A Standby-Sparing Technique with Low Energy-Overhead for Fault-Tolerant Hard Real-Time Systems

Alireza Ejlali
Department of Computer Engineering
Sharif University of Technology
1458889694 Tehran, Iran
ejlali@sharif.edu

Bashir M. Al-Hashimi
School of Electronics and Computer Science
University of Southampton
SO17 1BJ Southampton, U.K.
bmah@ecs.soton.ac.uk

Petru Eles
Department of Computer and Information Science
Linköping University
SE–581 83 Linköping, Sweden
petel@ida.liu.se

ABSTRACT

Time redundancy (rollback-recovery) and hardware redundancy are commonly used in real-time systems to achieve fault tolerance. From an energy consumption point of view, time redundancy is generally more preferable than hardware redundancy. However, hard real-time systems often use hardware redundancy to meet high reliability requirements of safety-critical applications. In this paper we propose a hardware-redundancy technique with low energy-overhead for hard real-time systems. The proposed technique is based on standby-sparing, where the system is composed of a primary unit and a spare. Through analytical models, we have developed an online energy-management method which uses a slack reclamation scheme to reduce the energy consumption of both the primary and spare units. In this method, dynamic voltage scaling (DVS) is used for the primary unit and dynamic power management (DPM) is used for the spare. We conducted several experiments to compare the proposed system with a fault-tolerant real-time system which uses time redundancy for fault tolerance and DVS with slack reclamation for low energy consumption. The results show that for relaxed time constraints, the proposed system provides up to 24% energy saving as compared to the time-redundancy system. For tight deadlines when the time-redundancy system can tolerate no faults, the proposed system preserves its fault-tolerance but with about 32% more energy consumption.

Categories and Subject Descriptors

B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; C.3 [**Special-Purpose and Application-Based Systems**]: Real-Time and Embedded Systems

General Terms

Design, Reliability, Performance

Keywords

Hard real-time systems, Energy minimization, Reliability

1. INTRODUCTION

Many fault-tolerant real-time systems use time-redundancy techniques [1-5] where slack time is exploited to tolerate faults by performing recovery executions. However, for hard real-time systems that are used in safety-critical applications, time-redundancy techniques (e.g., roll-back recovery) have proved to be of limited utility and cannot achieve the high reliability requirements of safety-critical applications [17]. Indeed, when high reliability is the primary concern (e.g., mission- or safety-critical applications [18]), the use of hardware-redundancy techniques (also called replication [16] or hardware fault-tolerance [7]) is necessary [16]. Furthermore, the effectiveness of time-redundancy techniques is inherently dependent on the available amount of slack time so that in real-time systems with tight deadlines, the effectiveness of the time-redundancy techniques may be very low [6][7]. In this case, the use of hardware redundancy can decouple the fault tolerance from the amount of slack time and provide high reliability even when deadlines are tight. However, as hardware-redundancy techniques inherently exploit redundant hardware resources, they generally impose considerable energy overhead [7]. For example, triple modular redundancy (TMR) and duplication are two well-known hardware-redundancy techniques that can clearly increase the energy consumption by 200% and 100% respectively [18]. Therefore, careful considerations must be taken when hardware redundancy is used in hard real-time systems with limited energy budget.

In this paper we propose a hardware-redundancy technique for hard real-time systems to achieve high reliability without incurring high energy overhead. The proposed technique is based on one of the conventional hardware redundancy techniques, called standby sparing [18]. Traditionally, there are two types of standby sparing: hot and cold [18]. In the hot standby-sparing technique, the spare operates in parallel with the primary unit and is prepared to take over at any time. Clearly, the hot standby-sparing technique imposes considerable energy overhead as the spare is always operational. In the cold standby-sparing, the spare is idle until the primary unit fails and is replaced with the spare. One advantage of cold standby sparing is that the spare does not consume power until needed to replace the primary unit. However, as we will see in this paper (Section 2), in a hard real-time system, sometimes the spare must be activated even before the primary unit fails; otherwise a failure in the primary unit may result in missing a deadline. Therefore, the cold standby-sparing technique cannot be used in hard real-time systems. In the proposed standby-sparing technique, the spare is neither a cold spare nor a hot one. In fact, dynamic power management (DPM) [10] is used to reduce the energy consumption of the spare, i.e., it is kept as idle as possible, taking into account the limitations of hard real-time systems. Dynamic voltage scaling (DVS) is used to reduce the energy consumption of the primary unit; however we believe DVS is not suitable for the spare unit because: *i*) unlike the primary unit, the spare unit is not always operational and hence the energy consumption of the spare unit is not as prominent as that of the primary unit, *ii*) it has been observed that DVS considerably increases the system vulnerability to external disturbances [6], therefore we do not use DVS to avoid degrading the reliability of the spare unit so that it can always serve as a reliable backup for the primary unit. In the proposed system, when DVS is used to reduce the energy of the primary unit, it may increase the operation time of the spare unit (Section 2), therefore the supply voltage of the primary unit should be determined by considering the energy consumption of both the primary and spare units. For the proposed standby-sparing system, we have developed an online energy-management method which uses a slack reclamation scheme (i.e., can exploit dynamic slacks) to reduce the energy consumption of both the primary and spare units.

