DOI 10.12694/scpe.v19i4.1438
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 19, Number 4, pp. 387–400. http://www.scpe.org © 2018 SCPE

ENERGY-EFFICIENT REAL-TIME SCHEDULING ALGORITHM FOR FAULT-TOLERANT AUTONOMOUS SYSTEMS*

HUSSEIN EL GHOR† JULIA HAGE‡ NIZAR HAMADEH§ AND RAFIC HAGE CHEHADE¶

Abstract. For the past decades, we have experienced an aggressive technology scaling due to the tremendous advancements of Integrated Circuit technology. As massive integration continues, the power consumption of the IC chips exponentially increases which further degraded the system reliability. This in turn poses significant challenges to the design of real-time autonomous systems. In this paper, we target the problem of designing advanced real-time scheduling algorithms that are subject to timing, energy consumption and fault-tolerant design constraints. To this end, we first investigated the problem of developing scheduling techniques for uniprocessor real-time systems that minimizes energy consumption while still tolerating up to k transient faults to preserve the system's reliability. Two scheduling algorithms are proposed: the first scheduler is an extension of an optimal fault-free energy-efficient scheduling algorithm, named ES-DVFS. The second algorithm aims to enhance the energy saving by reserving adequate slack time for recovery when faults strike. We derive a necessary and sufficient condition that must be efficiently checked for the time and energy feasibility of aperiodic jobs in the presence of failures. Later, we formally prove that the proposed algorithm is optimal for a k-fault-tolerant model. Our simulation results demonstrate that the proposed schedulers can efficiently improve energy savings when compared with previous works.

Key words: Real-time systems, real-time scheduling, fault tolerance, energy consumption, processor demand, ES-DVFS scheduler

AMS subject classifications. 68M15, 94C12

1. Introduction.

Autonomous systems are becoming increasingly important in our lives. In these autonomous devices, the management of energy is a crucial issue. They are more and more varied and appear in extremely diverse sectors such as transport (avionics, cars, buses, ..), multimedia, mobile phones, game consoles, etc. A large part of autonomous systems have needs for autonomy and limitations of space (small size) and energy (limited consumption). As a result, the major technological and scientific challenge is to build systems of trust from the point of view of the functionalities provided and the rendered quality of service. Its more about designing these systems at an acceptable cost.

For the past several decades, we have experienced tremendous growth of real-time systems and applications largely due to the remarkable advancements of IC technology. However, as transistor scaling and massive integration continue, the dramatically increased power/energy consumption and degraded reliability of IC chips have posed signficant challenges to the design of real-time embedded systems [1]. Hence, it is imperative to propose efficient and effective power/energy management techniques for real-time systems while still guaranteeing the timing constraints. For the past years, extensive real-time energy-efficient scheduling algorithms have been proposed to minimize the processor energy consumption for embedded systems [2], [3].

Such a problem is usually treated by Dynamic Voltage and Frequency Scaling (DVFS) methods that affect the processor speed, which directly affects the energy consumption of the system. The energy-efficient scheduling of real-time jobs on a DVFS processor has been extensively studied in the previous decade [4], [5], [6].

At the same time, it is observed that as autonomous real-time systems become more and more complex, the required level of reliability for such systems appears to be another open problem. Such comples systems are usually situated at harsh, remote or inaccessible locations. Consequently, it is often difficult and sometimes even impossible to repair and to perform maintenance. This necessitates the use of fault-tolerant techniques. Fault-tolerant computing stands for the reliable (correct) execution of system software and user programs in the presence of failures [7]. Nowadays, the impacts of system failures become more and more substantial, ranging from personal inconvenience, disruption of our daily lives, to some catastrophic consequences such as huge financial loss. Conceivably, guaranteeing the reliability of computing systems has also been raised

*This work was supported by the Laboratory of Embedded and Networked Systems at the Lebanese University.
†LENS Laboratory, Faculty of Technology, Lebanese University, B.P. 813 Saida, LEBANON. (husseinelghor@ul.edu.lb).
‡Faculty of Technology, Lebanese University, B.P. 813 Saida, LEBANON.
§LENS Laboratory, Faculty of Technology, Lebanese University, B.P. 813 Saida, LEBANON (nizar.hemadeh@ul.edu.lb).
¶LENS Laboratory, Faculty of Technology, Lebanese University, B.P. 813 Saida, LEBANON (rhagechehade@ul.edu.lb).

to be a first-class design concern. Recent studies indicate that the emerging energy-efficient design techniques further increase the susceptibility of VLSI circuits to transient faults [4]. Left unchecked, the high power/energy consumption and deteriorating reliability of IC chips will handicap the availability of future generations of real-time computing systems. Hence, faults have to be detected and convenient recovery methods must be performed within the timing constraints.

Processor faults can be mainly seperated into two categories: transient and permanent faults [8]. Transient faults are temporary malfunctioning of the computing unit or any other associated components caused by factors such as electromagnetic interference and cosmic ray radiations, which causes incorrect results to be computed. On the contrary, a permanent or hard fault in hardware is an erroneous state that is continuous and stable. Permanent faults in hardware are caused by the failure of the computing unit. We focus in this paper on the transient fault since, in most computing systems, the majority of errors are due to transient faults [9]. In the case of an energy-efficient system, reliability also means ensuring that the system will never be short of energy to ensure its treatment. Anticipation of possible cases of energy can, again, be implemented on the basis of the flexibility offered by the system at the level of the execution of the tasks.

In this work, we target the problem of real-time scheduling under reliability and energy constraints. Its about considering real-time tasks that have needs which are expressed on the one hand in terms of processing time and energy consumed by the processor and on the other hand in terms of the number of tolerated faults. A task configuration is energy overloaded, this means that the amount of energy consumed is greater than the amount of energy available. In addition, the amount of execution time requested is smaller than the available capacity, the system will therefore typically be able to meet all its deadlines or else catastrophic consequences will occur. A major question that needs to be answered is: how to schedule real-time tasks in case of energy where the system keeps reliable and able to tolerate up to k faults.

To answer this question, a uniprocessor Earliest Deadline First (EDF) scheduler is first analyzed to derive an efficient and exact feasibility condition by considering energy management and fault-tolerance. Second, the proposed algorithm is designed to achieve energy autonomous utilization of the processor while meeting the task deadlines.

The rest of the paper is organized as follows. In the next section, we summarize the related work. In section 3, we introduce the model and terminology. The fault tolerant speed schedule was then presented in section 4. Section 5 presents the experimental results to demonstrate energy savings and Section 6 concludes the paper.

2. Related Work. Researchers in both academia and industry have resorted to various techniques to minimize energy consumption in computing systems. Among these, DVFS technique has risen as one of the best framework level methods for energy consumption. DVFS scheduling reduces the supply voltage and frequency when conceivable for preserving energy consumption. Subsequently, a great number of procedures considering the issue of limiting the energy consumption without jeopardizing the timing constraints on uniprocessor platforms are widely proposed in literature for different task models. Many of the previous work that studied the problem of energy efficient frameworks for real-time embedded systems employ the DVFS technique [4], [10], [11], [12], [13].

