant improvements over other related techniques in terms of energy efficiency.

The rest of the paper is organized as follows. Section 2 introduces models that are used in this paper. Section 3 presents related properties. Section 4 presents detailed algorithms of reliability constraints and energy efficiency. Simulation results are shown in Section 5. Finally, section 6 concludes this work.

2. System model and problem formulation

In this section, system model will first be presented including three sub-models which are task model, energy consumption model, and fault model. After that the problem which motivates this work will be formulated.

2.1. Task model

A task set consists of n periodic tasks as $T = \{T_1, T_2, \ldots, T_n\}$ on a uniprocessor system with a single line of execution being considered. The voltage/frequency of processor is adjustable. Each task T_i is characterized by (C_i, P_i, D_i), where C_i is the worst case execution time (WCET) of T_i when the processor is running in the maximum frequency, P_i is the period of T_i, and D_i is the relative deadline. Like most recent works on periodic task scheduling [19,31], $D_i = P_i$ is assumed. Each task T_i releases the jth job $J_{i,j}$ at the time instant $(j-1)P_i$, with the deadline $d_{i,j} = (j-1)P_i + D_i = j \cdot P_i$. If the execution time of $J_{i,j}$ is *exetime* when the processor runs in the maximum frequency $f_{\max}$ (in this paper, the maximum frequency be defined as 1, that is $f_{\max} = 1$). When the processor run in the frequency f, the execution time of $J_{i,j}$ is $exetime \times \frac{f_{\max}}{f}$. If all the tasks in the task set can meet their deadline, the task set T is schedulable, or else T is non-schedualable.

If the processor utilization of task T_i is $U_i = \frac{C_i}{P_i}$, the utilization of the whole system is $U_{sys} = \sum_{i=1}^{n} \frac{C_i}{P_i}$, when $U_{\text{sys}} \leq 1$, the task set T is schedulable [32].

2.2. Energy consumption model

In this paper, the case where the frequency of the processor can be continuously adjusted in the range of $[f_{\min}, f_{\max}]$ is considered [12,20,21]. According to the related works about the energy consumption model from [19–21,23,26], the energy function of CMOS chip is given by

$$P(f) = P_{\text{s}} + \hbar(P_{ind} + P_{\text{d}}) = P_{\text{s}} + \hbar(P_{ind} + C_{ef} f^m). \quad (1)$$

In (1), P_{s} is the static power consumption, which is used to maintain the basic circuit and the clock. P_{s} would not be 0 until the system is power off. P_{ind} is the leakage power consumption unrelated to the frequency, which is a constant and can be eliminated

when the processor is sleep. P_d is the dynamic consumption caused by charge/discharge of gate circuit, which is related to the processor frequency. The parameter m is a number greater than 2 [19]. Based on [19–21,26], in this paper the parameter m is set to $m=3$. In (1), $\hbar$=1 means that the processor is active, and $\hbar$=0 means that the processor is power off or sleep. When the system is active, the power consumption of system is given by

$$P(f) = C_{ef} f^m + \beta,$$

where

$$\beta = P_s + P_{ind} > 0. \tag{2}$$

In time interval $[t_1, t_2]$, when the processor is running in frequency f, the energy consumption is given by

$$E(f) = \int_{t_1}^{t_2} P(f)dt. \tag{3}$$

Therefore, the energy consumption of the processor running at frequency f for a single period is

$$E = (C_{ef} f^m + \beta)/f.$$

Taking the first order derivative of E, we have

$$(m-1)C_{ef} f^{m-2} - \beta f^{-2}.$$

If $(m-1)C_{ef} f^{m-2} - \beta f^{-2} = 0$, The frequency with minimum energy consumption is $f_{critical} = \sqrt[m]{\frac{\beta}{C_{ef}(m-1)}}$, When the processor runs in frequency less than $f_{critical}$, the energy consumption would be more because of the longer execution time. Therefore, to save energy, the frequency of the processor is within the range of $[f_{critical}, f_{max}]$.

2.3. Fault model

Reliability is the probability of that a job can be executed correctly before deadline [19]. It is important for most of real-time systems such as aerospace and medical systems. The runtime faults can be divided into transient faults and permanent faults. In CMOS chips the lower frequencies are accompanied by lower voltage levels and, as a result, become more vulnerable to transient faults caused by cosmic particles [20,23]. Therefore, the transient faults are more frequent [18,33], and this paper just considers the transient faults. According to [19–21,23,26], the transient fault rate of processor in frequency f (corresponding to voltage v) is given by

$$\lambda(f) = \lambda_0 g(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{min}}}. \tag{4}$$

where d is a constant greater than 0, λ_0 is the average transient fault rate in the max frequency. The lower the frequency is, the more transient faults are, so when $f < f_{max}$, $g(f) > 1$. When the processor runs in the minimum frequency, the system has the biggest fault rate, where $\lambda(f_{min}) = \lambda_0 10^{\frac{d(1-f_{min})}{1-f_{min}}} = \lambda_0 10^d$.

The reliability of system is the rate of which system run correctly, if the runtime transient fault follows Poisson distribution, the probability that task T_i run correctly is: $R_i = e^{-\lambda(f_i)\frac{c_i}{f_i}}$ [20,21,23]. When T_i is executed in the maximum frequency, the probability is $R_i^0 = e^{-\lambda_0 C_i}$. If all the n tasks run in the maximum frequency, the reliability of system is $R_0 = \Pi_{i=1}^n R_i^0$.

Similar to [19], the transient fault is detected at the completion of a job's execution. If a transient fault occurs, the job will be re-executed. In this paper, the overhead of fault detection has been considered into WCETs for each task.

2.4. Problem formulation

Consider a processor system and a task set $T = \{T_1, T_2, \ldots, T_n\}$, where T is EDF schedulable with the reliability R_0 when the processor always run in the maximum frequency and the energy consumption is regardless. If DVFS is used to adjust the frequency, the transient faults rate will be bigger and the reliability will be less. The challenge of this paper is how to optimize the energy efficiency with DVFS while guaranteeing the reliability of the embedded system is not less than R_0.

3. Related properties

In this section, two theorems will be presented to provide theoretic supports for this work.

Theorem 1. *Without the consideration of deadline, if n tasks are executed in a certain time interval, and the execution frequencies of n tasks are $f_1, f_2, \ldots, f_n$, respectively. when $f_1 = f_2 = \cdots = f_n \geq f_{critical}$, the system has the minimum energy consumption (the processor may be off if $f < f_{critical}$).*

Proof. We apply the contrapositive to prove the Theorem 1. If Theorem 1 is not correct, there may exist less energy consumption when some tasks are executed in unequal frequency. Assuming task i and task j are executed in frequencies f_i and f_j ($f_i \neq f_j$) with the execution times t_i and t_j, respectively. Without loss of generality, it is assumed that $f_i > f_j$. In this condition, the total of energy consumption of task i and task j is given by

$$\begin{aligned} E(f_i \neq f_j) &= p(f_i)t_i + p(f_j)t_j \\ &= (C_{ef} f_i^m + \beta)t_i + (C_{ef} f_j^m + \beta)t_j. \end{aligned} \tag{5}$$

Now consider that task i and task j execute at the same frequency f, and their execution times are t_i' and t_j', respectively. In this condition, the total of energy consumption of task i and task j is given by

$$\begin{aligned} E(f) &= p(f_i)t_i' + p(f_j)t_j' \\ &= p(f)(t_i' + t_j'). \end{aligned} \tag{6}$$

Consider that the execution time and frequency of other tasks remain unchanged and the execution time of task i and task j is also constant, that is $t_i' + t_j' = t_i + t_j$, so $f = \frac{f_i t_i + f_j t_j}{t_i + t_j}$. When task i and task j are executed in the same frequency f, the energy consumption is given by

$$E(f) = (C_{ef} f^m + \beta)(t_i + t_j). \tag{7}$$

According to (5), (7) and $f = \frac{f_i t_i + f_j t_j}{t_i + t_j}$, the following equation can be constructed:

$$\frac{(E(f_i \neq f_j) - E(f))}{C_{ef}} = f_i^m t_i + f_j^m t_j - \frac{(f_i t_i + f_j t_j)^m}{(t_i + t_j)^{m-1}} \tag{8}$$

According to the Holder Inequality [34], for real numbers p and q, $p \geq 1$, $q < +\infty$ and $\frac{1}{p} + \frac{1}{q} = 1$, and any real numbers or complex numbers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, it is always true that:

$$\begin{aligned} &|a_1 b_1| + |a_2 b_2| + \cdots + |a_n b_n| \\ &\leq (|a_1|^p + |a_2|^p + \cdots + |a_n|^p)^{\frac{1}{p}} \\ &\times (|b_1|^q + |b_2|^q + \cdots + |b_n|^q)^{\frac{1}{q}} \end{aligned} \tag{9}$$