Some research works, e.g., [2][3][8], have addressed both fault tolerance and low energy-consumption in fault-tolerant real-time systems that are based on time-redundancy. However, these works have focused on time redundancy and have not considered hardware redundancy. [15] has proposed a technique to exploit voltage scaling to reduce the energy overhead of TMR when it is used for real-time systems. This technique can reduce the energy overhead of a TMR system to a level comparable to that of a duplicated system. However, this work has not considered any slack reclamation scheme. [6] has proposed to use a combination of information redundancy and time redundancy to address the resource conflict between time-redundancy techniques and DVS on slack. However, this work has not considered hardware-redundancy techniques and does not provide any energy-management method for fault-tolerant real-time systems.

The main contributions of this paper are:

- We will provide a standby-sparing technique for hard real-time systems where DVS and DPM are employed to reduce the energy consumption of the primary and spare units respectively.
- We will show that when we want to reduce the energy consumption of the proposed standby-sparing system by exploiting dynamic slacks, we face with a problem that involves decisions under stochastic uncertainties. We will provide an online energy-management method to solve this problem. This online method determines the supply voltages of the primary unit at runtime. Furthermore, it decides when the spare unit should be activated.
- We will use an analytical approach to develop the energy-management method and to show that the proposed standby-sparing system and its associated energy-management method operate effectively.

To evaluate the proposed standby-sparing technique, we have conducted several experiments using MPARM tool [21][23], MiBench benchmarks [22], and several synthetic schedules. The experimental results show that the energy and execution time overheads of our proposed online energy management method are negligible (less than 0.15%). We also compared the proposed system with a fault-tolerant real-time system which uses time redundancy instead of hardware redundancy. To provide a fair comparison, it is assumed that the time-redundancy system also uses DVS with slack reclamation for low energy consumption. The results show that for relaxed time constraints, the proposed system consumes less energy than the time-redundancy system. For tight deadlines when the time-redundancy system is not fault tolerant, the proposed system still preserves its fault-tolerance.

The rest of the paper is organized as follows. Section 2 describes the proposed standby-sparing system. In Section 3, we have developed analytical energy models for the proposed standby-sparing system. In Section 4, we have shown that the problem of minimizing the energy consumption of the proposed standby-sparing system involves decisions under stochastic uncertainties and provided a solution for this problem. In Section 5 simulation results are presented and discussed. Finally, Section 6 concludes the paper.

2. PROPOSED SYSTEM

In this paper, we consider a standby-sparing system which is composed of two identical processors. One of them is called the primary unit and operates as the main processor, while the other one is called the spare unit and replaces the primary unit when it fails. Clearly, a standby-sparing system requires an error detection mechanism to determine if a task finishes successfully or not. In the context of fault-tolerant real-time systems, the error detection mechanisms is usually assumed to be part of the software architecture and the error detection overhead is considered as part of the task execution time [1][5]. Similarly, in this paper, we assume that an error detection mechanism is part of the software architecture. It should be noted that the use of fault detection mechanisms is not limited to standby-sparing and they should also be used for fault-tolerant real-time systems that use rollback-recovery (time-redundancy) [8].

To reduce the energy consumption of the standby-sparing system, DVS [10][11] is used for the primary unit and DPM [10] is used for the spare. From a reliability point of view, DPM is more suitable for the spare as compared to DVS, because: 1) DVS would have a negative impact on the reliability of the spare [6], while we want the spare to be a reliable backup for the primary unit, 2) when the spare is idle and does not operate, it is not susceptible to transient faults [18], hence keeping the spare idle for longer periods results in more reliability for the spare. For both the DVS and DPM techniques, we use a slack reclamation scheme, i.e., dynamically created slacks are exploited to achieve energy saving. Dynamic slacks result at runtime when tasks consume less than their worst-case execution time [2]. The use of dynamic slacks helps to achieve more energy saving as compared to the techniques that only use static slack time which is the difference between the deadline and the worst-case execution time [2][19].