Yao et al. [10] developed a DVFS scheme for a set of aperiodic real-time tasks scheduled under EDF policy with a focus of minimizing dynamic power consumption for real-time systems. In [12], authors considered the temperate and leakage dependencies and proposed an efficient DVFS scheme to minimize the overall energy consumption while guaranteeing the timing constraints of a real-time system. Later in [13], we settle the hypothesis for enhancing energy saving in real-time systems, we proposed an energy-efficient scheduling algorithm for aperiodic tasksin real-time embedded systems. Specifically, we applied the DVFS technique to the concept of real-time process scheduling. Further, we proposed in [14] an energy guarantee scheduling and voltage/frequency selection algorithm targeting at real-time systems with energy harvesting capability. We show that our scheduler achieves capacity savings when compared to other schedulers.

On the other side, fault tolerance, and in general reliability, objectives are of paramount importance for embedded systems [15]: faults and failures can occur in real-time computing systems and can result in deadline violations and/or hardware errors. Since soft errors are more common in computing systems, most researches related to fault tolerance focus on soft errors. Such research efforts was done on scheduling techniques with the joint consideration of energy efficiency and fault tolerance.

Zhu et al. [16] investigated the reliability problem of a real-time system as the probability to execute all tasks, in abscence or presence of faults. Following this, authors proposed a linear and an exponential model that can detect the effects of DVFS technique on the transient fault rate. They demonstrated that minimizing energy consumption through DVFS can reduce the system reliability. For this sake, they presented a recovery scheme to schedule a recovery for each scaled job to compensate the reliability loss caused by DVFS.

Melhem et al. [17] targeted the reliability problem for a set of periodic tasks scheduled under EDF on a monoprocessor with the restriction that there is at most one failure (i.e. $k = 1$). Authors presented a checkpointing scheme that can reduce the fault-recovery overhead significantly at the cost of runtime overhead, this means by inserting checkpoints, which may potentially improve the system schedulability and leave more space for energy management. Zhang et al. [18] investigated the same problem but on fixed-priority real-time tasks. For this sake, authors proposed a DVFS scheme combined with checkpointing that is able to tolerate faults for a set of periodic tasks to minimize energy consumption.

More recently, Zhao et al. [19] proposed the Generalized Shared Recovery (GSHR) technique to reserve computing resources, which can be used by other tasks to enhance the energy efficiency. Later, this work was extended to be applied to a real-time periodic task model [15]. The proposed algorithms aim to determine the processor scaling factor and the reserved resources for every task to enhance the minimization of energy while still guaranteeing the reliability requirement at the task-level. The advantage of the GSHR scheduler comes from the fact that the reliability of the system can be increased when aplying the DVFS technique.

Recently, Han et al. developed effective scheduling algorithms that can save energy when considering that the proposed real-time system can tolerate up to k failures when scheduling a set of aperiodic tasks on a single processor under the EDF policy [20]. For this sake, authors proposed three algorithms: The first two algorithms are based on the previous work performed in [10]. The third algorithm extends the first two by considering that the computing resources are no longer reserved and hence better energy saving performance can be achieved. The main drawback of this work is that the problem of improving the system reliability in presence of failures cannot be solved by a simple modification to the work done in [10].

3. Model and Terminology. We consider the system model and their corresponding notations. Then, we present the problem formulation.

3.1. Task Model. We consider a set of n independent aperiodic real-time jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$, where J_i denotes the i^{th} job in $\mathcal{J}$ and is characterized by a three tuple (a_i, c_i, d_i). The definition of these parameters are as follows:

- a_i is referred to the arrival time, this means that the time when job J_i is ready for execution.
- c_i stands for the worst case execution time (WCET) under the maximum available speed S_{max} of the processor.
- d_i is considered as the absolute deadline of job J_i.

We denote the laxity of the job J_i by $d_i - (a_i - c_i)$. We consider that the job set $\mathcal{J}$ is said to be feasible in the real-time manner and under fault-free scenario. In other words, there exists a feasible schedule for $\mathcal{J}$ in abscence of energy considerations, where all deadlines in are respected.

3.2. Power and Energy Model. We assume the speed / frequency of the processor is equipped with a DVFS-enabled with N discrete frequencies f ranging from $f_{min} = f_1 \leq f_2 \leq \cdots \leq f_N = f_{max}$. We consider the notation processor speed S_N, or slowdown factor, as the ratio of the computed speed to the maximum processor speed, this means that $S_N = f_N / f_{max}$ [13]. The CPU speed can be changed continuously in $[S_{min}, S_{max}]$. Consequently, when a job J_i is executed under speed S_i, the worst case execution time of J_i becomes equal to c_i/S_i.

For embedded systems, the processor and off-chip devices such as memory, I/O interfaces and underlying circuits mainly consume the major part of the energy [21]. In this paper, we distinguish between frequency-dependent and frequency-independent power components. Specifically, we adopt the overall power consumption (P) at a slowdown factor S as follows:

$$P = P_{ind} + P_{dep} = P_{ind} + C_{ef} S^{\alpha} \tag{3.1}$$

Where P_{ind} stands for the frequency-independent power that includes the constant leakage power and the power

consumed by off-chip devices [20], which is independent of the system frequency and supply voltage. C_{ef} is denoted as the effective switching capacitance. α is the dynamic power exponent, which is a constant usually larger than or equal to 2.
P_{dep} is considered to be the frequency-dependent active power, which includes not only the processor power, but also any power that depends on the processing speed S. Consequently, the energy consumption of a job J_i that runs at the speed S_i, denoted as $E_i(S_i)$, can be expressed as:

$$E_i(S_i) = (P_{ind} + C_{ef} S_i^{\alpha}).\frac{c_i}{S_i} \tag{3.2}$$

3.3. Energy Storage Model. Our system relies on an energy storage unit (battery or supercapacitor) with a nominal capacity, namely C, that corresponds to a maximum stored energy. The energy level of the battery must remain between two predefined boundaries, namely C_{min} and C_{max}, where $C = C_{max} - C_{min}$. We consider that $C(t)$ stands for the energy level in the energy storage unit at time t. We state that the energy stored in the battery at any time is less than the storage capacity, that is

$$C(t) \leq C \quad \forall\, t \tag{3.3}$$

3.4. Fault Model. During the execution of an operation computing system, both permanent and transient faults may occur due to various reasons, like hardware defects or system errors. In this paper, we focus on transient faults since it has been shown to be dominant over permanent faults especially with scaled technology sizes [23].

We consider that the proposed system can afford a maximum of k transient faults. The used system is usually able to detect faults when a job ends its execution. We assume that the energy and time overhead caused by fault detection, denoted as EO_i and TO_i respectively, are not negligible and are independent of the variations in the processor frequency.

Generally, there is not restriction on the occurrence of faults during the execution of jobs and multiple faults may occur when executing a single job [20]. The fault recovery scheme in this paper is based on re-executing the affected job. Consequently, R_i stands for the maximum recovery overhead for executing a job J_i under the maximum speed S_{max}, which is equal to c_i, or $R_i = c_i$. When a fault occurs during any job execution, say J_i , a recovery job is released having the same deadline d_i, which is subject to preemption as well.

3.5. Terminology. We now give some definitions we will be useful throughout the rest of this paper.

Definition 3.1. *A schedule Γ for a job set $\mathcal{J}$ is said to be valid if the deadlines of all jobs of $\mathcal{J}$ are met in Γ, starting with a storage fully charged [13].*

Definition 3.2. *A system is said to be feasible if there exists at least one valid schedule Γ for $\mathcal{J}$ with a given energy source. Otherwise, it is infeasible [13].*

In this paper, we consider that the limiting factors are not only time but are either, both time and energy, only energy or only time. We focus here on feasible systems only.
Formally, we introduce a novel terminology that is peculiar to energy constrained real-time computing systems.

