Feedback-Based Energy Management in a Standby-Sparing Scheme for Hard Real-Time Systems

Mohammad Khavari Tavana  Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
tavana@ce.sharif.edu

Mohammad Salehi  Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
mohammad_salehi@ce.sharif.edu

Alireza Ejlali  Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
ejlali@sharif.edu

Abstract—The interaction between fault tolerance and energy consumption is an interesting avenue in the realm of designing embedded systems. In this paper, a scheme for reducing energy consumption in conventional standby-sparing systems is introduced. In the proposed method, the primary unit exploits dynamic voltage scaling (DVS) and dynamic power management (DPM) is employed for the spare unit. The framework which is used in the primary unit is composed of a feedback system to follow up workload along with a three-layer yet light-weight energy manager which guarantees hard real-time constraints of the system. Moreover, an optimal approach (but not practical) as a margin for the minimum energy consumption of this system is presented and the capability of other methods in reducing energy consumption is compared. Simulation results show an improvement in energy saving as compared with previous works and also show that the proposed method is near optimal for task sets with different dynamic workloads.

Index Terms—Energy management; Fault-tolerance; Hard real-time system; Feedback control;

I. INTRODUCTION

A wide range of real-time embedded systems are hard real-time which are used in safety-critical applications, i.e., failure in these systems may jeopardize humans life, lead to severe economic loss, or result in extensive environmental damage [1]. These systems must be designed to provide fault tolerance capability and high reliability. Examples include medical care devices, avionics systems, surveillance systems and nuclear power plants. Furthermore, the total energy consumption for embedded systems which are battery-operated is one of the major concerns and has attracted a lot of research interest [3,4,5,10,11]. Fault tolerance in real-time systems can be achieved by hardware [2,3] or time redundancy [4,5]. Time-redundancy techniques try to use slack time to tolerate faults by performing recovery executions. These are relatively cost-effective techniques for tolerating faults, but for those applications where high predictability is a critical issue using time redundancy can be a risky approach indeed [2]. On the other hand, hardware-redundancy techniques which employ redundant hardware resources incur substantial energy overhead to the system. Therefore, hardware redundancy in energy constrained hard real-time systems must be employed intelligently.

In this paper, we propose a new standby-sparing technique which uses feedback to follow up workload and decrease energy consumption. There are two traditional types of standby-sparing: i) hot standby-sparing where the spare operates in parallel with the primary unit. ii) cold standby-sparing where the spare power is off until needed to replace the primary unit. From the energy consumption point of view, cold sparing has advantages over hot sparing, but due to the time required to power up the spare, this scheme cannot be employed in hard real-time systems. On the other hand, in hot sparing where the spare operates in parallel with the primary unit, energy overhead is significant. An intermediate approach which can be suitable for energy-constrained hard real-time systems is proposed by [3], where the spare is neither a cold nor a hot one. In this system, the spare is always on, but does not work in parallel with the primary unit and may execute a task before the primary unit fails due to the time constraints of the system.

Like in [3], in our proposed system, the spare unit employs dynamic power management (DPM) and the primary unit uses dynamic voltage scaling (DVS) to reduce the total energy consumption of the system. Using DVS, tasks may be lengthened in the primary unit which may lead to more acting time of the spare unit. Therefore, for determining the supply voltage, the overall energy consumption in both units must be considered. Unlike in [3], in this paper, a framework for energy management of such a system is developed which determines the supply voltage of the primary unit based on a feedback system and slack distribution algorithm (section IV).

In the realm of energy-constrained real-time systems several works have been done. Some research works, e.g., [4,5] have proposed using time redundancy to tolerate faults while DVS is exploited to decrease energy consumption in the system. Due to the trade-off available in using time redundancy in these systems, (for fault tolerance or lowering energy consumption) voltage selection must be done carefully. These works are based on worst-case execution times and do not consider slack reclamation. Moreover, time-redundancy techniques are not suitable for tight deadline applications [7], [8] has also considered resource conflict between time redundancy techniques and DVS on using slack and proposed to employ a combination of time and information-redundancy, [9] has proposed a scheme to reduce the energy overhead of TMR which is applicable in real-time systems. This scheme allows one of the modules...
slow down processing while finishing the computation before the deadline is guaranteed. [11] has exploited DVS for a dual-processor system to decrease the energy consumption. Both processors engage DVS such that energy consumption becomes minimized. In this work, reliability challenges related to using DVS is overlooked. Due to the negative effect of voltage scaling on systems reliability, [12] has proposed a reliability aware power management for a dual-processor system which is used in real-time systems. In this approach, whenever the available slack is large enough to provide the extra copies of the task, switching to a lower voltage is allowed. [3] has proposed a low-energy standby-sparing technique where DVS is used in the primary unit and DPM is used in the spare unit. Although lowering voltage in primary unit can jeopardize reliability, a reliable spare is ready to take place the primary unit in case of errors. Due to the random execution time of the tasks, they proposed a stochastic optimization approach for decreasing energy consumption.