It should be noted that the proposed standby-sparing system does not use its slack time for fault tolerance. The fault tolerance is achieved by using the spare and the slack time is only used for reducing the energy consumption. Therefore, unlike time-redundancy techniques, the proposed system preserves its fault-tolerance even when the available slack is small.

The proposed standby-sparing system does not need any dedicated scheduler. Indeed, it is assumed that a static schedule already exists for a single processor system which has no fault-tolerance or energy-management mechanism and the proposed standby-sparing system uses this same schedule to run the given application. Since such simple static schedules (without any fault-tolerance or energy-saving mechanism) can be effectively synthesized using existing techniques [9], this paper does not address scheduling problems. In this paper, we consider hard real-time systems, and it is assumed that time constraints are imposed by specifying global deadlines for groups of consecutive tasks. For example, Fig. 1a shows an example static schedule where two global deadlines are specified for two groups of tasks. In this schedule, the deadline D1 is for the group T1, T2, T3, T4, T5 so that these tasks should be executed consecutively before the deadline D1. Also, the deadline D2 is for the group T6, T7, T8 and they should be executed consecutively before the deadline D2. It can be seen from Fig. 1a that there is no static slack in the schedule of Fig. 1a. Throughout this section, without loss of generality, we consider only schedules that do not have any static slack time. This is because, as we will see in Section 4, our proposed energy management method exploits dynamically created slacks to reduce the energy consumption, rather than using static slacks. Even if a static slack be available in the schedule, it can be exploited by our proposed technique as if it is a dynamic slack which is created at runtime (Section 5). It should be noted that the schedule of Fig. 1a may have been synthesized from various task graphs. For example Figs. 1b and 1c show two possible task graphs that the schedule of Fig. 1a may be synthesized from. However, as mentioned in this section, our proposed method does not involve scheduling and it is assumed that a static schedule like what is shown in Fig. 1a is already available. When the proposed standby-sparing system is executing such a schedule, it does not change the temporal order of the tasks to avoid violating dependencies and precedence constraints that may exist in the original task graph.

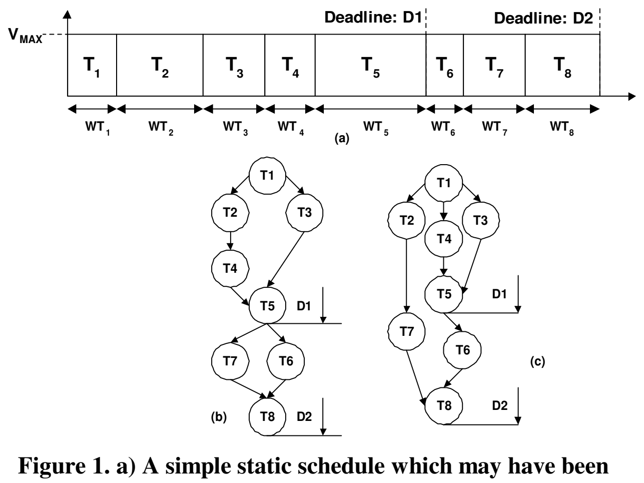

Figure 1. a) A simple static schedule which may have been synthesized from either of the task graphs (b) or (c)

In the following, we describe how the proposed standby-sparing system operates. Suppose that a static schedule like what is shown in Fig. 1a exists for a single processor system operating at the maximum possible supply voltage V_{MAX}. Consider a group of n tasks T_0 through T_{n-1} with a deadline D. When tasks are executed at the supply voltage V_{MAX}, each task T_i has a worst-case execution time WT_i, and an actual execution time AT_i. Each task T_i is executed on the primary unit at a supply voltage V_i, which may be less than the maximum supply voltage V_{MAX}. For each task T_i, we define the normalized supply voltage ρ_i as follows:

$$\rho_i = \frac{V_i}{V_{MAX}} \tag{1}$$

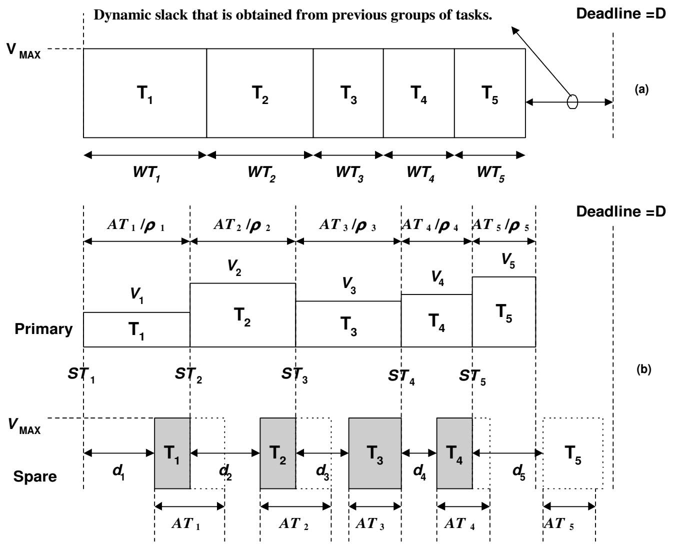

Figure 2. a) A group of tasks in a static schedule, b) the same tasks running on the proposed standby-sparing system

When a reduced supply voltage is used for a task T_i, the worst-case execution time is prolonged from WT_i to WT_i/ρ_i and the actual execution time is prolonged from AT_i to AT_i/ρ_i. As an example, Fig. 2 shows how a group of 5 tasks with a global deadline D (Fig. 2a) is executed on the proposed standby-sparing system. It should be noted that the slack in Fig. 2a is not a static slack and it is a dynamic slack, which is created because the previous group of tasks has finished sooner than its worst-case execution time. While in the primary unit each task T_i is executed at a normalized supply voltage ρ_i, in the spare unit the backup copy of each task T_i is executed at the maximum supply voltage, but with a delay d_i. During the delay time d_i the spare unit is in idle mode to conserve energy. Also, whenever a task T_i which is being executed on the primary unit finishes successfully, the backup copy of this task, which is being executed on the spare unit, is dropped as it is no longer required. This implies that an increase in the delay d_i results in more energy saving for the backup copy of the task T_i because as the delay d_i increases, the fraction of the backup task T_i which is executed on the spare unit becomes smaller (Fig. 2b). However, d_i cannot be increased arbitrarily as it may result in missing the deadline if a fault occurs in the primary unit. The proper value for the delay d_i can be calculated as follows. Let ST_i be the time at which the task T_{i-1} finishes running on the primary unit and the task T_i starts running on this unit. We have:

$$ST_i = \sum_{j=1}^{i-1} \frac{AT_j}{\rho_j} \tag{2}$$

Suppose that during the execution of the task T$_i$ on the primary unit, a fault occurs. In this case, the backup copy of the task T$_i$ which is being executed on the spare will not be dropped and its execution will be continued. To meet the deadline, there is a need to have enough time to finish not only the backup task T$_i$ (with the worst-case execution time WT_i), but also all the subsequent tasks (i.e., the tasks T$_{i+1}$ through T$_n$). If we execute all these tasks (T$_i$ through T$_n$) at the maximum supply voltage V_{MAX}, The worst-case time it takes to execute all these tasks will be:

$$WRT_i = \sum_{j=i}^{n} WT_j \tag{3}$$

Therefore, the maximum possible value for the delay d_i that allows the system to meet the deadline is:

$$d_i = D - ST_i - WRT_i = D - (\sum_{j=1}^{i-1} \frac{AT_j}{\rho_j}) - (\sum_{j=i}^{n} WT_j) \tag{4}$$

The execution of each backup task T$_i$ on the spare should be delayed by the time d_i (given by Eq. 4) to achieve energy saving for the spare without missing the deadline.

Although d_i is a parameter which has been defined for the backup tasks on the spare, it is noteworthy that the d_i value given by Eq. 4 is also equal to the dynamic slack which is available to the original task T$_i$ on the primary unit. This is because ST_i (Eq. 2) is indeed the time that has been elapsed to execute the tasks T$_1$ through T$_{i-1}$ on the primary unit, and WRT_i is indeed the worst-case time which is required to execute the remaining tasks T$_i$ through T$_n$ at the maximum supply voltage on the primary unit and hence D-ST_i-WRT_i (i.e., the d_i value given by Eq. 4) is also equal to the dynamic slack which is available to the original task T$_i$ on the primary unit. Hence, we can write:

$$DS_i = D - ST_i - WRT_i = D - (\sum_{j=1}^{i-1} \frac{AT_j}{\rho_j}) - (\sum_{j=i}^{n} WT_j) \tag{5}$$

where DS_i is the dynamic slack which is available to the task T_i.