Definition 3.3. *A schedule Γ for a job set $\mathcal{J}$ is said to be time-valid if the deadlines of all jobs of $\mathcal{J}$ are met in Γ, considering that $\forall\ 1 \leq i \leq n$, $E_i(S_i) = 0$ [13].*

Definition 3.4. *A system is said to be time-feasible if there exists at least one time-valid schedule Γ for $\mathcal{J}$. Otherwise, it is infeasible [13].*

Definition 3.5. *A schedule Γ for a job set $\mathcal{J}$ is said to be energy-valid if the deadlines of all jobs of $\mathcal{J}$ are met in Γ, considering that $\forall\ 1 \leq i \leq n$, $c_i = 0$ [13].*

Definition 3.6. *A system is said to be energy-feasible if there exists at least one time-valid schedule Γ for $\mathcal{J}$. Otherwise, it is infeasible [13].*

3.6. Problem Formulation. We formulate the problem in this paper as follows:
Given a set of real-time jobs $\mathcal{J}$ of n independent aperiodic jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ having the following parameters: a release time, a worst-case execution time (WCET) and a deadline, that are executed on a DVS-enabled processor. Is it possible to minimize the overall energy consumption for all jobs in $\mathcal{J}$ along with the potential recovery operations without deadline violations under any fault scenario with at most k

transient faults? In our work, we use the ES-DVFS scheduling policy [13], which was proved to be optimal in minimizing the total energy consumption for uniprocessor systems under conventional or non-fault-tolerant analysis techniques. To answer this question, we have to find the CPU speed decisions (including the recoveries) in order to minimize the overall energy consumption within predefined timing constraints when no more than k faults occur.
We consider that the processor is equipped with a set of discrete speed values for the whole time interval where $\mathcal{J}$ is executed as a speed schedule.

4. Fault Tolerant Speed Schedule.

4.1. Overview of the Scheduling Scheme. We develop an approach of a fault-tolerant DVFS scheduling for dynamic-priority real-time job set on uniprocessor systems to enhance energy savings while still guaranteeing the timing constraints. The proposed algorithm is based on the Energy Saving - Dynamic Voltage and Frequency Scaling (ES-DVFS) algorithm that we previously proposed in [13].

DEFINITION 4.1. *A job set $\mathcal{J}$ is said to be k-fault tolerant if all jobs and potential recovery operations can be completed before their corresponding deadlines under any fault scenario with at most k transient faults.*
To better understand our approach and before proceeding, we first state some basic definitions and then briefly reiterate the general concept of ES-DVFS.

DEFINITION 4.2. *Given a real-time job set $\mathcal{J}$ of n independent aperiodic jobs such that $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$.*

- *$\mathcal{J}(t_s, t_f)$ stands for the set of jobs contained in the time interval $\phi = [t_s, t_f]$, i.e jobs that are ready to be processed at time t_s and with deadlines not more than t_f. $\mathcal{J}(\phi) = \{J_i \mid t_s \leq a_i < d_i \leq t_f\}$.*
- *$W(\phi)$ denotes the total amount of workload of jobs in $\mathcal{J}(\phi)$ in the time interval $[t_s, t_f]$, that means that the total worst case execution time of jobs that are completely contained in the time interval,*

$$W(\phi) = \sum_{J_i \epsilon \mathcal{J}(\phi)} c_i \tag{4.1}$$

- *The processor load $h(\phi)$ over an interval $\phi = [t_s, t_f]$ is defined as*

$$h(\phi) = \frac{W(\phi)}{t_f - t_s} \tag{4.2}$$

- *The intensity of jobs in the time interval $\phi = [t_s, t_f]$, denoted as $I(\phi)$, is defined as*

$$I(\phi) = \max_{J_j \epsilon \mathcal{J}(\phi)} \left(\frac{\sum\limits_{d_i \leq d_j} c_i}{d_j - (t_f - t_s)} \right) \tag{4.3}$$

- *We consider that the* fault-related overhead *of a time interval $\phi = [t_s, t_f]$, denoted as $W_k(\phi)$ is*

$$W_k(\phi) = W_r(\phi) + W_{TO}(\phi) \tag{4.4}$$

Where $W_r(\phi)$ stands for the worst-case reserved workload to be used in case of recovery, i.e. $W_r(\phi) = k \times (R_l + TO_l)$ and l represents the index of the job with the longest recovery time in $\mathcal{J}(\phi)$. $J_l = \{J_i \mid max(R_i + TO_i),\ J_i \epsilon \mathcal{J}(t\phi)\}$ and $W_{TO}(\phi)$ denotes the overhead imposed by fault detection from regular jobs, i.e.

$$W_{TO}(\phi) = \sum_{J_i \epsilon \mathcal{J}(\phi)} TO_i \tag{4.5}$$

Further, $W_k(\phi) \geq W_{k-1}(\phi)$ for $k \geq 1$, since all the jobs' recovery have strictly positive execution times. For this sake, we restrict our analysis to the k-fault tolerance in such a way that there exists exactly k faults when we investigate the worst-case reserved recovery of failures with at most k faults.

- *The energy demand of a job set $\mathcal{J}$ on the time interval $\phi = [t_s, t_f]$ is*

$$g(\phi) = \sum_{t_s \leq r_k, d_k \leq t_f} E_k(S_k) \tag{4.6}$$

Given a real-time job set $\mathcal{J}$, ES-DVFS was provably optimal in minimizing energy consumption in on-line energy-constrained setting by providing sound dynamic speed reduction mechanisms [13]. ES-DVFS produces an energy efficient technique by scaling down the processor frequency iwhile still guaranteeing the timing constraints. Using this framework, the slowdown factor of the ready jobs to be executed is not fixed in the used interval as the previous work in [10] but is dynamically adjusted on the fly. ES-DVFS is employed as follows:

- *Step 1:* Identify an interval $\phi = [t_s, t_f]$, add the ready jobs to the job queue $\mathcal{Q}$ and select the job J_i with the highest priority.
- *Step 2:* Calculate the effective processor load $h(\phi)$ and intensity $I(\phi)$ using equations 4.2 and 4.3 respectively.
- *Step 3:* Set the speed S_i of job J_i to the maximum between $h(\phi)$ and $I(\phi)$.
- *Step 4:* In case of preemption, update S_i.
- *Step 5:* Remove job J_i from the queue $\mathcal{Q}$.
- *Step 6:* Repeat step (1) - (6) until $\mathcal{Q}$ is empty.

Moreover, we proved that ES-DVFS provides an optimal speed schedule for a given job set $\mathcal{J}$.

LEMMA 4.3. *[13] An optimal speed schedule for a job set $\mathcal{J}$ is defined on a set of time intervals $\phi = [t_s, t_f]$ in which the processor maintains a constant speed $S_i = \max\Big(I(\phi), h(\phi)\Big)$ where $h(\phi)$ and $I(\phi)$ are respectively the workload and intensity of jobs in $\phi = [t_s, t_f]$ and each of these intervals $[t_s, t_f]$ must start at t_s and with deadlines at or earlier than t_f .*

4.2. Concepts for the EMES-DVFS Model. ES-DVFS is proved to be optimal in case of fault-free conditions. Hence, To make the above ES-DVFS fault-tolerant, we adopt an approach, named MES-DVFS, that takes the fault recovery into consideration when calculating the effective processor load and intensity in any interval $\phi = [t_s, t_f]$, i.e. to replace $h(\phi)$ and $I(\phi)$ with $h_m(\phi)$ and $I_m(\phi)$ respectively, such that

$$h_m(\phi) = \frac{\sum_{J_i \epsilon \mathcal{J}(\phi)} c_i + k \times R_l}{d_{max} - W_{TO}(\phi) - k \times TO_l} \tag{4.7}$$

Where d_{max} is the longest deadline in $\mathcal{J}(\phi)$, l stands for the index of the job with the maximum recovery in $\mathcal{J}(t\phi)$ and $W_{TO}(\phi)$ stands for the overall fault-detection overheads for regular jobs as stated in Definition 4.2.