If $k>1$, $p=k+1$, $q=\frac{k+1}{k}$, $a_i=\frac{x_i}{\frac{k}{y_i^{k+1}}}>0$ and $b_i=y_i^{\frac{k}{k+1}}>0$, the first two elements of inequality (9) are

$$x_1+x_2\le\left(\frac{x_1^{k+1}}{y_1^k}+\frac{x_2^{k+1}}{y_2^k}\right)^{\frac{1}{k+1}}(y_1+y_2)^{\frac{k}{k+1}}. \tag{10}$$

Exponentiation by $(k+1)$ on both sides of (10), we get:

$$(x_1+x_2)^{k+1}\le\left(\frac{x_1^{k+1}}{y_1^k}+\frac{x_2^{k+1}}{y_2^k}\right)(y_1+y_2)^k \tag{11}$$

Inequality (11) divided by $(y_1+y_2)^k$, yields:

$$\frac{(x_1+x_2)^{k+1}}{(y_1+y_2)^k}\le\frac{x_1^{k+1}}{y_1^k}+\frac{x_2^{k+1}}{y_2^k} \tag{12}$$

Now, (8) can be written as

$$\frac{(E(f_i\ne f_j)-E(f))}{C_{ef}}=\frac{{f_i}^m{t_i}^m}{{t_i}^{m-1}}+\frac{{f_j}^m{t_j}^m}{{t_j}^{m-1}}-\frac{(f_it_i+f_jt_j)^m}{(t_i+t_j)^{m-1}}. \tag{13}$$

In (13), if $f_it_i=x_1, f_jt_j=x_2, t_i=y_1$, and $t_j=y_2$. Because $m>2$, according to (12), the inequality $E(f_i\ne f_j)>E(f)$ can be deduced, which is conflict with hypotetical. So Theorem 1 is proved.

Theorem 2. *Assume that the n tasks execute on a uniprocessor system, and the transient fault follows the Poisson distribution. Once the transient faults occur to tasks with turned down execution frequencies, the reliability of system will not be dropped below R_0 if these tasks are executed again with the maximum execution frequency.*

Proof. When all the n tasks are executed with the maximum frequency, according to the fault model mentioned in Section 2.3, the reliability of system is $R_0=\Pi_{i=1}^n R_i^0$. When energy efficiency has been optimized by DVFS, considering there are $x(x\in X)$ tasks whose frequency are reduced, the other $y(y\in Y)$ tasks are still executed with the maximum frequency, so $T=X\bigcup Y$ and $X\bigcap Y=\phi$. Therefore, the reliability of the system can be expressed as $R=R_XR_Y$. If k tasks in X encounter transient faults, then these faulted tasks will be re-executed with maximum frequency and stored into task set $K(K\subseteq X)$. So, R_X is given by

$$R_X=\Pi_{x\in K}R_x^0\ge\Pi_{x\in X}R_x^0. \tag{14}$$

$$R_Y=\Pi_{y\in Y}R_y^0. \tag{15}$$

Therefore, the reliability of system is $R=R_XR_Y>R_0$

4. Reliability constraint and energy-efficient schedule

In this section, greedy energy-efficient scheduling (GEE) algorithm is proposed to minimize the execution frequency while guaranteeing reliability by using slack time. The GEE algorithm assigns slack time greedy to the current job, which will lead to the imbalance of the tasks' execution frequency. Therefore, other three algorithms are proposed to further balance tasks' execution frequency. The first two algorithms are based on processor utilization and the last one is based on actual execution time.

4.1. Greedy energy-efficient scheduling algorithm (GEE)

The key to reduce energy consumption under the reliability constraint is to fully utilize slack time. During slack time, the execution frequency of each task will be adjusted to minimize energy consumption while guranteeing the task is able to be finished before the deadline. Consider that all the tasks execute for their WCET in the maximum frequency $f_{\max}$. If the system utilization $U_{sys}=1$, no task can be turned down. If $U_{sys}<1$, the slack time can be collected in the runtime, and turn down the frequency of some tasks. In this way the task must be ensured to re-execute in the maximum frequency when transient faults occur. To get the available slack time *slackTime*, a virtual task *VRT* with the period $P_{VRT}=\min_{i\le1\le n}\{P_i\}$ and the $C_{VRT}=(1-\sum_{i=1}^n\frac{C_i}{P_i})P_{VRT}$ is constructed. C_{VRT} is the actually slack time which is generated periodic and can be used by real runtime tasks. Assume that *curJob* is current executing job and the slack time would decrease/increase under following situations: (1) *curJob*'s frequency f_{curJob} is reduced; (2) transient faults occur and *curJob* is recovered; (3) *curJob* completes ahead of WCET; (4) the processor is idle. Based on the above situation, the following 5 rules are defined to manage *slackTime*:

- Rule 1: *slackTime* will increase C_{VRT} when a *VRT* arrives.
- Rule 2: When *curJob* is completed correctly with frequency f_{curJob}, the *slackTime* will decrease $\frac{C_{curJob}}{f_{curJob}}-\frac{C_{curJob}}{f_{\max}}$.
- Rule 3: When the frequency is reduced and transient fault occurs during the execution of *curJob*, *curJob* must be re-executed, the *slackTime* will decrease $\frac{C_{curJob}}{f_{curJob}}$.
- Rule 4: When *curJob*'s actual execution time is less than the WCET, *curJob* is completed ahead, the *slackTime* will increase $\frac{C_{curJob}}{f_{\max}}-AET_{curJob}$. Here AET_{curJob} is actual execution time of *curJob*.
- Rule 5: If processor is idle, the *slackTime* decrease 1 for every time unit.

The energy consumption can be optimized by turning down the frequency based on the *slackTime*. A greedy energy efficiency scheduling algorithm is presented under the constraint of reliability as Algorithm 1 GEE.

Algorithm 1. GEE

Input: *TaskSet*
Output: Set the execution frequency of the task and execute

construct a *VRT*
at each time tick, do
if *VRT* arrives **then**
  $slackTime \longleftarrow slackTime + C_{VRT}$
end if
if a frequency turned down task fault **then**
  $slackTime \longleftarrow slackTime - \frac{C_{curJob}}{f_{curJob}}$
  turn the frequency to maximum and re-execute *curJob*
end if
if *curJob* completed correctly **then**
  $slackTime \longleftarrow slackTime - C_{curJob}\times\frac{f_{\max}-f_{curJob}}{f_{curJob}}$
  if *ReadyQ* is not empty **then**
    curJob$\longleftarrow$ get task from *ReadyQ*
    SetFrequency1(*curJob*, *slackTime*) and execute *curJob*
  else
    processor sleep
    $slackTime \longleftarrow slackTime - 1$
  end if
end if
if *newJob* arrives **then**
  if processor is idle **or** $D_{newJob}<D_{curJob}$ **then**
    if $curJob \ne NULL$ **then**
      insert Rem_{curJob} into *ReadyQ*
      $slackTime \longleftarrow slackTime - (C_{curJob}-Rem_{curJob})\times\frac{f_{\max}-f_{curJob}}{f_{curJob}}$
    end if
    $curJob \longleftarrow newJob$
    SetFrequency1(*curJob*, *slackTime*) and execute *curJob*
  else
    insert *newJob* into *ReadyQ*
  end if
end if

In GEE, *curJob* represents current executing job. C_{curJob} and D_{curJob} represent WCET and absolute deadline of *curJob*, respectively. Rem_{curJob} denotes the remaining time requirement of *curJob*.

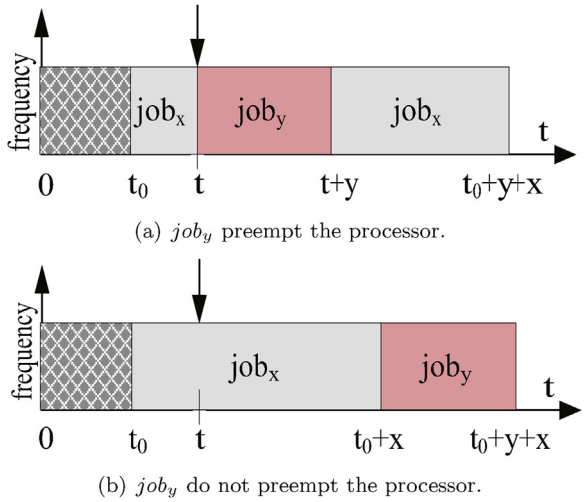


Fig. 1. The situation when high priority job arrive.