The problem which is considered in the rest of this paper is how, for each task T$_i$, the two parameters ρ_i and d_i should be determined online so that the energy consumption becomes minimized while guaranteeing that the deadline will not be missed. It should be noted that although ρ_i is the normalized voltage at which the task T$_i$ is executed on the primary unit, we will see in Section 3 that the energy consumption of the spare also depends on the parameter ρ_i.

3. ENERGY CONSUMPTION MODEL

In this section we develop analytical models for the energy consumption of the standby-sparing system.

Primary Unit: Considering the use of the DVS technique, the energy consumption of each task T$_i$ on the primary unit is [6][10]:

$$E_{PR}(T_i) = C_{eff} V_i^2 f_i \cdot (\frac{AT_i}{\rho_i}) \tag{6}$$

where C_{eff} is the average switched capacitance for the primary unit, V_i and f_i are respectively the supply voltage and the operational frequency during the execution of the task T$_i$, and (AT_i/ρ_i) is the time it takes to execute the task T$_i$ (Section 2). For the DVS technique, it can be assumed that there is an almost linear relationship between V_i and f_i [11], therefore using Eq. 1 we can write $\rho_i = V_i/V_{MAX} = f_i/f_{MAX}$, where f_{MAX} is the operation frequency associated to the supply voltage V_{MAX}. Therefore, the energy E_{PR}(T$_i$) of Eq. 6 can also be written as:

$$E_{PR}(T_i) = C_{eff} V_{MAX}^2 f_{MAX} \rho_i^2 AT_i \tag{7}$$

Spare Unit: To calculate the energy consumption of the backup task T$_i$ on the spare, we consider three possible cases based on the times at which the original and backup copies of a task T$_i$ start and finish. As observed in Section 2, when a task T$_i$ is executed on the proposed system (Fig. 2), the original copy of T$_i$ starts running on the primary unit at the time ST_i and finishes at the time ST_i+AT_i/ρ_i. On the other hand, the backup copy of T$_i$ on the spare starts at the time ST_i+d_i and finishes at the time ST_i+d_i+AT_i provided that the task is not dropped.

Case 1: The original copy of T$_i$ finishes before the backup copy of T$_i$ starts

In this case, since "The finish time of the original copy" ≤ "The start time of the backup copy", we have:

$$ST_i + \frac{AT_i}{\rho_i} \le ST_i + d_i \quad \equiv \quad \frac{AT_i}{\rho_i} \le d_i \tag{8}$$

In this case, the backup copy on the spare will be completely dropped if the original copy finishes successfully. This is because before the backup copy can even start, the original copy has finished successfully and hence the backup copy is not required. Such a scenario has occurred for the task T_5 in Fig. 2 where $AT_5/\rho_5 \le d_5$. For this case, the energy consumption of the spare is:

$$E_{SPR}(T_i) = 0 \quad when \quad \frac{AT_i}{\rho_i} \le d_i \tag{9}$$

Case 2: The original copy of T$_i$ finishes after the backup copy of T$_i$ starts, however the original copy finishes before the backup copy finishes and hence the rest of the backup copy is dropped.

In this case, since "The start time of the backup copy" < "The finish time of the original copy", we have:

$$ST_i + d_i < ST_i + \frac{AT_i}{\rho_i} \quad \equiv \quad d_i < \frac{AT_i}{\rho_i} \tag{10}$$

Also, since "The finish time of the original copy" < "The finish time of the backup copy", we have:

$$ST_i + \frac{AT_i}{\rho_i} < ST_i + d_i + AT_i \quad \equiv \quad \frac{AT_i}{\rho_i} - AT_i < d_i \tag{11}$$