In addition, the intensity of the jobs in $\mathcal{J}(\phi)$ at current time t is

$$I_m(t) = \max_{J_j \epsilon \mathcal{J}(\phi)} \left(\frac{\sum_{d_i \leq d_j} c_i + k \times R_l}{d_j - t - W_{TO}(\phi) - k \times TO_l)} \right) \tag{4.8}$$

Aydin, in [25], showed that the feasibility condition of scheduling a job set $\mathcal{J}$ by using EDF scheduler on a monoprocessor that can tolerate up to k transient faults can be summarized as

THEOREM 4.4. *[25] Given a real-time job set $\mathcal{J}$ that can tolerate a maximum of k faults and with maximum processor speed ($S_{max} = 1$), if for each interval $[t_s, t_f]$, we have*

$$\frac{\sum_{J_i \epsilon \mathcal{J}(t_s, t_f)} c_i + W_{ft}(t_s, t_f)}{t_f - t_s} \leq 1 \tag{4.9}$$

When a fault is detected, and for the sake of enhancing the total savings for both the original jobs and their recovery copies, MES-DVFS runs the copy of the recovered job using a scaled processor speed ($S_i \leq S_{max}$).

However, this may not be energy efficient since, in practice, the fault rate is usually very low.

An extended approach for MES-DVFS (we call it EMES-DVFS), is to execute the recovery copies under the maximum possible processor speed, usually at S_{max}.

Hence, the intensity calculation of the jobs in $\mathcal{J}(\phi)$ can be modified correspondingly, as equation 4.10

$$I_e(t) = \max_{J_j \epsilon \mathcal{J}(\phi)} \left(\frac{\sum\limits_{d_i \leq d_j} c_i}{d_j - t - W_k(\phi)} \right) \tag{4.10}$$

Further, the effective processor load of the jobs in $\mathcal{J}(\phi)$ can also be modified correspondingly, as equation 4.11

$$h_e(\phi) = \frac{\sum\limits_{J_i \epsilon \mathcal{J}(\phi)} c_i}{d_{max} - W_k(\phi)} \tag{4.11}$$

4.3. Description of the EMES-DVFS Scheduler. In what follows, we consider a given set of n jobs $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ that can tolerate up to k faults. Let $\mathcal{Q}(\phi)$ be the list of uncompleted jobs ready for execution at in the time interval $\phi = [t_s, t_f]$. We can formulate our EMLPEDF algorithm to obey the following rules:

Rule 1: The EDF priority order is used to select the future running jobs in $\mathcal{Q}(\phi)$.
Rule 2: The processor is imperatively idle in $[t_s, t_s + 1)$ if $\mathcal{Q}(\phi)$ is empty.
Rule 3: The processor is imperatively busy in $[t_s, t_s + 1)$ if $\mathcal{Q}(\phi)$ is not empty and $0 < C(t_s) \leq C$. Hence, the following steps must be performed:

1. Select the job, say J_i with the highest priority.
2. Calculate the effective processor load $h_e(\phi)$ and intensity $I_e(\phi)$ using equations 4.11 and 4.10 respectively.
3. Set the speed S_{ei} of job J_i to the maximum between $h_e(\phi)$ and $I_e(\phi)$.

Rule 4: If $S_{ei} < S_{min}$, then $S_{ei} = S_{min}$ $\forall$ $J_i \epsilon \mathcal{J}(\phi)$.
Rule 5: If job, say J_j is released with $d_j < d_i$, then update S_{ei} by **Rule 3**.
Rule 6: If job, say J_k is released with $d_k > d_i$, then complete the execution of J_i.
Rule 7: If job, say J_k is released with $d_k > d_i$, and $c_k > d_k - d_i$ then update S_{ei} by **Rule 3**.
Rule 8: Calculate the energy consumption $E_i(S_{ei})$ according to eq. (3.2).
Rule 9: Calculate the remaining energy in the battery at the end of the execution.
Rule 10: Remove job J_i from the queue $\mathcal{Q}(\phi)$.
Rule 11: Repeat step (1) - (8) until the queue $\mathcal{Q}$ is empty.

4.4. Feasibility Analysis. When the job set $\mathcal{J}$ is feasible, it is not difficult to verify that $S_e(\phi) \leq S_m(\phi)$ for a given interval $\phi = [t_s, t_f]$ since $\sum\limits_{J_i \epsilon \mathcal{J}(\phi)} c_i + W_k(\phi) \leq t_f - t_s$, where $S_m(\phi)$ and $S_e(\phi)$ are equal to $\max\left(I_m(\phi), h_m(\phi)\right)$ and $\max\left(I_e(\phi), h_e(\phi)\right)$ respectively.

More importantly, since EMES-DVFS can successfully schedule a given job set $\mathcal{J}$, then we can guarantee the feasibility of the resulted schedule. This is stated in the theorem below.

THEOREM 4.5. *EMES-DVFS can guarantee that the deadlines of all jobs in $\mathcal{J}$ can be met as long as the following two constraints are satisfied : (1) no more than k faults occur; (2) $\forall$ $i \epsilon [1, n]$, where n is the number of jobs in the job set $\mathcal{J}$, we have $S_{ei} \leq 1$ and $C(t) > 0$ $\forall$ t.*

Proof. In EMES-DVFS, for a given time interval $\phi = [t_s, t_f]$, we can only execute jobs and their recovery copies included in the interval. For any job with higher priority, say J_h, and ready at time t, with possible execution overlapping with ϕ , the EMES-DVFS scheduler must preempt the running job and begin executing J_h and hence we will have a new slowdown factor that forces J_h to finish within the interval ϕ. Similarly, for

any job with lower priority (e.g. J_l) with an execution time that can possibly overlap with ϕ, the EMES-DVFS scheduler will update the processor speed and continue execution. J_l is excluded from execution in the time interval ϕ and will be postponed to the next interval. Hence, scheduling decisions are only applied online when any of the following events occur: 1) Event 1: a new aperiodic job arrives and is added to the job queue. 2) Event 2: the current job is completely executed.
Hence, for proving this theorem, it is sufficient to prove that when the processor speed is set to S_{ei}, then we can guarantee that all jobs in ϕ are scheduled in the worst case scenario (i.e. against k faults), as long as $S_{ei} \leq 1$ and the energy reservoir is not fully depleted ($C(\phi) > 0$). We prove this by contradiction. Consider that $J_b = (r_b, c_b, d_b)\ \epsilon\ \mathcal{J}(\phi)$ misses its deadline when the processor speed is set to S_{ei}. Here, we have 2 cases:
Case 1: J_b misses its deadline because of time starvation. Then we must be able to find a time $t \leq r_b$, such that for interval $\phi' = [t, d_b]$, we have $\frac{W(\phi')}{S_{ei}} + W_k(\phi') \geq d_b - t$. Here we have 2 cases:

Case 1a: J_b is not contained in the time interval ϕ (i.e. $d_b \geq t_f$). In this case, event 2 occurs, which means that the current running job, say J_i, will complete its execution without being preempted and job J_b will be executed in the next time interval. This contradicts that J_b misses its deadline.