The main contributions of this paper are:

- We employ a PID controller to exploit the correlation between the past history of workload and its near future. Based on an analytical approach, we propose an optimum energy management scheme for individual tasks which minimizes the overall energy consumption, including both the primary and spare units.
- We propose two light-weight heuristic approaches in order to distribute slack time to reduce the overall energy consumption for multiple tasks in the standby-sparing system.
- With respect to the convexity of the energy consumption in proposed standby-sparing system, we develop a static approach which minimizes the overall energy consumption. This approach is not applicable in the real world because needs a look ahead knowledge of all tasks execution times. However, this is just used for comparison and as a lower bound for the energy consumption of the standby-sparing system.

The proposed method has been tested with several different workload patterns and synthetic tasks and has been shown to have better proficiency as compared to former methods.

The rest of the paper is organized as follows. Section II and III present the system and energy consumption models respectively. Section IV presents the proposed three-layer framework for decreasing the energy consumption. Section V presents an optimal but impractical solution for energy minimization in the standby sparing system. Sections VI and VII present the simulation results and conclusions respectively.

II. SYSTEM MODEL

In this paper, we consider a standby-sparing system which consists of two homogeneous processors, i.e., the primary and spare units. The primary unit is the main processor and its output is considered as the system output in the error-free scenario. Whenever the primary unit fails, the system will switch to the spare unit and continues task execution.

An error detection mechanism must be applied in order to determine if the tasks finish successfully or not. We assume the error detection mechanism is part of the software architecture and the overhead imposed by the error detection is considered as part of the task execution time [3] [13]. Whenever the primary unit finishes a task execution successfully, will inform the spare unit through some control signals which provides for synchronization purposes. In case of any transient errors detected by primary unit or even a complete failure, the spare has this capability to take over the primary unit at any time and feeds the system output.

In this work, we consider applications consisting of \( n \) real-time tasks \( \Gamma = \{T_1, T_2, \ldots, T_n\} \) which must be executed on the standby sparing system. The tasks share a common global deadline \( D \), which is also the period (or frame) of the task set. The group of tasks is scheduled based on the precedence which modeled by a task graph. A sample task graph is composed of some nodes and vertices (e.g. Fig. 1a). Each node in the task graph represents a task while the directed edges represent the control or data dependencies between the tasks. We assume the tasks are schedulable effectively with the existing static scheduling algorithms. A sample schedule of the task graph Fig. 1a has been shown in Fig. 1b. The worst-case execution time (WCET) for the task \( T_i \) at the maximum frequency \( f_{\text{max}} \) is denoted by \( WT_i \) and has been written beside each node. Also, each task has an actual execution time \( AT_i \) which varies with respect to the system workloads. Since the primary unit uses DVS, each task is executed at the supply voltage \( V_i \), which may be less than the maximum supply voltage \( V_{\text{max}} \).

The duty of the energy manager in the system is to find the voltage of each task properly such that reduces the overall energy consumption without violating timing constraints. We consider the normalized supply voltage \( \rho_i \) as follows:

\[
\rho_i = \frac{V_i}{V_{\text{max}}} \tag{1}
\]
III. ENERGY CONSUMPTION MODEL

By using DVS, the energy consumption of the primary unit when operating at the scaled supply voltage $V_i$ is [14]:

$$E(T_i) = C_{eff} V^2 f_i \left(\frac{AT_i}{\rho_i}\right)$$

(2)

where $C_{eff}$ is the effective switching capacitance, $V_i$ and $f_i$ are respectively the scaled supply voltage and operational frequency at which the system executes the task $T_i$, and $AT_i/\rho_i$ is the task execution time which is scaled by $\rho_i$ due to DVS. Note that, for DVS, there is almost a linear relationship between $V_i$ and $f_i$ [14], hence $\rho_i = V_i/V_{max} = f_i/f_{max}$ where $V_{max}$ is the maximum voltage corresponding to the maximum operational frequency $f_{max}$. Therefore, Eq. 2 can be written as:

$$E(T_i) = C_{eff} V_{max}^2 f_{max} \rho_i^2 AT_i$$

(3)