ReadyQ denotes the ready queue. The GEE algorithm uses EDF as the basic scheduling mechanism and adjusts the processor execution frequency based on the *slackTime*. The GEE algorithm can be divided into 4 parts. Lines 3–5 are the first part, in which *slackTime* will increase when *VRT* arrives. The second part is lines 6–9. In this part, if a *curJob*'s frequency is turned down and some transient fault occurs, *curJob* will be re-executed in maximum frequency. Lines 10–19 are the third part, when *curJob* is finished correctly, the processor executes the next job in the ready queue if it is not empty. Otherwise, the processor sleeps. The fourth part is Lines 20–31, when *newJob* arrives, GEE judges its priority. If the priority is greater than *curJob* or the processor is idle, GEE adjusts the frequency for the *newJob* and executes this *newJob*. In this way, preemption may occur during which the preempted *curJob* is added to the ready queue or the *newJob* is added to the ready queue. All the four parts manage the *slackTime* based on the rules mentioned above and turn down the frequency of *curJob* as low as possible. The algorithm for adjusting frequency is shown in Algorithm 2 (SetFrequency1).

Algorithm 2. SetFrequency1

Input: *curJob* and *slackTime*
Output: set the execution frequency of *currJob*

1: $recovertime \longleftarrow \frac{C_{curJob}}{f_{\max}}$
2: $s \longleftarrow D_{curJob} - recovertime - currentTime$
3: $s \longleftarrow \min(s, slackTime)$
4: **if** $s \leq recovertime$ **then**
5: $f_{curJob} \longleftarrow f_{\max}$
6: **else**
7: $f_{curJob} \longleftarrow \frac{C_{curJob}}{s}$
8: **end if**

In GEE, when several jobs have the same deadline, the longer execution time a job has, the greater priority it gets. This mechanism can balance the execution frequency, because if a job with longer execution time can be executed in lower frequency, faults recovery time would has been reserved which is no less than the slack time demanded by the next job.

When a job with higher priority arrives, the processor preemption happens. The context switch in preemption is used to save the state information of current executing job, which will lead to more time and energy consumption. Actually, preemption is not always necessary. As shown in Fig. 1, in the figure, the *X*-axis represents time and the *Y*-axis represents execution frequency. job_x is begun to execute in t_0, and is supposed to finish in $t_0 + x$ (x is the WCET of job_x). In instant t, a job job_y (y is the WCET of job_y) with higher priority arrives, the processor is preempted by job_y based on the EDF strategy (shown in Fig. 1(a)). Consider that during the execution of job_x and job_y, no other job with priority higher than job_x arrives. If If no preemption occurs, the job_y is delay still meets the deadline, as shown in Fig. 1(b), the energy consumption will be less because of the less preemption and the scheduling is also feasible. Hence, the delay of jobs may lead to less preemption which make the scheduling simpler.

GEE is based on the greedy strategy, which will lead to the imbalance of execution frequencies of other tasks and system energy consumption also cannot be minimized. To alleviate this problem, if more tasks are executed with reduced frequencies, the overall energy consumption can be reduced dramatically. Therefore, a greedy energy-efficient algorithm based on processor utilization is presented which considers all the tasks in the system.

4.2. Greedy energy-efficient scheduling based on processor utilization (GEEPU)

Thinking intuitively from the task level, if a task T_i has the utilization $U_i < 1 - U_{sys}$, this task's frequency can be turned down. Consider a task set $T_{f_{low}}$ which can be executed in lower frequency, the total utilization of the set is $U_{f_{low}} = \sum_{U_i < 1-U_{sys}} U_i$. This paper also considers a task set $T_{f_{\max}} = T - T_{f_{low}}$ which executes in maximum frequency $f_{\max}$. The total utilization of $U_{f_{\max}} = \sum_{U_j \geq 1-U_{sys}} U_j$, according to Theorem 1, if the transient faults can be ignored, the processor executes $T_{f_{\max}}$ with the maximum frequency and executes $T_{f_{low}}$ with the same lower frequency $f_{low} = \frac{U_{f_{low}}}{1-U_{f_{\max}}}$ for the rest of the time, the energy consumption would be minimum. In the runtime, when a job can be executed in the frequency $f \leq f_{\text{low}}$, the frequency is turned to $\frac{(f+f_{low})}{2}$.

When setting the task execution frequency, consider that the *VRT* may arrive in runtime and increase the *slackTime*, which can lower the frequency of the task. So the function SetFrequency1 can be modified as Algorithm 3 (SetFrequency2):

Algorithm 3. SetFrequency2

Input: *curJob* and *slackTime*
Output: set the execution frequency of *currJob*

1: $recovertime \longleftarrow \frac{C_{curJob}}{f_{\max}}$
2: $s \longleftarrow D_{curJob} - recovertime - currentTime$
3: $b \longleftarrow$ Judge whether the next *VRT* arrives before $currentTime + \frac{C_{curJob}}{f_{\max}}$
4: $s \longleftarrow \min(s, slackTime + b \times C_{VRT})$
5: **if** $s \leq recovertime$ **then**
6: $f_{curJob} \longleftarrow f_{\max}$
7: **else**
8: $f_{curJob} \longleftarrow \frac{C_{curJob}}{s}$
9: **if** $f_{curJob} < f_{\text{low}}$ **then**
10: $f_{curJob} \longleftarrow \frac{f_{curJob} + f_{low}}{2}$
11: **end if**
12: **end if**

Sometimes, $T_{f_{low}}$ cannot be constructed for some task sets, and if that happens, $f_{\text{low}} = 0$. In this condition, the algorithm SetFrequency2 will not be executed for Lines 9–11, SetFrequency2 is actually similar to the SetFrequency1.

This paper also considers the preemption simply in GEEPU. When jobs with higher priority arrive, whether jobs can be delay is analyzed, which can lower the preemption.

4.3. Example

Consider a task set $T = \{T_1(P_1 = 7,\ C_1 = 2),\ T_2(P_2 = 7,\ C_2 = 1),\ T_3(P_3 = 7,\ C_3 = 1),\ T_4(P_4 = 14,\ C_4 = 2)\}$, the system utilization $U_{\text{sys}} = 71.4\%$. When the processor runs in the maximum frequency, we can construct a $VRT(P = 7,\ C = 2)$ based on Section 4.1. In a hyperperiod (the lowest common multiple of all task's period), the