Case 1b: J_b is contained in the time interval ϕ (i.e. $d_b \leq t_f$). In this case, event 1 occurs, which means that the EMES-DVFS scheduler updates the processor speed schedule for all the ready jobs including the newly arrived one and consequently S_{ei} is updated to another slowdown value S'_{ei} such that $S'_{ei} \geq S_{ei}$ and $\phi' \subseteq \phi$. Here we have 2 cases:

Case 1b1: Effective processor load $h_e(\phi')$ is greater than the intensity $I_e(\phi')$. This means that $S'_{ei}(\phi') = h_e(\phi')$. But, $h_e(\phi')$ is equal to $\frac{W(\phi')}{d_{max} - W_k(\phi')}$ which is greater than or equal to S_{ei}. Since $S'_{ei}(\phi') \geq S_{ei}$, then we have $\frac{W(\phi')}{S'_{ei}} \leq \frac{W(\phi')}{S_{ei}}$. Take Eq. 4.11 into the right-hand side of the above inequality and add $W_k(\phi')$ to both sides. We get $\frac{W(\phi')}{S'_{ei}} + W_k(\phi') \leq d_b - t \leq d_{max} - t$. This violates the assumption that J_b misses its deadline.

Case 1b2: Effective processor load $h_e(\phi')$ is smaller than the intensity $I_e(\phi')$. This means that $S'_{ei}(\phi') = I_e(\phi')$. But, $I_e(\phi')$ is equal to $\max\limits_{J_j \epsilon \mathcal{J}(\phi')} \left(\frac{\sum\limits_{d_i \leq d_j} c_i}{d_j - t - W_k(\phi')} \right)$ which is greater than or equal to S_{ei}. Since $S'_{ei}(\phi') \geq S_{ei}$, then we have $\frac{\sum\limits_{d_i \leq d_j} c_i}{S'_{ei}} \leq \frac{\sum\limits_{d_i \leq d_j} c_i}{S_{ei}}$. Take Eq. 4.10 into the right-hand side of the above inequality and add $W_k(\phi')$ to both sides. We have $\frac{W(\phi')}{S'_{ei}} + W_k(\phi') \leq d_b - t$. This violates the assumption that J_b misses its deadline.

Case 2: J_b misses its deadline because of energy starvation. This means that $.d_b$ is missed with energy demand $g(d_b) = 0$. Then, we must find a time $t_0 \leq d_b$ where a job with deadline after d_b is released and no other job is ready just before t_0 and the battery is fully replenished i.e. $C(t_0) = C$. The processor is busy at least in the time interval $\phi' = [t_0, d_b]$. Here we also have 2 cases:

Case 2a: No job with deadline greater than d_b executes within the time interval ϕ'. This means that all the jobs that execute within ϕ' have release time greater than or equal to t_0 and a deadline no more than d_b. The amount of energy needed to fully execute these tasks is $g(\phi')$. But since the processor is always busy in the time interval ϕ', then jobs are executed with the minimum possible speed. Further, the energy reservoir is fully charged at t_0. Consequently, $g(\phi') < C(t_0) < C$. We conclude that all ready jobs within ϕ' can be fully executed with no energy starvation, which contradicts the deadline violation at d_b with $C(d_b) = 0$.

Case 2b: At least one job, say J_m is released within time interval ϕ' and with with $r_m > r_b$. Here we have 2 cases:

Case 2b1: J_m is released with $d_m < d_b$, therefore we have to update S_{ei} by Rule 3. Let t_2 be the latest time where J_m is executed. As d_m is lower than d_b and jobs are executed according to preemptive EDF, we have $r_m \geq r_b$ and J_b is preempted by the higher priority job J_m. Thus, the processsor speed must be updated, otherwise d_m will be violated. Since the processor is busy all the times in $[t_2, d_b]$ and the job set $\mathcal{J}$ is time-feasible, then S_{em} will be the minimum speed for the execution of J_m and consequently $g(t_2, d_b) < C(d_b)$. Hence, the amount of energy that J_m require is at most $g(t_2, d_b)$. That contradicts deadline violation and

$C(d_b) = 0$.

Case 2b2: J_m is released with $d_m > d_b$. We consider two cases: (i) $c_m < d_k - d_b$, hence J_b will complete its execution (Rule 6) and the proof is therefore similar to *case 2a*. (ii) $c_m > d_k - d_b$, hence S_{eb} must be updated (Rule 7) and the proof is therefore similar to *case 2b1*.

Since all jobs in $\mathcal{J}$ are successfully executed within time intervals in EMES-DVFS and all running jobs within the corresponding time intervals are still schedulable when applying their corresponding speed, we prove the theorem.

□

We state the optimality of EMES-DVFS by proving that a job set $\mathcal{J}$ is feasible in a k-fault-tolerant if and only if all the jobs in $\mathcal{J}$ are executed without violating time and energy constraints. This violation is due to one of the two following reasons: either job, say J_i lacks time (Lemma 4.6) or job J_i lacks energy (Lemma 4.7) to complete its execution before or at deadline d_i. The time starvation occurs when deadline d_i is missed with the energy reservoir not exhausted at d_i. On the other side, the energy starvation case is when the energy reservoir is fully depleted at d_i and J_i is not completed.

Further, the feasibility of the EMES-DVFS scheduler is guaranteed, which is formulated in Lemma 4.6 and Lemma 4.7.

LEMMA 4.6. *A real-time job set $\mathcal{J}$ can be time-feasible in a k-fault-tolerant manner by EMES-DVFS if and only if all the jobs in $\mathcal{J}$ can still respect their deadlines when they are executed according to the processor speeds determined by EMES-DVFS for every time interval $[t_s, t_f]$.*

Proof. Only if part. Directly follows Theorem 4.5.

If part. Suppose the contrary. Let us consider $\mathcal{J}(\phi)$ as the set of jobs contained in the time interval $\phi = [t_s, t_f]$, this means jobs that are ready to be processed at or after time t_s and with deadlines at or earlier than t_f. We denote a fault pattern $f = \{f_1, f_2, \cdots, f_n\}$, where f_i refers to the number of faults affecting job J_i and its recovery. Hence, we say that f is a k-fault pattern if the total number of faults is exactly k. Formally $h_e(\phi) \leq 1$ for all intervals $[t_s, t_f]$. However, there is a j-fault pattern $j \leq k$ (say f^j) resulting in deadline miss(es). Let us assume that the first deadline violation occurs at $t = d_i$ and that t_0 is the latest time preceding d_i such that either the processor is idle or a job (recovery) of deadline $> d_i$ is executing.