Because $C_{eff} V_{max}^2 f_{max}$ is constant, we normalized the energy consumption equation by removing $C_{eff} V_{max}^2 f_{max}$. Consequently, the normalized energy consumption of the processor while executing the task $T_i$ can be written as:

$$NE(T_i) = \rho_i^2 AT_i$$

(4)

IV. PROPOSED FEEDBACK-BASED SYSTEM

Hard real-time scheduling approaches rely on a priori knowledge of the worst-case execution times of the tasks to guarantee meeting the system deadlines. But, it has been shown for many real-world applications that the task execution times may be significantly less than their worst-case execution time [15]. If there is a correlation between the execution time of the task instances (jobs), feedback control techniques have been shown to be effective in soft [16] and hard [17] real-time systems.

As Fig. 2 shows, our proposed system framework employs a PID controller to track workload fluctuations. The error value $e_{ij}$ is computed periodically by differentiating the controlled variable ($AT$) and the set point ($ET$), where $ij$ indicates the $j$th job of the task $T_i$. The error value $e_{ij}$ is given by:

$$e_{ij} = AT_{ij} - ET_{ij}$$

(5)

where $AT_{ij}$ and $ET_{ij}$ are the actual and estimated execution time of the $j$th job of the task $T_i$ respectively. The PID controller uses $e_{ij}$ to calculate the estimated execution time of the next job ($j + 1$). The parallel form [16] of the PID controller which is employed by the proposed system [17] is:

$$\Delta ET(T_{ij}) = K_p e_{ij} + K_I \sum_{I} e_{ij} + K_D \frac{e_{ij} - e_{ij-(D-W)}}{D-W}$$

(6)

where $K_p, K_I$, and $K_D$ are the proportional, integral and derivative coefficients respectively. The proportional coefficient affects the system speed to adapting errors, the integral coefficient increases the system accuracy, and the derivative coefficient increases the speed and stability of the system. The $I$ window and $D$ window are the integral and derivative windows respectively and show how many of the former tasks instances contribute to estimate the next execution time. A positive value of $\Delta ET_{ij}$ means that the task execution time is increased and consequently a negative value shows a decrease in the task execution time. The estimated execution time for the next job is computed by Eq. 7:

$$ET_{ij(j+1)} = ET_{ij} - \Delta ET_{ij}$$

(7)

The proposed energy manager consists of three main units.

1) **Slack distributor**: Exploits $ET_{ij}$, $ET_{ij(i+1)}$, ..., $ET_{nj}$ to distribute the slack time globally in order to reduce the overall energy consumption.

2) **Voltage/Frequency selector**: Uses the slack time passed by the slack distributor unit to locally minimize the energy consumption of the task $T_i$.

3) **Task splitter**: Guarantees the deadlines, by considering $ET_{ij}$ and the operational frequency determined by the voltage/frequency selector.

Each of these will be discussed in more detail in the following sections.

A. Task Splitting

The minimum energy consumption for a task set in a single-processor system is achieved by the even-distribution of slack times among the tasks [14]. Finding an optimal solution needs a priori knowledge of the tasks execution times. In other words, it is not possible to derive the optimal speed assignment without knowing the actual execution times in advance [3,17]. The task splitting is a simple scheme which tries to consume all the slack budget for those parts of the tasks which their execution is more probable. This scheme reduces the energy consumption more efficiently. Moreover, it can guarantee the deadlines despite abrupt changes in the workload.

Using task splitting each task $T_i$ is divided into two subtasks $T_i^A$ and $T_i^B$ (see Fig. 3). Fig. 3a shows the task $T_i$ with the worst-case execution time $WT_i$ and the relative deadline $d_i$. Even though, all the tasks share a common global deadline, there is also a relative deadline for each of them which is equal to $d_i = D - \sum_{j=1}^{n} WT_j$. If any tasks in the application miss their relative deadlines, may jeopardize strict timeliness and hard real-time property of the system.