Inequalities 10 and 11 can be written together as:

$$\frac{AT_i}{\rho_i} - AT_i < d_i < \frac{AT_i}{\rho_i} \tag{12}$$

In this case, unlike Case 1, as the backup copy starts before the original copy finishes, a part from the beginning of the backup copy is executed (the shaded areas in Fig. 2b). However, as the original copy finishes before the backup copy finishes, the backup copy is not executed completely and is dropped once the original copy finishes so that the backup copy is executed only for a duration AT_i/ρ_i-d_i. Such a scenario has occurred for the tasks T_1, T_2, and T_4 in Fig. 2 where $AT_1/\rho_1-AT_1<d_1<AT_1/\rho_1$, $AT_2/\rho_2-AT_2<d_2<AT_2/\rho_2$, and $AT_4/\rho_4-AT_4<d_4<AT_4/\rho_4$. For this case, the energy consumption of the spare is:

$$E_{SPR}(T_i)=C_{eff}V_{MAX}^2 f_{MAX}\left(\frac{AT_i}{\rho_i}-d_i\right) \qquad \text{when} \quad \frac{AT_i}{\rho_i}-AT_i\le d_i<\frac{AT_i}{\rho_i} \tag{13}$$

Case 3: Both the original and backup copies of T_i finish at the same time.

In this case, since "The finish time of the original copy" = "The finish time of the backup copy", we have:

$$ST_i+\frac{AT_i}{\rho_i}=ST_i+d_i+AT_i \qquad \equiv \qquad \frac{AT_i}{\rho_i}-AT_i=d_i \tag{14}$$

In this case, since the original copy does not finish before the backup copy finishes, the backup copy is not dropped and is executed completely. Such a scenario has occurred for the task T_3 in Fig. 2 where $AT_3/\rho_3-AT_3=d_3$. For this case, the energy consumption of the spare is:

$$E_{SPR}(T_i)=C_{eff}V_{MAX}^2 f_{MAX}AT_i \qquad \text{when} \qquad \frac{AT_i}{\rho_i}-AT_i=d_i \tag{15}$$

It can be seen from the above discussion that in Cases 1, 2, and 3 we have considered the d_i values that are respectively in the ranges $AT_i/\rho_i\le d_i$, $AT_i/\rho_i-AT_i<d_i<AT_i/\rho_i$, and $AT_i/\rho_i-AT_i\le d_i$. Therefore, in the above three cases, we have in fact considered the d_i values in the range $AT_i/\rho_i-AT_i\le d_i$. On the other hand, we have proved in Theorem 2 in Appendix A that d_i is not less than AT_i/ρ_i-AT_i, therefore all the possible values of d_i have been considered in the above three cases. Considering all the three possible cases, the energy consumption of each backup task T_i on the spare is:

$$E_{SPR}(T_i)=\begin{cases} 0 & \frac{AT_i}{\rho_i}\le d_i \\ C_{eff}V_{MAX}^2 f_{MAX}\left(\frac{AT_i}{\rho_i}-d_i\right) & \frac{AT_i}{\rho_i}-AT_i\le d_i<\frac{AT_i}{\rho_i} \end{cases} \tag{16}$$

Total Energy Consumption and Normalized Energy: The energy which is consumed by the whole system (both the primary and spare units) to execute a task T_i is:

$$E(T_i)=E_{PR}(T_i)+E_{SPR}(T_i) \tag{17}$$

where $E_{PR}(T_i)$ is given by Eq. 7 and $E_{SPR}(T_i)$ is given by Eq. 16. Therefore, the total energy which is consumed by the system to execute all the tasks T_i, $1\le i\le n$, is:

$$E_{TOT}=\sum_{i=1}^{n}E(T_i) \tag{18}$$

It can be seen from Eqs. 7 and 16 that the energies of Eqs. 7, 16, 17, and 18 have a constant factor $C_{eff}V_{MAX}^2 f_{MAX}$. For the sake of simplicity, we remove this constant factor from the energies of Eqs. 7, 16, 17, and 18 and call the reset 'normalized energy'. The normalized energies are defined by the following equations (Eqs. 19 through 22):

$$NE_{PR}(T_i)\overset{\Delta}{=}\frac{E_{PR}(T_i)}{C_{eff}V_{MAX}^2 f_{MAX}}=\rho_i^2 AT_i \tag{19}$$

$$NE_{SPR}(T_i)\overset{\Delta}{=}\frac{E_{SPR}(T_i)}{C_{eff}V_{MAX}^2 f_{MAX}}=\begin{cases} 0 & \frac{AT_i}{\rho_i}\le d_i \\ \left(\frac{AT_i}{\rho_i}-d_i\right) & \frac{AT_i}{\rho_i}-AT_i\le d_i<\frac{AT_i}{\rho_i} \end{cases} \tag{20}$$