We note that the time t_0 is well-defined in a way that it corresponds to a job release time. In addition, the processor is continuously busy executing jobs (recovery) in the time interval $\phi^0 = [t_0, d_i)$. Now, let us denote $f^0 \subset f^j$ be the subset of faults affecting jobs in the time interval ϕ^0. Note that the number of faults in f^0 is obviously smaller than k. Since EDF is a work-conserving scheduling algorithm, this means that the processor is never kept idle unless there are no ready jobs, the deadline violation at d_i and the above definition of t_0 imply that the available processor time in the interval time interval ϕ^0 was not sufficient to accommodate the increase in the processor demand even there is no energy starvation in the interval ϕ^0 (the battery is not fully replenished at time d_i). Consequently, we obtain $\sum_{J_i \epsilon \mathcal{J}(\phi^0)} c_i + j \times R_l > d_{max} - W_{TO}(\phi^0) - j \times TO_l$, where j is the number of faults in ϕ^0 and l stands for the index of the job with the longest recovery time in $\mathcal{J}(\phi^0)$. But since $\sum_{J_i \epsilon \mathcal{J}(\phi)} c_i + W_k(\phi) \leq d_i - t_0$ ($h_e(\phi) \leq 1$) and $W_k(\phi) > W_k(\phi^0)$ and $\sum_{J_i \epsilon \mathcal{J}(\phi)} c_i > \sum_{J_i \epsilon \mathcal{J}(\phi^0)} c_i$, we get $\sum_{J_i \epsilon \mathcal{J}(\phi^0)} c_i + W_k(\phi^0) \leq d_i - t_0$ contradicting our assumption that a deadline violation occurs at d_i.

□

LEMMA 4.7. *A real-time job set $\mathcal{J}$ can be energy-feasible in a k-fault-tolerant manner by EMES-DVFS if and only if the deadlines of all the jobs in $\mathcal{J}$ can be met when they are executed based on the processor speeds determined by EMES-DVFS considering that for every time interval $[t_s, t_f]$, $g(t_s, t_f) > 0$.*

Proof. Only if part. Directly follows Theorem 4.5.

If part. Suppose the contrary. Let us consider $\mathcal{J}(\phi)$ as the set of jobs contained in the time interval $\phi = [t_s, t_f]$. We also denote a fault pattern $f = \{f_1, f_2, \cdots, f_n\}$, where f_i refers to the number of faults affecting job J_i and its recovery. Hence, we say that f is a k-fault pattern if the total number of faults is exactly k and the energy in the reservoir is sufficient to execute all jobs in ϕ. Formally $g(\phi) > 0$ for all intervals $[t_s, t_f]$. However, there is a j-fault pattern $j \leq k$ (say f^j) resulting in deadline miss(es) due to energy starvation. Let us assume that the energy reservoir becomes empty at $t = d_i$ ($C(d_i) = 0$) and that t_0 is the latest time preceding d_i such that

the processor is still executing a job (recovery) of deadline $< d_i$.
We note that the time t_0 is well-defined in a way that it corresponds to a job release time. In addition, the processor is continuously busy executing jobs (recovery) in the time interval $\phi^0 = [t_0, d_i)$. Now, let us denote $f^0 \subset f^j$ be the subset of faults affecting jobs in the time interval ϕ^0. Note that the number of faults in f^0 is obviously smaller than k. Since EDF is a work-conserving scheduling algorithm, this means that the processor is never kept idle unless there are no ready jobs, the deadline violation at d_i and the above definition of t_0 imply that the available energy in the reservoir in the time interval ϕ^0 was not sufficient to accommodate the increase in the energy demand even there is no time starvation in the interval ϕ^0, this means $\sum_{J_i \epsilon \mathcal{J}(\phi^0)} c_i + j \times R_l \leq d_{max} - W_{TO}(\phi^0) - j \times TO_l$, where j is the number of faults in ϕ^0 and l stands for the index of the job with the maximum recovery time in $\mathcal{J}(\phi^0)$. But since the job set $\mathcal{J}(\phi)$ is feasible, then $g(\phi) > 0$. In addition, the energy demand in ϕ is greater than the energy demand in ϕ^0, since the number of faults in ϕ (k) is more than that in ϕ^0 (j), i.e. $g(\phi) > g(\phi^0)$. Hence, we get $g(\phi) > g(\phi^0) > 0$ contradicting our assumption that a deadline violation occurs at d_i because of energy starvation. □

Now, we may draw Theorem 4.8, a major result of optimality for uniprocessor scheduling in a k-fault-tolerant manner by EMES-DVFS with time and energy constraints.

THEOREM 4.8. *The EMES-DVFS scheduling algorithm is optimal for a k-fault-tolerant model.*

Proof. According to Lemma 4.6, EMES-DVFS can schedule a given set of jobs $\mathcal{J}$ in a k-fault-tolerant manner, without violating timing constraints when the energy demand is lower than the maximum energy that is available in the reservoir. According to Lemma 4.7, EMES-DVFS can schedule a given set of jobs Γ in a k-fault-tolerant manner, without violating energy constraints when the processor demand is cannot exceed the maximum available processor time that could be available in any given time interval. As a conclusion, if EMES-DVFS can schedule a given set of jobs $\mathcal{J}$ for a given time or/and energy constraints without time starvation and energy starvation, are the only two reasons for deadline violations, then we conclude that EMES-DVFS is optimal. □

5. Evaluation. In this section, we study the performance of four scheduling algorithms: EMES-DVFS, MES-DVFS, NPM and LPSSR proposed in [20]. NPM scheme executes jobs with maximum frequency and does not scale down the voltage/frequency.

We developed a discrete event-driven simulator in C that generates a job set $\mathcal{J}$ where the number of jobs varies from 10 to 50. The simulation is repeated 100 times for the same number of jobs.

For the sake of clarity, we use NPM as a reference schedule that represents the schedule of given set of jobs $\mathcal{J}$ without incorporating DVFS. This means that all jobs or recoveries in $\mathcal{J}$ are executed under the maximum processor speed S_{max}. We consider that all the plotted energy consumptions are normalized to NPM. However, to give LPSSR a fair chance, we consider the same parameters as used in [20]. Hence, we consider the following parameters: We assumed that $\alpha = 2$, $C_{ef} = 1$, $P_{ind} = 0.05$, and S_{min} is set to 0.25.
The proposed algorithms are tested with randomly generated job sets as follows: the choice of the arrival time a_i and the relative deadline of each job J_i is assumed to be distributed uniformly in the time interval $[0s, 100s]$ and $[50s, 100s]$ respectively. Moreover, the worst case execution time c_i is randomly generated such that $c_i < d_i$. The timing and energy overhead of detecting faults is considered as 10% of the WCET and its energy consumption respectively. As for the fault arrival rate, we consider 2 cases: safety-critical real-time system with range of 10^{-10} to 10^{-5} /hour or in harsh environment with a range between 10^{-2} and 10^{2} /hour.

All simulation results are computed on a discrete DVFS processor that operates on 8 frequency levels $\{1.00, 0.86, 0.76, 0.67, 0.57, 0.47, 0.38, 0.28\}$ as in the PentiumM processor.

We report here two sets of experiments. The first set is designed to show the energy consumption of the 4 approaches by varying the number of jobs. In the second experiment set-up, we compare the energy consumption by varying the number of faults.

5.1. Experiment 1: Energy Consumption by Varying the Number of Faults. In this set of simulations, we evaluated the impact of the number of faults on energy savings. In this experiment, we took a fixed number of jobs equal to 15 and the number of tolerated faults varies between 1 and 10. Again, we repeat the experiment for 100 times with different generated test cases but with the same number of faults. We took then the average results, which are shown in figure 5.1.