The available slack time $L_i$ for the task $T_i$ is $d_i - WT_i$ which is obtained from the previous tasks, i.e., $T_1$ to $T_{i-1}$. The available slack time is created when the previous tasks
consume less than their worst-case execution times. As Fig. 3b shows, the task $T_i$ has been split into two parts, $T_i^A$ which is the part that is likely to be executed and is equal to $ET_i$, and $T_i^B$ which has a low probability to be executed and is equal to $WT_i - ET_i$. The entire slack budget is consumed for the first portion of the task $T_i^A$ which is more probable to be executed and the second portion $T_i^B$ is executed with the highest frequency to guarantee deadlines. After determining $\rho_i$ by the voltage/frequency selector, the duty of the task splitter is just setting the time at which the processor must switch to the highest frequency. This can be done by adding the current time to $ET_i/\rho_i$. This aggressive approach for using the slack times is used in [17] and [20] to decrease the energy consumption in a single-processor system.

### B. Voltage/Frequency Selection

In the proposed standby-sparing technique, the primary unit uses DVS and employs the task splitting to exploit the slack time more effectively for the energy minimization purposes. The spare unit just uses DPM. In other words, it postpones executing the given tasks as much as possible in order to minimize the time overlap between the primary and spare units. The spare unit can delay executing tasks up to the available slack time [3]. The voltage/frequency selector assigns a frequency (voltage) to the task $T_i$ considering $ET_i$ and the relative deadline such that the expected energy consumption becomes minimized. Contrary to single-processor systems, in this standby-sparing system using all the available slack time is not a good policy to reduce the energy consumption (section VI). Because, prolonging tasks in the primary unit may increase the time overlap between the primary and spare units, hence frequency (voltage) of the primary unit must be selected carefully.

Let $d_i^*$ be the relative deadline of the subtask $T_i^A$, which is equal to $d_i - (WT_i - ET_i)$, consequently the available slack time is:

$$L_i = d_i - WT_i = d_i^* - ET_i$$  

(8)

The problem is that how much of the slack budget $L_i$ must be consumed for the subtask $T_i^A$ to minimize the energy consumption. Consider $\alpha_i \ (0 \leq \alpha_i \leq 1)$ be the fraction of the slack time which is used by the task $T_i$. Executing the tasks on the primary and spare units can be expressed in two cases:

#### Case 1: There is a time overlap between tasks in the primary and spare units. In other words, the given task $T_i$ finishes in the primary unit after the same copy of $T_i$ in the spare unit starts.

In this circumstance the expected energy consumption $NE_{exp}(T_i)$ of the task $T_i$ is given as:

$$NE_{exp}(T_i) = \frac{(ET_i)}{ET_i + \alpha_i L_i}^3 + \left(ET_i + \alpha_i L_i - L_i\right)$$  

(9)

The value of $\alpha_i$ which minimizes $NE_{exp}(T_i)$ (Eq. 9) is given by:

$$\alpha_i = \frac{\sqrt{2} - 1}{L_i}$$  

(10)

Consequently, the voltage/frequency scaling ratio $\rho_i$ which minimizes the energy consumption can be obtained from:

$$\rho_i = \frac{ET_i}{ET_i + \alpha_i L_i} = \left(\sqrt{2}\right)^{-1}$$  

(11)

The $\rho_i$ in Eq. 11 is only valid when $ET_i + \alpha_i L_i > L_i$, then we have:

$$ET_i + (\sqrt{2} - 1)ET_i > (d_i^* - ET_i) \equiv ET_i \frac{L_i}{d_i^*} > (\sqrt{2} + 1)^{-1}$$  

(12)

As the fraction of the slack time $\alpha_i$ which is used by the task $T_i$ is less than or equal to 1, we have:

$$\alpha_i = \frac{(\sqrt{2} - 1)ET_i}{L_i} \leq 1 \equiv ET_i \frac{L_i}{d_i^*} \leq (\sqrt{2})^{-1}$$  

(13)

When the inequality 13 does not hold, $\alpha_i$ will be obtained greater than 1. It means that the available slack time is not enough to achieve the minimum energy consumption. Nevertheless, it is beneficial to consume all the slack time and the $\rho_i$ is:

$$\rho_i = \frac{ET_i}{ET_i + L_i} = \frac{ET_i}{d_i^*} \quad \text{when } \frac{ET_i}{d_i^*} > (\sqrt{2})^{-1}$$  

(14)

According the Eq. 11 and Eq. 14, the voltage/frequency scaling ratio is obtained as:

$$\rho_i = Max\{\left(\sqrt{2}\right)^{-1}, \frac{ET_i}{d_i^*}\} \quad \text{when } \frac{ET_i}{d_i^*} > (\sqrt{2} + 1)^{-1}$$  

(15)

#### Case 2: There is no time overlap between the tasks in the primary and spare units. In other words, the given task $T_i$ finishes in the primary unit before the same copy of the task in the spare unit starts.