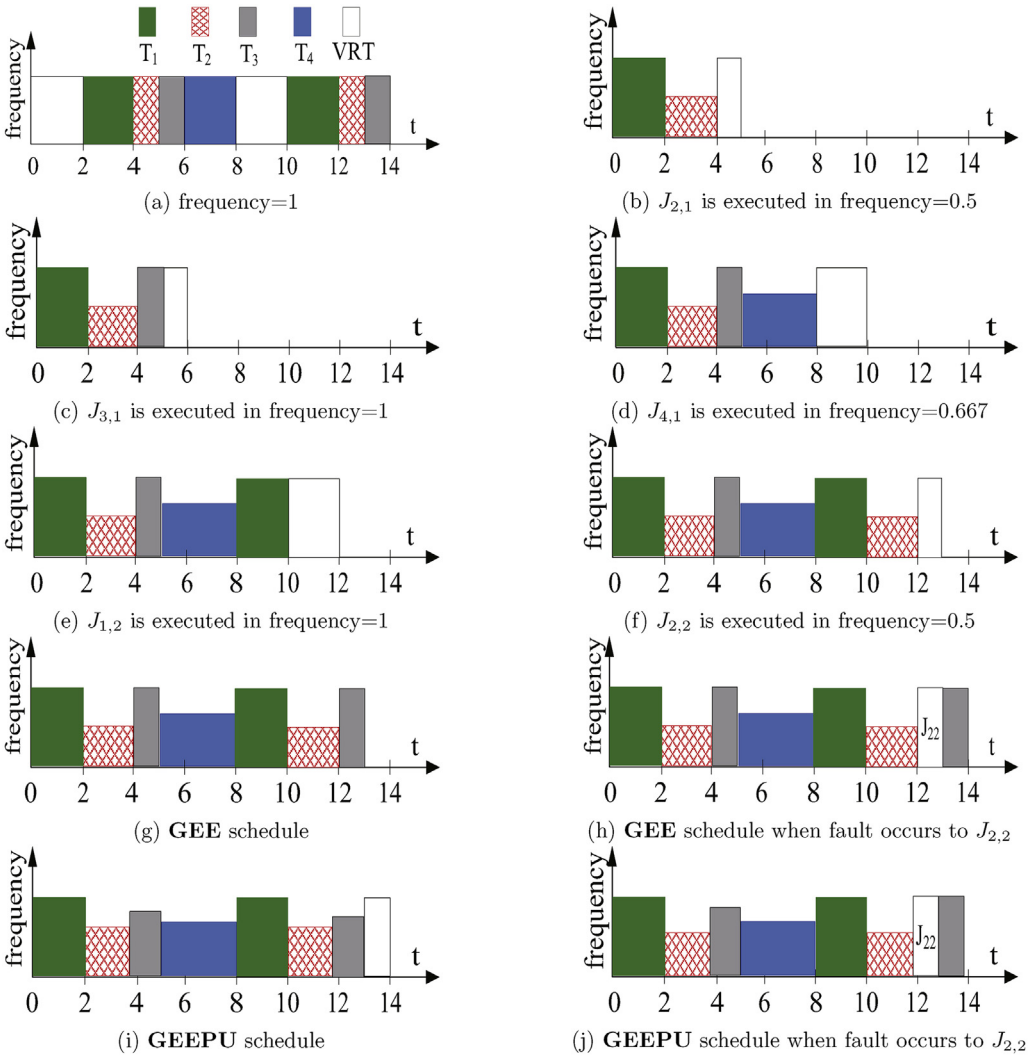


Fig. 2. Scheduling for 4 tasks.

schedule result is shown in Fig. 2(a) by using EDF. Because *VRT* does not need to be actually executed, *VRT* and T_1 can switch for each other. When GEE is used for scheduling the tasks, the schedule has a slack time of 2 at the start. Obviously, the slack time is not long enough to allow $J_{1,1}$ turning down the frequency, but is enough for $J_{2,1}$ to turn the frequency to 0.5 and keep a slack time of 1 as the recovery time, as shown in Fig. 2(b). $J_{3,1}$'s frequency cannot be turned down as shown in Fig. 2(c). $J_{4,1}$ get a slack time of 3 because of the arriving of the next *VRT* and can be executed in the frequency 0.667, as shown in Fig. 2(d). At the instant, $J_{1,2}$, $J_{2,2}$ and $J_{3,2}$ is ready, but $J_{4,1}$ has the same deadline, so no preemption happens. $J_{1,2}$ has a slack time of 2 and cannot turn down the frequency, as shown in Fig. 2(e). $J_{2,2}$ can turn the frequency to 0.5 as shown in Fig. 2(f). The slack time remains 1 at last so $J_{3,2}$ has to be executed in the maximum frequency. The whole schedule is shown in Fig. 2(g). If some transient faults occur to $J_{2,2}$, $J_{2,2}$ would recover execution, as shown in Fig. 2(h).

Fig. 2(i) shows the results of scheduling using the GEEPU algorithm. According to Section 4.2, $f_{low} = 0.6$ and $J_{1,1}$ is executed in frequency 1, $J_{2,1}$ in frequency 0.55, which can be finished at instant 3.82 with a slack time of 1.18. $J_{3,1}$ can be executed in frequency 0.85 and would finish at instant 5 with a slack time of 1. $J_{4,1}$ has a slack time of 3 in runtime so it can turn the frequency to 0.67 and is supposed to be finished at instant 8 with a slack time of 2. $J_{1,2}$ cannot adjust the frequency and would be finished at instant 10 with the slack time of 2. $J_{2,2}$ can be executed in frequency 0.55 and finished at instant 11.82. $J_{3,2}$ can be executed in frequency 0.85 and then finishes at instant 13. If some transient faults occur to $J_{2,2}$ and $J_{2,2}$ has to recover the execution, $J_{3,2}$ will be finished at instant 13.82 as shown in Fig. 2(j).

According to the fault model mentioned above and the experimental parameters in Section 5.1, Without any fault, the energy consumption of GEE is 8.69 and GEEPU is 8.23. Furthermore, even with the fault occurrence to $J_{2,2}$, the energy consumption of GEE is 9.79 and GEEPU is 9.59. Therefore, GEEPU is more energy efficient with a guaranteed reliability.

4.4. Global energy-efficient scheduling based on processor utilization (GLEEPU)

As mentioned in Section 4.2, Consider a task set $T = \{T_1(P_1 = 4, C_1 = 1), T_2(P_2 = 5, C_2 = 1), T_3(P_3 = 5, C_3 = 1), T_4(P_4 = 10, C_4 = 2)\}$, the processor utilization is in the maximum frequency $U_{sys} = 85\%$. All the tasks do not meet the situation that $U_i < 1 - U_{sys}$. GEEPU is similar to GEE, and tasks cannot be executed in a fairly balanced frequency. To deal with this kind of situation, a global energy efficient scheduling algorithm based on processor utilization (GLEEPU) is designed, which do not consider the situation of $U_i < 1 - U_{sys}$, but with the consideration of the average execution frequency $f_{avg} = U_{sys} \times f_{max}$. When the frequency can be adjusted, let the exe-

cution frequency of the task be as close as possible to f_{avg}, and *slackTime* can be fully utilized. Therefore, when *curJob*'s frequency f_{curJob} is less than f_{avg}, the frequency f_{curJob} is adjusted to $\frac{f_{avg}+f_{curJob}}{2}$.

4.5. Dynamic global energy-efficient scheduling based on the actual execution time (DGAET)

In real situation, many jobs finish earlier than their WCET, the actual execution time (AET) is often much less than the WCET. This situation will produce some extra slack time for the system which can be used to optimize the energy consumption. Besides, how to use the slack time should be considered, it can affect the next job only, or affect several latter jobs globally. Obviously, the energy consumption would be much more balanced if the slack time can be used globally. So, a global energy efficiency algorithm based on actual execution time (DGAET) is designed which set the basic frequency $f_{base} = f_{avg}$. If there exist some slack time because of that the actual execution time is less than WCET, f_{base} is turned down to get lower frequency, and if the actual execution time is equal to WCET, f_{base} is recovered. So the function of task execution frequency can be modified as Algorithm 4 (SetFrequency3):

Algorithm 4. SetFrequency3

Input: *curJob* and *slackTime*
Output: set the execution frequency of *currJob*

1: $recovertime \longleftarrow \frac{C_{curJob}}{f_{\max}}$
2: $s \longleftarrow D_{curJob} - recovertime - currentTime$
3: $b \longleftarrow$ Judge whether the next *VRT* arrives before $currentTime + \frac{C_{curJob}}{f_{\max}}$
4: $s \longleftarrow \min(s, slackTime + b \times C_{VRT})$
5: **if** $s < recovertime$ **then**
6: $f_{curJob} \longleftarrow f_{\max}$
7: $f_{base} \longleftarrow f_{avg}$
8: **else**
9: $f_{curJob} \longleftarrow \frac{C_{curJob}}{s}$
10: **if** $f_{curJob} < f_{base}$ **then**
11: $f_{curJob} \longleftarrow \frac{f_{curJob}+f_{base}}{2}$
12: $f_{base} \longleftarrow f_{curJob}$
13: **end if**
14: **end if**

In Algorithm DGAET, to gather the slack time for a task which is completed ahead, the expected completion time of *curJob* is calculate by $\frac{C_{curJob}}{f_{\max}}$. When the *curJob* is completed, the AET_{curJob} can be obtained, therefore, the slack time and AET_{curJob} in Rule 4 can be calculated.

Algorithm 5. DGAET

Input: *TaskSet*
Output: Set the execution frequency of the task and execute

Construct *VRT*
$f \longleftarrow f_{base}$
at each time tick
if *VRT* arrives **then**
$slackTime \longleftarrow slackTime + C_{VRT}$
end if
if a frequency turned down task fault **then**
$slackTime \longleftarrow slackTime - \frac{C_{curJob}}{f_{curJob}}$
turn the frequency to maximum and re-execute *curJob*
end if
if *curJob* is completed correctly **then**
$slackTime \longleftarrow slackTime + (\frac{C_{curJob}}{f_{\max}} - (currentTime - st_{curJob}))$
if *ReadyQ* is not empty **then**
$currentJob \longleftarrow$ get task from *ReadyQ*
SetFrequency3(*curJob*, *slackTime*) and execute *curJob*
$st_{curJob} \longleftarrow currentTime$
else
processor sleep
decrease $slackTime \longleftarrow slackTime - 1$
end if
end if
if *newJob* arrives **then**
if processor idle or ($D_{newJob} < D_{curJob}$ and *newJob* do not need to delay) **then**
if $curJob \neq NULL$ **then**
insert Rem_{curJob} into *ReadyQ*
$slackTime \longleftarrow slackTime - (C_{curJob} - Rem_{curJob}) \times \frac{f_{\max} - f_{curJob}}{f_{curJob}}$
end if
$curJob \longleftarrow newJob$
SetFrequency3(*curJob*, *slackTime*) and execute *curJob*
$st_{curJob} \longleftarrow currentTime$
else
insert *newJob* into *ReadyQ*
end if
end if

Similar to GEE, algorithm DGAET includes 4 parts. Lines 4–6 show the situation when *VRT* arrives. Lines 7–10 show the situation when some transient faults occurs to the task whose frequency is turned down. Lines 11–21 show the situation when the current job is finished, and the next job is scheduled. Lines 22–34 show the situation of preemption when a new job arrives. In Algorithm DGAET, if the *curJob* is finished in *currentTime* with an actual execution time less than the WCET, some slack time is produced which is equal to $\frac{C_{curJob}}{f_{\max}} - (currentTime - st_{curJob})$, while st_{curJob} is the start time of *curJob*.