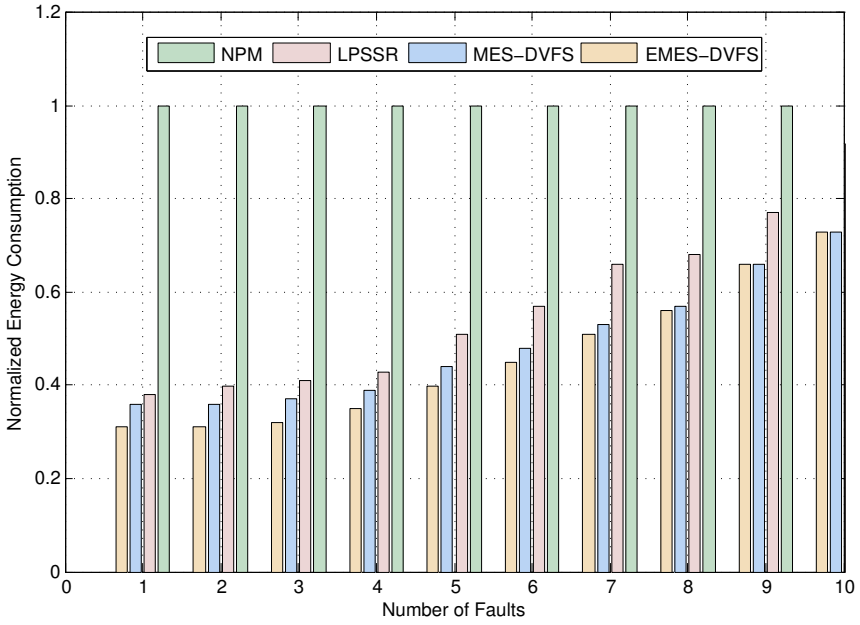

FIG. 5.1. *Energy savings by varying the numbers of faults.*

From the figure 5.1, we can find that EMES-DVFS and MES-DVFS can achieve energy savings compared to LPSSR and NPM. Clearly, we find that the energy consumptions by the four schedulers increase rapidly as the number of faults increases, since more expected energy may be consumed due to the increased number of recovery jobs being executed, which in turn limits the maximum amount of dynamic slack used. However, as the number of faults increases, the energy consumption in EMES-DVFS and EMES-DVFS grows but less dramatically.

As illustrated in figure 5.1, the EMES-DVFS and MES-DVFS approaches attain respectively around 19% and14% more energy saving than LPSSR. The reason is that the optimal dynamic slack time to minimize the expected energy consumption is used to the maximum extent by employing speed assignment on the fly. On the contrary, LPSSR is significantly affected when we increase the number of system faults and the system consumes of about 22% additional energy when we increase the fault occurrences from 1 to 10. On the other hand, EMES-DVFS could tolerate up to 5 times more faults with same energy as consumed by LPSSR.

As a conclusion, the advantage of our approaches (EMES-DVFS and MES-DVFS) over the other two (LPSSR and NPM) in terms of energy savings is evident in this experiment where EMES-DVFS and MES-DVFS can still guarantee tolerance even under 10 faults and with more energy saving than LPSSR and the energy savings drops around 19% under LPSSR when we compare it with EMES-DVFS.

5.2. Experiment 2: Energy Consumption by Varying P_{ind}. In this experiment, we study the impact of frequency-independent power P_{ind} on energy savings. P_{ind} varies between $[0, 0.3]$ for each job and the number of jobs is fixed at 15. According to figure 5.2 , the larger the P_{ind}, the higher the energy consumption. This is due to the fact that as the P_{ind} increases, the contribution of frequency independent energy consumption becomes more dominant, the energy-efficient frequency increases and consequently DVFS has fewer opportunities to be applied. Even under this situation, EMES-DVFS still has the best performance in terms of energy consumption (EMES-DVFS attains approximately 18% more energy saving than LPSSR).

5.3. Experiment 3: Percentage of feasible Job Set. In this experiment, we take interest in the percentage of feasible job set that respect their deadlines with the four scheduling algorithms by varying the energy storage capacity. From this experiment, we can deduce two measures. The first one gives us an indication about the percentage of time during which all deadlines are still respected. The second one gives, for each

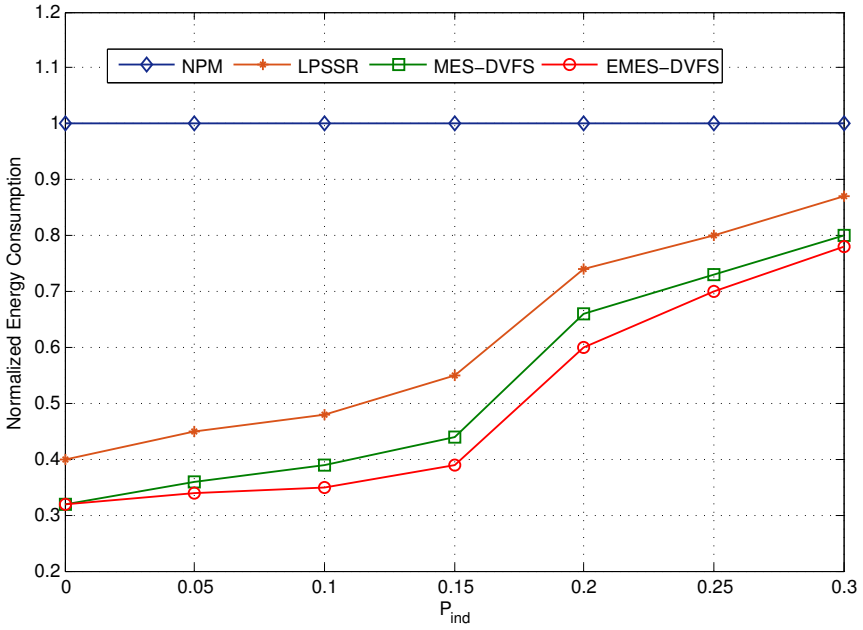

FIG. 5.2. *Energy savings by varying* P_{ind}.

approach and for a given processor load, the minimum size of the storage that ensures time and energy feasibility. We report here the results of two simulation studies where the processor load is set to 0.4 and 0.8, respectively.

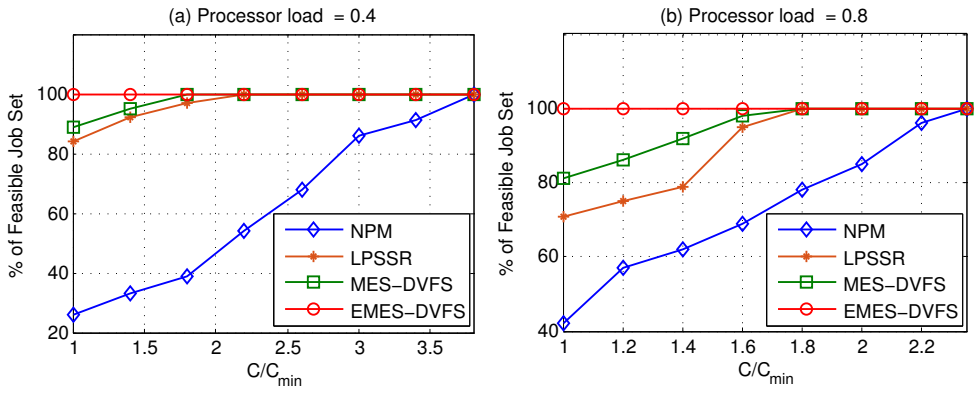

FIG. 5.3. *Percentage of feasible job set. (a) Low processor load. (b) High processor load.*

Figure 5.3 depicts the percentage of feasible job sets that meet their deadlines over the energy storage capacity C. For each job set, we compute the minimum storage capacity C_{min} which permits achieving time and energy feasibility under EMES-DVFS. We then begin to increase the energy storage capacity till the 4 approaches achieve neutral operation.