In this circumstance, only the primary unit is functioning and the expected energy consumption is:

$$NE_{exp}(T_i) = \frac{(ET_i)}{(ET_i + \alpha_i L_i)^2}$$  

(16)

As Eq. 16 shows, the expected energy consumption is a monotonically decreasing function of $\alpha_i$. The maximum value
of $\alpha_i$ considering that there is no overlap between the tasks in the primary and spare units will give the optimum. Therefore, $\rho_i$ must be select such that $ET_i$ is prolonged till starting the execution of the task in the spare unit which is equal to the slack time $L_i$. Hence, $\rho_i$ which minimizes the energy consumption is:

$$\rho_i = \frac{ET_i}{L_i} \quad \text{when} \quad \frac{ET_i}{d_i} \leq \left(\frac{\sqrt{2}}{1} + 1\right)^{-1} \quad (17)$$

Using Eq. 15 and Eq. 17 the optimum value of $\rho_i$ which cover the two cases can be rewritten as Eq. 18:

$$\rho_i = \begin{cases} \frac{ET_i}{L_i}, & \frac{ET_i}{d_i} \leq \left(\frac{\sqrt{2}}{1} + 1\right)^{-1} \\ \max\left\{\left(\frac{\sqrt{2}}{1} - 1\right)^{-1}, \frac{ET_i}{d_i}\right\}, & \frac{ET_i}{d_i} > \left(\frac{\sqrt{2}}{1} + 1\right)^{-1} \end{cases} \quad (18)$$

Eq. 18 is the straightforward rule which used by voltage/frequency layer to determine the processor’s voltage (frequency). Providing all the available slack time for task $T_i$ has been shown to be a good strategy as leads to be almost even-distribution of using slack time among tasks [14]. Although this statement is true for single processor systems, in the proposed standby-sparing system the story is totally different.

Considering the application which has been shown in Fig. 1, is being executed by the proposed standby-sparing system. As depicted in Fig. 4, $T_1$ is executed on the both primary and spare units simultaneously (assume there is no slack time at the beginning of the frame). $T_2$ can reclaim the slack time which is created when the task $T_1$ consumes less than its worst-case execution time and exploits all the available slack for prolonging $T_2^*$ (i.e., more probable to execute) to decrease the energy consumption. However, this leads to the execution of some part of $T_2^*$ (i.e., less probable to execute) in the primary processor and $T_2$ in the spare processor (the cross-hatched portion of the task in the spare unit). For the other tasks, in some cases, this strategy can help to fully drop a task in the spare unit (e.g., $T_5$), but in some other cases can lead to the execution of small portion (e.g., the tasks $T_4$ and $T_6$) or even a large portion (e.g., $T_3$) of the tasks. Therefore, selecting the operational voltage/frequency for the current task is an important case, so that affects the execution state of all the subsequent tasks in the both spare and primary units.

C. Slack Distribution

The voltage/frequency selector tries to minimize the energy consumption for the task $T_i$ without considering the remaining tasks $T_{i+1}$ through $T_n$ (a greedy strategy). The energy consumption of the different tasks are not independent of each other. In other words, there is a tradeoff between them, i.e., by consuming all the slack time for the task $T_i$, may lose the opportunity of decreasing the voltage level for the other tasks properly and decrease tasks overlap such that reduce the overall energy consumption effectively. While the voltage/frequency selector uses a greedy scheme, the slack distributor does not let the voltage/frequency selector to consume the entire slack budget. In other words, in contrast to the voltage/frequency selector which optimizes the energy consumption considering the current task solely, the slack distributor has a more broad vision and considers tasks (i.e., $T_i$ to $T_n$) for reducing the energy consumption globally.

The energy manager exploits the dynamic slack which arises in run-time due to the early tasks completion with respect to their worst-case. The slack distribution scheme which operates in run-time must be simple enough from the computational complexity point of view. In the following, we propose some heuristic schemes which try to distribute the slack times effectively.