4.6. Deadline

According to the construction of *VRT*, it is inserted to the task set *T* to make the processor utilization 100%. If *VRT*'s deadline is equal to the summary of the arriving time and period, the new task set is still schedulable. In this paper, the slack time is collected by *VRT* to adjust the frequency, no matter there is fault occur or not, the next job would be executed in some instant to meet the deadline.

4.7. Time complexity analysis

The worst case where n jobs arrive at the same time and are sorted by EDF in the ready queue has the time complexity of $O(n\log n)$. The time complexity of recovering a faulty task with a reduced frequency is $O(1)$. When the *curjob* is finished correctly, the next job in the ready queue will be executed, before which the execution frequency should be configured with a time complexity of $O(1)$. When the new task arrives, to determine whether or not to preempt has the time complexity of $O(n)$. So the time complexity is at most $O(n\log n)$ at each scheduling point.

5. Evaluation

5.1. Experimental parameters

In this paper, a C++ program is used to simulate the execution of task sets. Based on [19], the parameter settings in the energy consumption model are $\beta = 0.1$, $C_{ef} = 1$ and $m = 3$. Based on what is mentioned in Section 2.2, the energy-efficient frequency is obtained as $f_{\text{critical}} = 0.37$. The energy consumption is calculated by Eq. (3), and time is expressed by time units. The periods of tasks are uniformly distributed within the range of [10,100]. The tasks are divided into short period tasks with $p \in [10, 20]$, middle period tasks with $p \in (20, 80]$ and long period tasks $p \in (80, 100]$. The numbers of the 3 kinds of tasks are approximately equal. 100 task sets are constructed for every simulation, execute for 10^5 time units and take the average of the results. It is assumed that the occurrence of transient faults follows the Poisson distribution and the probability of faults in the maximum frequency f_{max} is $\lambda_0 = 10^{-10}$. When the

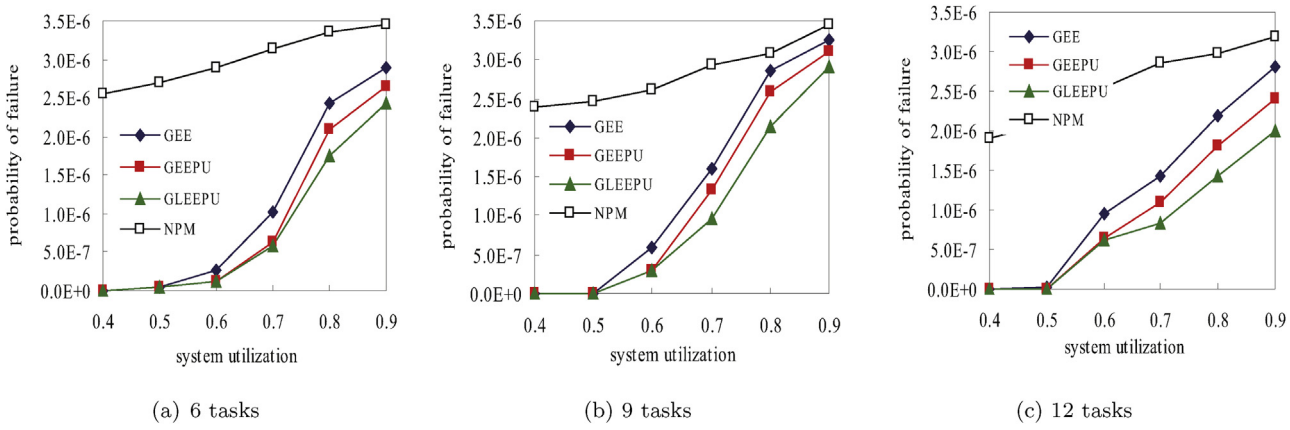


Fig. 3. The probability of failure.

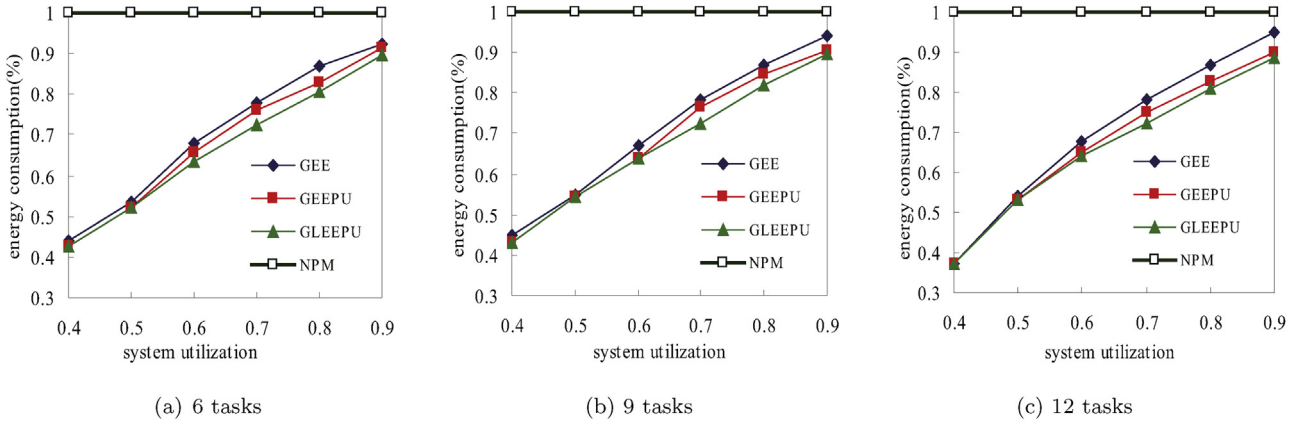


Fig. 4. The energy consumption.

frequency is turned down, the fault probability is $\lambda(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{\min}}}$, here $d = 2$ [19,23]. The probability of failure in runtime is $1 - \prod R_i$.

5.2. Situation when tasks run for WCET

In this section, all tasks are executed with WCET. The performance of the GEE, GEEPU and GLEEPU is evaluated using the NPM (no power management) [19] algorithm as a benchmark. The system overhead caused by preemption has not been considered. NPM do not adjust the frequency and let all the tasks run in the maximum frequency, when no task is executed, the processor sleeps.

Experiment 1: Fixed task number, varying processor utilization. In this experiment, the task's WCET is constructed as a random value, and processor utilization is set from 0.4 to 0.9. The 6 tasks, 9 tasks and 12 tasks are simulated separately, normalization the energy consumption refer to NPM. The probability of failure, energy consumption and preemption are analyzed as shown in Figs. 3–5.

As mentioned in Fig. 3, if the task number is fixed, the failure probability of the proposed four algorithms will be increased with an increasing system utilization. When the system utilization drops below 0.6, the failure probability of GEE, GEEPU, and GLEEPU are significantly smaller than NPM. Noticed that GLEEPU always has the lowest failure probability.

As shown in Fig. 4, GEE, GEEPU and GLEEPU are much better in energy efficient than NPM. When the utilization is 0.4, the energy consumption of the proposed 3 algorithms is about 44% of NPM. With the increase of processor utilization, the energy consumption increases because less tasks are turned to execute in lower frequency. GEE, GEEPU and GLEEPU have similar performance of energy consumption while GLEEPU is a little better.

As shown in Fig. 5, the number of preemption of NPM increased with an increasing system utilization. When the system utilization is below 0.8, NPM has the smallest number of preemption compared with other techniques. This is because, NPM is with the highest execution frequency, when the system utilization is small, the processor has more time in the idle state resulting in reduced preemption probability. Since GEE, GEEPU, and GLEEPU reduce their execution frequency, tasks have more chances to be preempted. When system utilization increases, the numbers of preemptions for GEE, GEEPU, and GLEEPU have no obvious variance. Noticed that GEE has no control for the preemption, therefore, it has a significant number of extra preemptions compared with GEEPU and GLEEPU. Once the system utilization is beyond 0.8, fewer preemptions happen on GEEPU and GLEEPU than NPM.