Under low processor load (figure 5.3a), it is observed that 100% of job sets meet their deadlines under EMES-DVFS when the energy storage capacity is 4510 energy units, i.e. $C = C_{min} = 4510$ energy units. We start then to increase C till it reaches 8118 where MES-DVFS becomes feasible. this means that means that EMES-DVFS can provide the same level of performance with a storage unit which is about 1.8 times less. The increase in the storage capacity will continue to increase till LPSSR and NPM becomes feasible where the energy storage unit must be respectively more than 2.2 and 3.8 times bigger with LPSSR and NPM to maintain zero

deadline miss, compared with EMES-DVFS.

The results for high processor load (figure 5.3b) follow the same trend. Unlike the previous experiment, the relative performance gain of EMES-DVFS in terms of capacity savings is decreasing when the processor load is increasing. EMES-DVFS obtains respectively capacity savings of about 37%, 44% and 57% compared with MES-DVFS, LPSSR and NPM.

It is important here to note that the four approaches require exactly the same storage size when the processor load is equal to 1 since the processor is continuously busy and there is no chance to apply DVFS.

In summary, this experiment points out that the proposed EMES-DVFS approach is very effective in reducing deadline miss rate and storage size even under high processor load. And lower is the processor load rate, higher is the capacity saving and our approach will then outperform the others by a high amount of energy savings.

6. Conclusions. In this paper, we presented and evaluated a novel approach, which aims to minimize energy consumption when scheduling a set of real-time jobs that can tolerate up to k transient faults while still respecting time and energy constraints. We explore the reserved slacks generated during run-time to the maximum extent in such a way that all the available slack time is used for energy reduction, which is carried out using dynamic voltage and frequency scaling (DVFS). Under this notion, we propose an algorithm that estimates an optimal speed reduction mechanism which maintains feasibility within predefined timing constraints when no more than k faults occur.

Our scheduler dynamically adjusts the jobs' slowdown factors by utilizing run-time slacks which may be increased for recovery demands of the system. It differs from the existing approach where job frequencies assignments are predetermined, and hence it is more flexible and adaptive in minimizing energy consumption while still keeping the systems reliability at a desired level. In addition, we presented two feasibility tests for recovery schemes under variable processor speed which decouples the time and energy constraints. The experimental results demonstrate that the proposed algorithm can significantly improve the energy savings compared with the previous works.

For future work, we will explore the adaptation of the proposed approaches to fixed priority environments in real-time energy harvesting systems.

Acknowledgments. This work was fully supported by a research grant from the Lebanese University.

REFERENCES

[1] E. Bini and G. C. Buttazzo. *Measuring the performance of schedulability tests.* Real-Time Systems, 30(1-2):129-154, May 2005.

[2] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. Cook. *Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors.* Micro, IEEE, 20(6):26-44, Nov 2000.

[3] R. Gupta. *Dynamic voltage scaling for system-wide energy minimization in real-time embedded systems.* Proceedings of the International Symposium on Low Power Electronics and Design ISLPED '04, pages 78-81, Aug 2004.

[4] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. *Power-aware scheduling for periodic real-time tasks.* IEEE Transactions on Computers, 53(5):584-600, 2004.

[5] H. Aydin, V. Devadas, and D. Zhu. *System-level energy management for periodic real-time tasks.* In Proc. of IEEE Real-Time Systems Symposium (RTSS), pages 313322, Dec. 2006.

[6] V. Devadas and H. Aydin. *On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications.* In Proc. ACM Conference on Embedded Systems Software (EMSOFT'08), 2008.

[7] D. Siewiorek and R. Swarz. *Reliable Computer Systems: Design and Evaluation.* Natick, MA: A. K. Peters, Ltd., 1998.

[8] Srinivasan J, Adve SV, Bose P, Rivers J, Hu CK. *Ramp: A model for reliability aware microprocessor design.* IBM Research Report, RC23048, 2003.

[9] Castillo X, McConnel SR, Siewiorek DP. *Derivation and calibration of a transient error reliability model.* IEEE Trans Comput 31:658671, 1982.

[10] F. Yao, A. Demers, and S. Shenker. *A scheduling model for reduced cpu energy.* In Foundations of Computer Science, Proceedings., 36th Annual Symposium on, pages 374-382, oct 1995.

[11] Y. Zhang, K. Chakrabarty, and V. Swaminathan. *Energy-aware fault tolerance in fixed-priority real-time embedded systems.* In Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design, ICCAD'03, 2003.

[12] H. Huang and G. Quan. *Leakage aware energy minimization for real-time systems under the maximum temperature constraint.* In Design, Automation Test in Europe Conference Exhibition (DATE), 2011, pages 1-6, March, 2011.

[13] Hussein El Ghor, E. M. Aggoune. *Energy efficient scheduler of aperiodic jobs for real-time embedded systems*, International Journal of Automation and Computing, pages 1-11, 2016.
[14] Hussein EL GHOR, Maryline CHETTO. *Energy Guarantee Scheme for Real-time Systems with Energy Harvesting Constraints*. International Journal of Automation and Computing, to appear.
[15] Baoxian Zhao, Hakan Aydin and Dakai Zhu. Energy Management under General Task-Level Reliability Constraints. IEEE 18th Real Time and Embedded Technology and Applications Symposium, 2012.
[16] Zhu D, Melhem R, Mosse D. *The effects of energy management on reliability in real-time embedded systems*. In: ICCAD '04, Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design, IEEE Computer Society, Washington, DC, pp 35-40, 2004.
[17] R. Melhem, D. Mosse, and E. Elnozahy. The interplay of power management and fault recovery in real-time systems, IEEE Transactions on Computers, 53(2):217-231, Feb 2004.
[18] Y. Zhang and K. Chakrabarty. *A unified approach for fault tolerance and dynamic power management in fixed-priority real-time embedded systems*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 25(1):111-125, jan. 2006.
[19] B. Zhao, H. Aydin, and D. Zhu. *Generalized reliability-oriented energy management for real-time embedded applications*. In 48th ACM/EDAC/IEEE Design Automation Conference (DAC), pages 381-386, june 2011.
[20] Qiushi Han, Linwei Niu, Gang Quan, Shaolei Ren and Shangping Ren. *Energy efficient fault-tolerant earliest deadline first scheduling for hard real-time systems*. Real-Time Systems 50:592-619, 2014.
[21] T. D. Burd and R. W. Brodersen. *Energy efficient cmos microprocessor design*. In Proc. of The HICSS Conference, Jan. 1995.
[22] J. W. S. W. Liu. Real-time Systems, NJ, USA: Prentice Hall, 2000.
[23] P. Hazucha and C. Svensson. *Impact of cmos technology scaling on the atmospheric neutron soft error rate*. IEEE Trans. on Nuclear Science, 47(6) 2586-2594, 2000.
[24] Pradhan DK. *Fault-tolerant computer system design*. Prentice-Hall Inc, Upper Saddle River, 1996.
[25] Aydin H. Exact fault-sensitive feasibility analysis of real-time tasks. IEEE Trans Comput 56(10):1372-1386, 2007.

Edited by: Dana Petcu
Received: Sep 4, 2018
Accepted: Nov 3, 2018