$$NE(T_i)\overset{\Delta}{=}\frac{E(T_i)}{C_{eff}V_{MAX}^2 f_{MAX}}=NE_{PR}(T_i)+NE_{SEC}(T_i) \tag{21}$$

$$NE_{TOT}\overset{\Delta}{=}\frac{E_{TOT}}{C_{eff}V_{MAX}^2 f_{MAX}}=\sum_{i=1}^{n}NE(T_i) \tag{22}$$

where $NE_{PR}(T_i)$ is the normalized energy consumption of the primary unit for the execution of the task T_i, $NE_{SPR}(T_i)$ is the normalized energy consumption of the spare for the execution of the task T_i, $NE(T_i)$ is the normalized energy consumption of the whole system (both the primary and spare units) for the execution of the task T_i, and NE_{TOT} is the normalized energy consumption of the whole system for the execution of all the tasks T_i , $1\le i\le n$. In the rest of this paper, we focus on how to minimize the normalized energy NE_{TOT} given by Eq. 22, which is consumed by the proposed standby-sparing system to execute all the tasks.

4. ENERGY MANAGEMENT METHOD

In this section we aim at providing a method to determine the parameters ρ_i and d_i to reduce the normalized energy NE_{TOT} given by Eq 22. As mentioned in Sections 1 and 2, in the proposed energy management method, we want to exploit dynamic slacks to save energy. Therefore, since dynamic slacks result at runtime, the energy-management method should be online and applied at runtime. One way to reduce the energy NE_{TOT} is to reduce the energy which is consumed by each individual task, i.e., $NE(T_i)$, $1\le i\le n$, separately. However, the energy consumptions of different tasks ($NE(T_i)$ for different i values) are not independent from each other and there is a tradeoff between them. For example, if the task T_i does not exploit all the available dynamic slack to reduce the energy $NE(T_i)$, the remaining slack will be available to the task T_{i+1} to reduce the energy $NE(T_{i+1})$. To deal with this issue, in the proposed method, we adopt a greedy strategy where for each task T_i, the parameters ρ_i and d_i are determined at the start of the task T_i with the aim of reducing the energy $NE(T_i)$, without considering the energy consumption of the remaining tasks T_{i+1} through T_n. One important advantage of this greedy strategy is that it helps to distribute the available slack time evenly among the tasks of a schedule. It has been shown that the even distribution of the available slack time among the tasks results in more energy saving as compared to an uneven distribution [10]. In fact, the available dynamic slack is liable to be distributed unevenly among the tasks. This is because the dynamic slack which is available to a task T_i is obtained only from its previous tasks (i.e., the tasks T_1 through T_{i-1}) when they consume less than their worst-case execution time. A task T_i can never exploit the dynamic slack which is obtained from its subsequent tasks (i.e., the tasks T_{i+1} through T_n). Therefore, those tasks that come later in a schedule have more chance to gain larger dynamic slacks as compared to the tasks that come earlier in the schedule. In the greedy strategy, for each task T_i, we exploit the available slack without any attempt to leave some slack for the subsequent tasks T_{i+1} through T_n. Therefore, considering that the tasks T_{i+1} through T_n have inherently more chance to gain larger dynamic slacks as compared to the task T_i, the greedy strategy helps to distribute the available slack time evenly. Due to the greedy strategy, for each task T_i, we only focus on reducing the energy $NE(T_i)$ without considering the energy consumption of the other tasks. The energy $NE(T_i)$ has already been given by Eq. 21, but to make it easy to follow the discussion, we have expanded it as follows:

$$NE(T_i) = NE_{PR}(T_i) + NE_{SPR}(T_i) = \underbrace{\rho_i^2 AT_i}_{\text{Energy of Primary Unit}} + \underbrace{\begin{cases} 0 & \dfrac{AT_i}{\rho_i} \le d_i \\ (\dfrac{AT_i}{\rho_i} - d_i) & \dfrac{AT_i}{\rho_i} - AT_i \le d_i < \dfrac{AT_i}{\rho_i} \end{cases}}_{\text{Energy of Spare Unit}} \qquad (23)$$