Experiment 2: Fixed processor utilization, varying task number. The 3, 6, 9, 12, and 15 tasks are constructed when the utilization is 0.5, 0.7 and 0.9, respectively. The probability of failure, energy consumption and preemption are analyzed as shown in Figs. 6–8.

As shown in Fig. 6, although the failure probabilities of all techniques become larger with the increase of system utilization, the proposed algorithms all have less failure probability than NPM. Overall, GLEEPU has the lowest failure probability.

As shown in Fig. 7, GEE, GEEPU and GLEEPU all have better energy efficiency than NPM. Averagely the energy consumption of the proposed algorithms is about 55% of NPM with the processor utilization of 0.5, 78% with processor utilization of 0.7, and 92% with

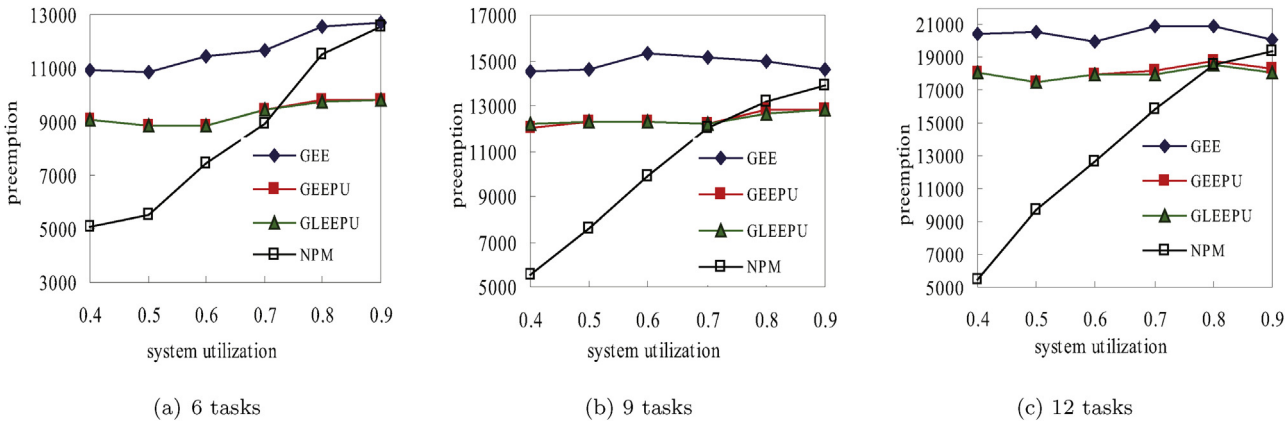


Fig. 5. The preemption.

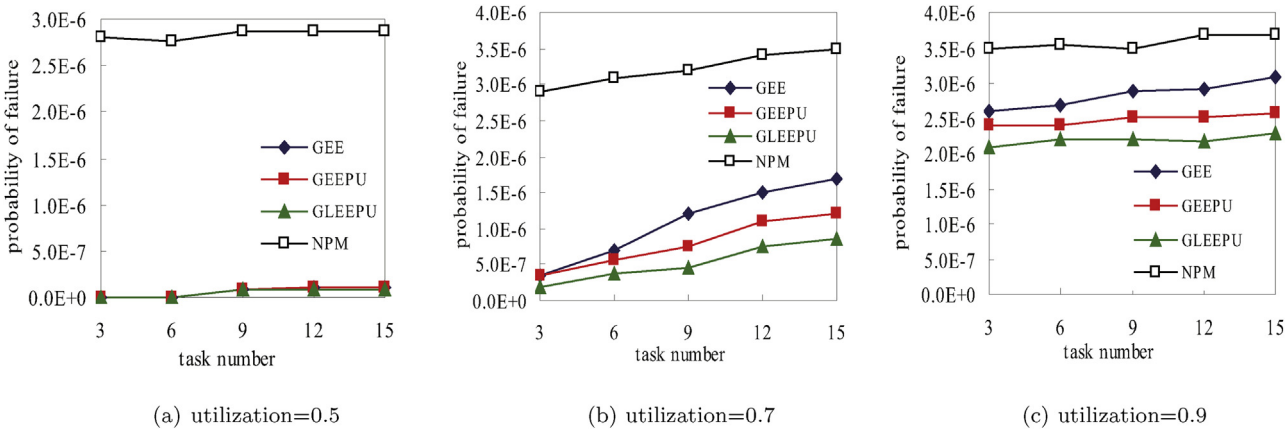


Fig. 6. The reliability with fixed utilization and varying task number.

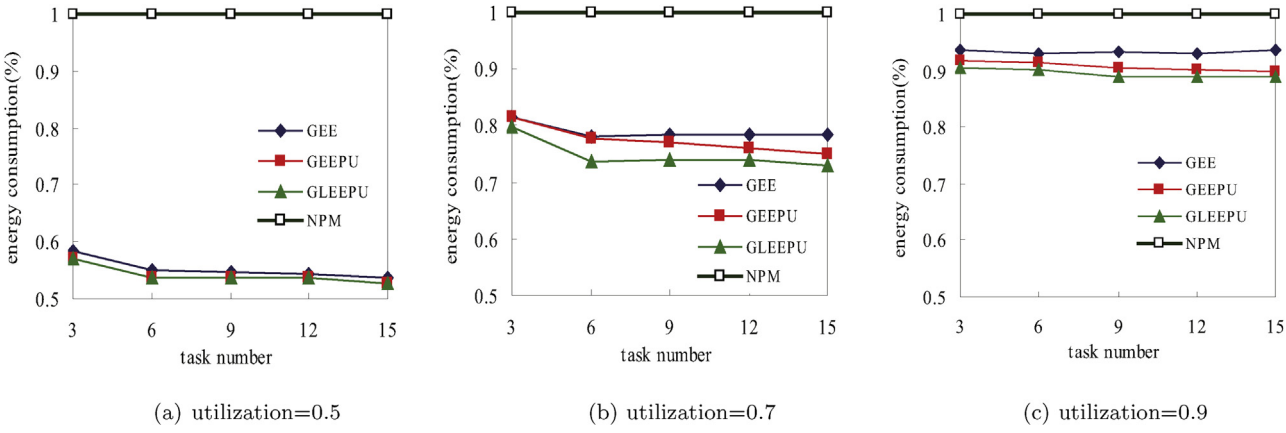


Fig. 7. The energy consumption with fixed utilization and varying task number.

processor utilization of 0.9. Among these three algorithms, GLEEPU is superior both in energy efficiency and reliability.

As shown in Fig. 8, GEE has the largest number of preemptions as it reduces the execution frequency. When the system utilization is 0.5, NPM has the smallest number of preemptions. When the system utilization is 0.7, the number of preemptions of NPM is approaching to that of GEEPU and GLEEPU. When the system utilization is 0.9, NPM has a larger number of preemptions than GEEPU and GLEEPU.

Experiment 1 and Experiment 2 show that despite the changes of system utilization, GEE, GEEPU and GLEEPU always have lower energy consumption and better reliability than NPM. For GEE, the number of preemptions is significantly larger than that of GEEPU and GLEEPU.

5.3. Comparison when tasks run for AET

In a real-life system, most tasks have less execution time than WCET. Therefore, in the following experiments, it is reasonable to assume that many tasks are with the execution time between the best-case execution time (BCET) and WCET. Hence, these tasks are assigned a random execution time with the range of [BCET, WCET].