**Heuristic 1:** The first heuristic approach for slack distribution is just let each task uses the slack budget in proportion to the estimated execution time. The more estimated execution time, the more slack time is allocated to the task. The Eq. 19 gives the amount of the slack budget which is passed through the slack distributor to the voltage/frequency selector in order to select $\rho_i$ and is indicated by $\psi_i$ and is given by:

$$\psi_i = \frac{ET_i}{\sum_{j=1}^{n} ET_j} \times L_i \quad (19)$$

The advantage of this approach is its simplicity but did not consider the amount of the slack times which may be reclaimed in the future from the next tasks. Additionally, abrupt changes in the subsequent tasks execution time can totally lessen the efficiency of this approach.

**Heuristic 2:** The slack budget which is available to the task $T_i$ is obtained from its previous tasks when they consume less than their worst-case execution time. Subsequent tasks have
Algorithm 1: Proposed Heuristic for Slack Distribution

Input: $WT, ET, ES, TW$
Output: Reserved_slack
1: for each sliding window ($SW$) from 1 to $TW$ do
2: Sorted_list = Sort all tasks in $SW$ based on $ET$
3: Remaining_slack = $ES_{i+TW-SW}$
4: for each task in Sorted_list do
5: $l = \text{Remove first item of the Sorted_list and return its index}$
6: $\psi_i = \sum_{i=SW}^{ET} ET \times \sum_{i=SW}^{SW} ES$
7: if $\psi_i > \text{Reserved_slack}[l]$ then
8: Allocated_slack = Min($\psi_i - \text{Reserved_slack}[l], \text{Remaining_slack}$)
9: Reserved_slack[l] += Allocated_slack
10: Remaining_slack -= Allocated_slack
end if
12: end for
14: return $\text{Reserved_slack}$

V. Optimal Solution

In this section, we describe the optimal voltage/frequency selection for decreasing the energy consumption in standby-sparing systems. This approach needs a look-ahead knowledge of the actual execution time. However, it is not applicable in real-world applications. This solution can be insightful for comparing different methods and as a margin for the maximum energy reduction.

Let $S_i^{SP}$ be the start time of the task on the spare processor. Because the spare unit uses DPM, the start time of each task can be simply achieved by Eq. 20.

$$S_i^{SP} = D - \sum_{j=i}^{n} WT_j$$

On the other hand, the start time of the tasks in the primary processor depends on the actual execution times and the amount of the slack time each of the tasks may consume for energy minimization purpose. Let $x_i$ be the amount of the slack time that each task consumes. Therefore, the start time of the task $T_{i+1}$ which is the finish time of the task $T_i$ can be written as:

$$S_{i+1}^{SP} = F_i^{Pr} = D - \sum_{j=1}^{i} (AT_j + x_j)$$

where $F_i^{Pr}$ is the finish time of the task $T_i$ while executing in the primary processor. Consequently, the normalized energy consumption $NE(T_i)$ for the task $T_i$ can be written as:

$$NE(T_i) = \left\{ \begin{array}{ll}
\frac{AT_i}{AT_i + x_i}^2 AT_i + & F_i^{Pr} - S_i^{SP} = F_i^{Pr} > S_i^{SP} \\
0 & \text{Otherwise}
\end{array} \right.$$  

Hence, the total energy consumption of an application which consists of $n$ tasks is defined by Eq. 23:

$$NE(\text{total}) = \sum_{i=1}^{n} NE_{\text{actual}}(T_i)$$

As we consider hard real-time systems with very tight deadlines, we simply assume there is no static slack in the application, and hence the first task cannot use DVS, but for the second task there may be a dynamic slack which is generated by the first task. Consequently, for task $T_n$, there may be $L_1 + L_2 + \ldots + L_{n-1}$ available slack time which is generated by the previous tasks. These constraints can be written by Eq. 24.

$$\begin{align*}
x_1 &\leq 0 \\
x_1 + x_2 &\leq L_1 \\
x_1 + x_2 + x_3 &\leq L_1 + L_2 \\
\vdots
\end{align*}$$

$$\sum_{i=1}^{n} x_i \leq \sum_{i=1}^{n-1} L_i$$
Minimizing the total energy consumption (given by Eq. 23) under the constraints given by the Inequalities 24 is an optimization problem which can be solved using geometric programming [18].