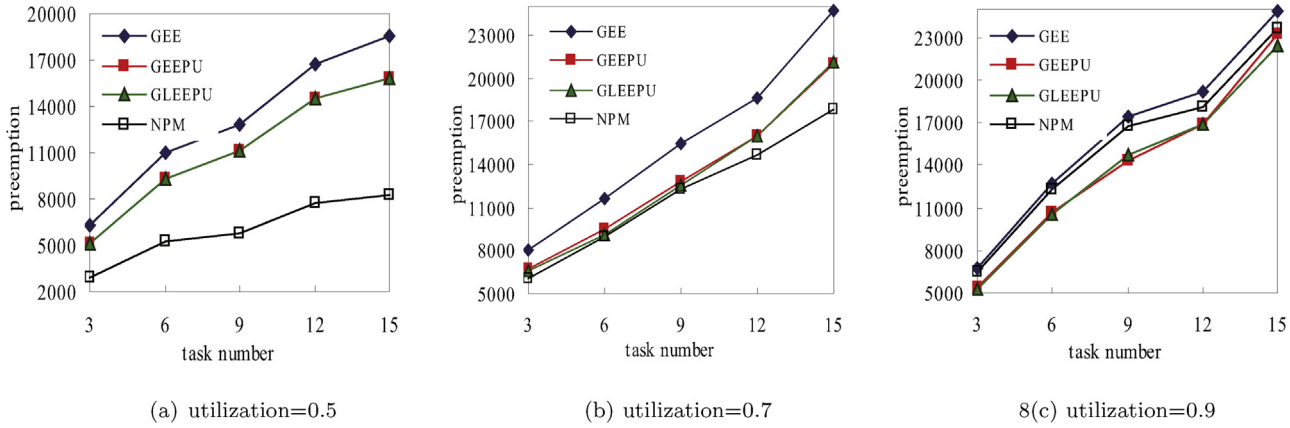


Fig. 8. The preemption with fixed utilization and varying task number.

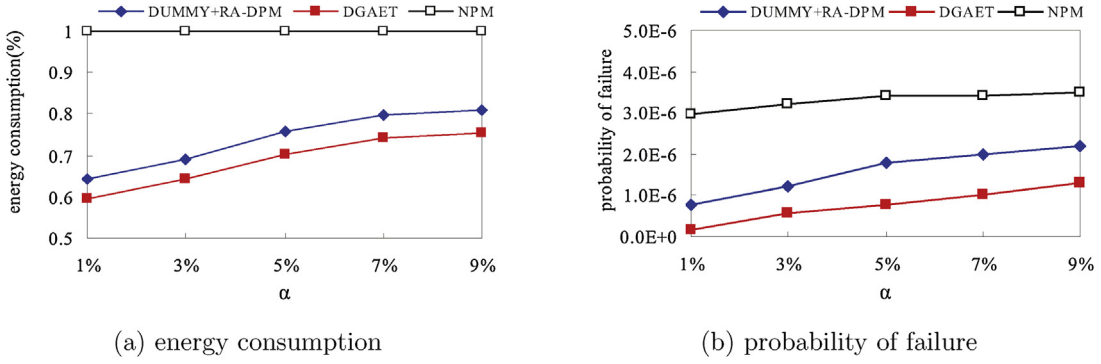


Fig. 9. Processor utilization = 0.8.

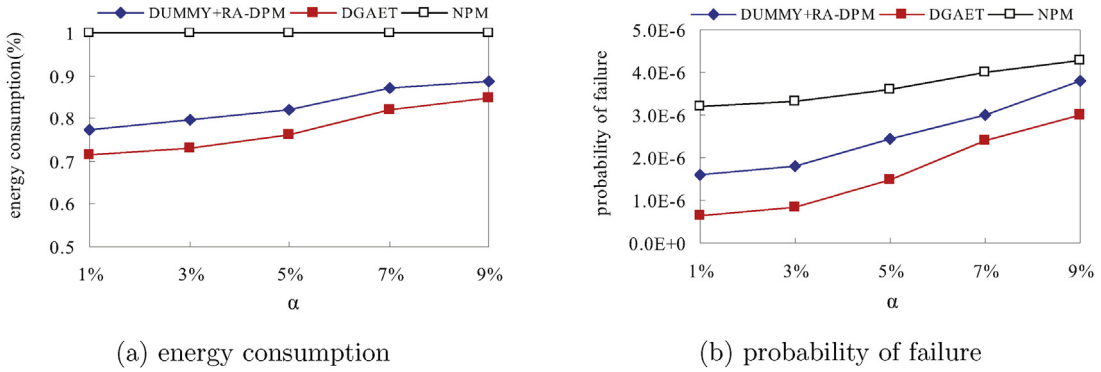


Fig. 10. Processor utilization = 0.9.

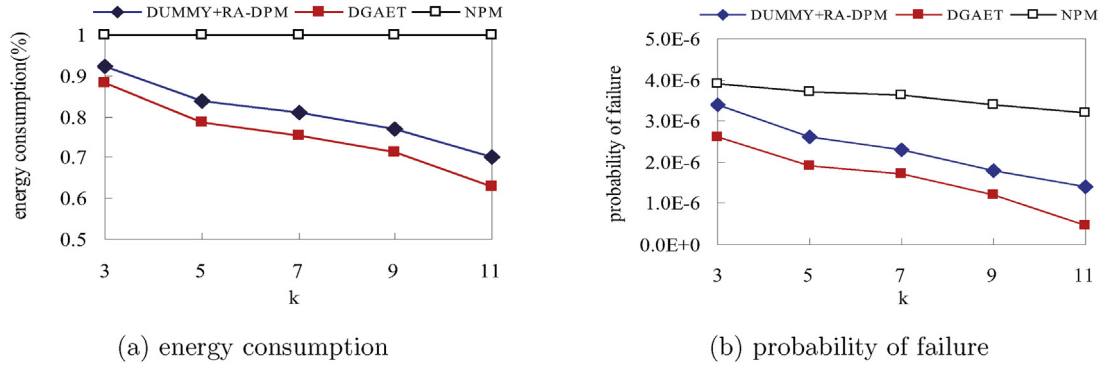
Besides, considering the preemption overhead, the time overhead for a single preemption of *curJob* is defined as $\alpha \times C_{curJob}$ and based on the current execution frequency f_{curJob} the energy overhead can be calculated. The performance of NPM, DUMMY+RA-DPM [19], and DGAET is evaluated, when dummy task in DUMMY+RA-DPM has the minimum period.

Experiment 3: 12 tasks are constructed with BCET/WCET = 0.2 to evaluate both the energy consumption and reliability of the proposed techniques with respect to the parameter α (1%, 3%, 5%, 7% and 9%). Figs. 9 and 10 show the evaluations with different processor utilizations of 0.8 and 0.9, respectively.

As shown in Figs. 9 and 10, DUMMY+RA-DPM and DGAET has less energy consumption and failure probability than NPM. When the preemption overhead increases, the energy consumption of DUMMY+RA-DPM and DGAET is slowly increasing relative to NPM, at the same time the failure probability of three algorithms have the tendency to increase. Among these three algorithms, DGAET is superior both in energy efficiency and reliability.

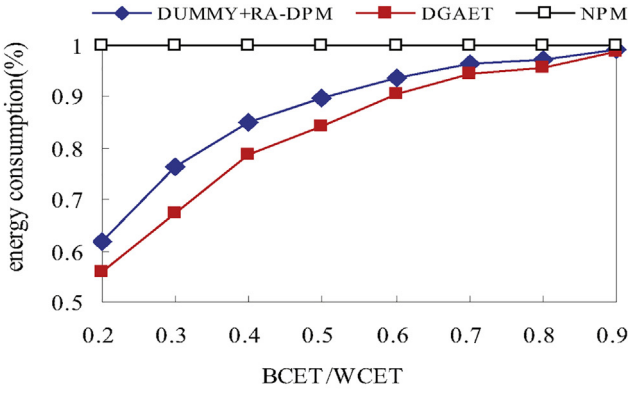
Experiment 4: 12 tasks with processor utilization 0.8, BCET/WCET = 0.2, α = 5%. There are k tasks with their execution time within [BCET, WCET], the rests are with the execution time of WCET. When k is 3, 5, 7, 9, and 11, Fig. 11 shows the results of the reliability and energy consumption.

As shown in Fig. 11, DUMMY+RA-DPM and DGAET has less energy consumption and failure probability than NPM. When the utilization is fixed, k is varied from 3 to 11, the energy consumption and failure probability of DUMMY+RA-DPM and DGAET are becoming smaller and smaller. DGAET has obvious better performance than DUMMY+RA-DPM and NPM.

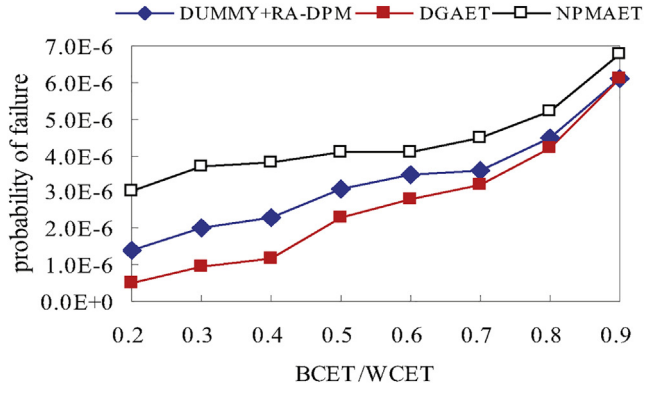


(a) energy consumption (b) probability of failure

Fig. 11. k from 3 to 11 with utilization 0.8.