VI. SIMULATION RESULTS AND DISCUSSIONS

To evaluate the proposed method, we synthesized various synthetic tasks sets. Each set consists of 5 to 40 tasks whose execution times are uniformly distributed between 20ms and 200ms. Three different workload patterns are considered: where task execution time fluctuates between 20% and 100% of WCET [15]. The simulation results were gathered for 3000 task instances for each workload. For instance, a sample of each workload is depicted in Fig. 6 for 300 task instances. These three workload patterns are modeled as, 

\[ k_1 + k_2 \cos(x\pi/2k_3) \]

\[ k_1 \pm k_2 \sin(x\pi/k_3) \]

\[ k_1 + k_2 2^{-x/k_3} \]

(Figs. 6a through 6c respectively). Each workload pattern consists of three terms which are determined randomly. The actual execution time of each task starts at \( k_1 \) where there is no fluctuation, \( k_2 \) shows the amplitude of fluctuation which clearly \( k_1 \pm k_2 \) must be in the range of WCET and the best-case execution time, and \( k_3 \) is a parameter that affects the amount of the time which takes the task execution time reach to \( k_1 \).

The time intervals between initiating workload fluctuations are set among 10 through 50 jobs for each pattern randomly. Although the simulation used the synthetic tasks, each of these workloads is based on some typical real-time applications [17]. The first pattern simulates events with computational demand over the time intervals. Finally, the third pattern which is common in the interrupt driven systems where triggering an event leads to abrupt but short-term fluctuations.

We used the Intel XScale PXA270 [19] model as a discrete DV5-enabled processor for comparing different energy management policies in the standby-sparing system. This processor supports seven frequency ranges from 13MHz to 624MHz. Because the transition among the processor frequencies is quite fast (150μs[19]) the time and energy overhead of changing speed is considered zero. The voltage/frequency selection for the feedback based system is simply applied by mapping continues voltages obtained by the energy manager framework to discrete voltages/frequencies that does not violate timing constraints.

Five different energy management methods for the standby-sparing system were implemented. 1) Optimal method which was presented in section V and is denoted by OPT. Exploiting this method is not feasible in real world real-time systems due to the needs of actual execution time of each task and using continuous voltage/frequency. 2) Stochastic optimization method proposed by [3], which is denoted by SO. 3) Feedback based method which employs all the available slack time greedily without considering the subsequent tasks. In other words, this method does not have the slack distribution layer and is denoted by FB_TGR. 4) Feedback based method which uses the heuristic 1 for the slack distribution layer and is denoted by FB_H1. 5) Feedback based method which uses the heuristic 2 for the slack distribution layer and is denoted by FB_H2. For the three feedback based schemes PID coefficients are as follow: \( K_P=0.87, K_D=0.13, K_I=0.06 \) which are achieved trough extensive simulation (similar to [17]), and the integral window (IW) and the derivative window (DW) are determined 5 and 1, respectively.

Fig. 7 shows the simulation results for the three patterns. All the results are normalized with respect to the FB_TGR method. Each point in Fig. 7 is obtained by calculating the average value for 1000 executions of the synthetic application where task execution times are selected uniformly from [20ms, 200ms]. Simulations show that FB_H2 is the closest to the OPT method in all the cases. For patterns 1 and 2 where the velocity of fluctuations is less intensive than the pattern 3, the graphs are more similar. While the application consists of less than 20 tasks, the FB_H1 method is better than the SO, but by increasing the number of tasks the SO method
outperforms the FB_H1. For finding the voltage/frequency of the task $T_i$, the FB_H1 considers all the subsequent tasks expected execution times. However, more tasks can effectively reduce the efficiency of FB_H1 method. This effect is more noticeable with the pattern 3, when the workload fluctuations are more intensive. Moreover, the gap between the FB_H2 and the OPT methods becomes wider. This could be as a result of the feedback system which its efficiency is degraded for this workload pattern. For a greater number of tasks efficiency of all methods is far away from OPT method, but for a small number of tasks the energy consumption becomes comparable. This can be based on this fact that for an application with more numbers of tasks the slack distribution among them becomes more challenging.

VII. Conclusion

A three-layer framework for reducing the energy consumption in a standby-sparing system was introduced which exploits a feedback system. Based on the actual execution time of the tasks which fed back into the system, optimal voltage/frequency selection for an isolated task is obtained and has been shown that this approach is far away from the global optimum for multiple-task applications. The slack distributor layer in the framework tries to distribute the available slack time such that the overall energy consumption becomes minimized. A greedy scheme with two heuristic approaches is proposed for this system. Simulation results show the proposed system improves the energy consumption as compared to the previous method.

REFERENCES