(a) energy consumption



(b) probability of failure

Fig. 12. BCET/WCET from 0.2 to 0.9 with utilization 0.9.

Experiment 5: 15 tasks with processor utilization of 0.9, $\alpha = 5\%$. The execution time of tasks follows uniform distribution in between BCET and WCET. When BCET/WCET is from 0.2 to 0.9, the reliability and energy consumption are shown in Fig. 12.

As shown in Fig. 12, DGAET has better performance in energy consumption and failure probability than NPM and DUMMY+RA-DPM.

Accord to Experiment 3, Experiment 4, and Experiment 5, DGAET is better than the pre-existing algorithm when the actual execution time is less than WCET and the time and energy consumption of preemption is considered.

6. Conclusion

DVFS is efficient to optimize the system energy consumption but would reduce the system reliability. In this paper, a virtual is constructed to collect slack time, and based on slack time, the task's execution frequency is adjusted. In order to guarantee the reliability, if a task's execution frequency is reduced and some transient fault occurs, the task will be re-executed in maximum frequency. The GEE algorithm is first proposed in this paper, then, a simple analysis of preemption is given which reduces preemption. The two algorithms based on the processor utilization are designed which make the execution frequency more balanced. Considering the fact that the AET of many tasks may be less than WCET, an algorithm based on the AET is also designed. Simulation results show that the proposed algorithms have significant improvements in terms of both energy efficiency and reliability over other related works.

Acknowledgements

The research was partially funded by the National Natural Science Foundation of China (Grant Nos. 61173036).

References

[1] N. Zheng, Z. Wu, M. Lin, L.T. Yang, Enhancing battery efficiency for pervasive health-monitoring systems based on electronic textiles Information, IEEE Trans. Technol. Biomed. 14 (2) (2010) 350–359.
[2] W. Jiang, G. Xiong, X. Ding, Energy-saving service scheduling for low-end cyber-physical systems, in: The 9th International Conference for Young Computer Scientists, 2008, ICYCS 2008, IEEE, 2008, pp. 1064–1069.
[3] Y. Ge, Y. Dong, H. Zhao, An energy management strategy for energy-sustainable cyber-physical system, in: 2014 9th International Conference on Computer Science & Education (ICCSE), IEEE, 2014, pp. 413–418.
[4] E. Lee, et al., Cyber physical systems: design challenges, in: 2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC), IEEE, 2008, pp. 363–369.
[5] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. King, M. Mullen-Fortino, S. Park, A. Roederer, et al., Challenges and research directions in medical cyber-physical systems, Proc. IEEE 100 (1) (2012) 75–90.
[6] S. Liu, J. Lu, Q. Wu, Q. Qiu, Harvesting-aware power management for real-time systems with renewable energy, IEEE Trans. Very Large Scale Integr. Syst. 20 (8) (2012) 1473–1486.
[7] P. Derler, E. Lee, A.S. Vincentelli, et al., Modeling cyber-physical systems, Proc. IEEE 100 (1) (2012) 13–28.
[8] R. Jejurikar, C. Pereira, R. Gupta, Leakage aware dynamic voltage scaling for real-time embedded systems, in: Proceedings of the 41st Annual Design Automation Conference, ACM, 2004, pp. 275–280.
[9] X. Kavousianos, K. Chakrabarty, A. Jain, R. Parekhji, Test scheduling for multicore socs with dynamic voltage scaling and multiple voltage islands, in: 2011 20th Asian Test Symposium (ATS), IEEE, 2011, pp. 33–39.
[10] A.M. Tokarnia, P.C. Pepe, L.D. Pagotto, Path-based dynamic voltage and frequency scaling algorithms for multiprocessor embedded applications with soft delay deadlines, in: 2011 14th Euromicro Conference on Digital System Design (DSD), IEEE, 2011, pp. 109–116.

[11] J.-K. Kim, H.J. Siegel, A. Maciejewski, R. Eigenmann, et al., Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling, IEEE Trans. Parallel Distrib. Syst. 19 (11) (2008) 1445–1457.
[12] J. Li, L. Shu, J.-J. Chen, G. Li, Energy-efficient scheduling in nonpreemptive systems with real-time constraints, IEEE Trans. Syst. Man Cybern. Syst. 43 (2) (2013) 332–344.
[13] A. Mishra, A.K. Tripathi, Energy efficient voltage scheduling for multi-core processors with software controlled dynamic voltage scaling, Appl. Math. Model. 38 (14) (2014) 3456–3466.
[14] W.Y. Lee, Energy-efficient scheduling of periodic real-time tasks on lightly loaded multicore processors, IEEE Trans. Parallel Distrib. Syst. 23 (3) (2012) 530–537.
[15] K. Li, X. Tang, K. Li, Energy-efficient stochastic task scheduling on heterogeneous computing systems, IEEE Trans. Parallel Distrib. Syst. 25 (11) (2014) 2867–2876.
[16] Y. Ding, X. Qin, L. Liu, T. Wang, Energy efficient scheduling of virtual machines in cloud with deadline constraint, Future Gener. Comput. Syst. 50 (2015) 62–74.
[17] E. Aldahari, Dynamic voltage and frequency scaling enhanced task scheduling technologies toward green cloud computing, in: 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science & Engineering (ACIT-CSII-BCD), IEEE, 2016, pp. 20–25.
[18] D. Zhu, R. Melhem, D. Moss, The effects of energy management on reliability in real-time embedded systems, in: IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004, IEEE, 2004, pp. 35–40.
[19] D. Zhu, H. Aydin, Reliability-aware energy management for periodic real-time tasks, IEEE Trans. Comput. 58 (10) (2009) 1382–1397.
[20] D. Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, in: Real-Time and Embedded Technology and Applications Symposium, 2006, Proceedings of the 12th IEEE, IEEE, 2006, pp. 397–407.
[21] D. Zhu, Reliability-aware dynamic energy management in dependable embedded real-time systems, ACM Trans. Embed. Comput. Syst. (TECS) 10 (2) (2010) 26.
[22] T. Wei, X. Chen, S. Hu, Reliability-driven energy-efficient task scheduling for multiprocessor real-time systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 30 (10) (2011) 1569–1573.
[23] M. Lin, Y. Pan, L.T. Yang, M. Guo, N. Zheng, Scheduling co-design for reliability and energy in cyber-physical systems, IEEE Trans. Emerg. Top. Comput. 1 (2) (2013) 353–365.
[24] Z. Li, L. Wang, S. Li, S. Ren, G. Quan, Reliability guaranteed energy-aware frame-based task set execution strategy for hard real-time systems, J. Syst. Softw. 86 (12) (2013) 3060–3070.
[25] B. Zhao, H. Aydin, D. Zhu, Shared recovery for energy efficiency and reliability enhancements in real-time applications with precedence constraints, ACM Trans. Des. Autom. Electron. Syst. (TODAES) 18 (2) (2013) 23.
[26] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, K. Li, Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster, Inf. Sci. 319 (2015) 113–131.
[27] Y. Xiang, S. Pasricha, Soft and hard reliability-aware scheduling for multicore embedded systems with energy harvesting, IEEE Trans. Multi-Scale Comput. Syst. 1 (4) (2015) 220–235.
[28] K.G.S. Jinkyu Lee, Preempt a job or not in EDF scheduling of uniprocessor systems, IEEE Trans. Comput. 63 (5) (2014) 1197–1206.
[29] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, G. Buttazzo, Optimal selection of preemption points to minimize preemption overhead, in: 2011 23rd Euromicro Conference on Real-Time Systems (ECRTS), IEEE, 2011, pp. 217–227.
[30] M. Bertogna, S. Baruah, Limited preemption EDF scheduling of sporadic task systems, IEEE Trans. Ind. Inform. 6 (4) (2010) 579–591.
[31] Y. Xiang, S. Pasricha, Run-time management for multicore embedded systems with energy harvesting, IEEE Trans. Very Large Scale Integr. Syst. 23 (12) (2015) 2876–2889.
[32] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM (JACM) 20 (1) (1973) 46–61.
[33] X. Castillo, S.R. McConnel, D.P. Siewiorek, Derivation and calibration of a transient error reliability model, IEEE Trans. Comput. 100 (7) (1982) 658–671.
[34] C. Paleologu, J. Benesty, T. Gaensler, S. Ciochină, Class of double-talk detectors based on the holder inequality, in: 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2011, pp. 425–428.