As it can be seen from Eq. 23, the energy consumption of the primary unit does not depend on the parameter d_i (i.e., the time by which the backup task T_i is delayed) and only the energy consumption of the spare unit depends on this parameter. Therefore, when we want to determine the parameter d_i, we only need to focus on the energy consumption of the spare. As mentioned in Section 2, the proper value for the delay d_i to achieve maximum energy saving for the spare is equal to the available dynamic slack DS_i. Therefore, in the proposed online energy management method, at the start of each task T_i, the parameter d_i should be simply set to the available dynamic slack (Eqs. 4 and 5):

$$d_i = DS_i = (D - WRT_i) - ST_i \qquad (24)$$

It should be noted that $(D\text{-}WRT_i)$ is not needed to be calculated online and can be easily calculated offline for each task and stored to be used at runtime because both D and WRT_i, which is given by Eq. 3, are known at design time. Also, ST_i is the start time of the task T_i and hence is the current time that the internal clock of the system shows at the time the task T_i starts running on the primary unit. While the parameter d_i can be simply determined at runtime using Eq. 24, the online estimation of the parameter ρ_i is not trivial. To investigate how the parameter ρ_i should be estimated, we rewrite Eq. 23 as follows:

$$NE(T_i) = \underbrace{\rho_i^2 AT_i}_{\text{Energy of Primary Unit}} + \underbrace{\begin{cases} 0 & \dfrac{AT_i}{d_i} \le \rho_i \\ (\dfrac{AT_i}{\rho_i} - d_i) & \dfrac{AT_i}{d_i + AT_i} \le \rho_i < \dfrac{AT_i}{d_i} \end{cases}}_{\text{Energy of Spare Unit}} \qquad (25)$$

```
Inputs:
- $x_i$[j],$y_i$[j],$w_i$[j], and $z_i$[j], where 1≤j≤K and
K is the number of possible supply voltages.
- $WT_i$, ($D$-$WRT_i$), $ST_i$

Outputs:
- $\rho_i$ and $d_i$

//$\rho$[j] (1≤j≤K) is the array which holds the
//possible supply voltages in ascending order.
//E is the expected value of normalized energy
//$x_i$[j],$y_i$[j],$w_i$[j], and $z_i$[j] have been
//calculated offline for 1≤j≤K, using Eq. 28.
//($D$-$WRT_i$) has been also calculated offline.
//$ST_i$ is the current time and is received from
//the system internal clock.

1:  $d_i$:=($D$-$WRT_i$)-$ST_i$; //Eq. 24
2:  di2:= $d_i$*$d_i$;
3:  $\rho_{min}$:=$WT_i$/($WT_i$+$d_i$);
4:  m:=1;
5:  while($\rho$[m]<$\rho_{min}$) m:=m+1;
6:  $\rho_i$:=$\rho$[m];
7:  if($w_i$[m]<=$d_i$) E:=$x_i$[m]
    else E:=$y_i$[m]+$z_i$[m]*di2; //Eq. 29
8:  for j:=m+1 to K
9:  {
10:   if($w_i$[j]<=$d_i$) TMP:=$x_i$[j];
      else TMP:=$y_i$[j]+$z_i$[j]*di2; //Eq. 29
11:   if(TMP<E) {E:=TMP; $\rho_i$:=$\rho$[j];}
12: }
```

Figure 3. The pseudo code of the proposed online energy management method

It should be noted that Eq. 25 is the same as Eq. 23, and the only difference is that the condition of each piece in the piecewise function has been rephrased to make the equation more proper for investigating the parameter ρ_i. We have proved in Theorem 1 in Appendix A that ρ_i should not be less than $WT_i/(d_i+WT_i)$ to avoid missing deadlines. Furthermore, it can easily be shown that $WT_i/(d_i+WT_i) \ge AT_i/(d_i+AT_i)$. Therefore, Eq. 25 can also be written as:

$$NE(T_i) = \underbrace{\rho_i^2 AT_i}_{\text{Energy of Primary Unit}} + \underbrace{\begin{cases} 0 & \dfrac{AT_i}{d_i} \le \rho_i \\ (\dfrac{AT_i}{\rho_i} - d_i) & \dfrac{WT_i}{d_i + WT_i} \le \rho_i < \dfrac{AT_i}{d_i} \end{cases}}_{\text{Energy of Spare Unit}} \qquad (26)$$

We have proved in Theorem 3 in Appendix A that the optimum value of ρ_i which minimizes the energy $NE(T_i)$ depends on the parameter AT_i, however AT_i is random and not known at the start of the task T_i. Therefore, the problem of minimizing the energy $NE(T_i)$ by adjusting the parameter ρ_i is in fact an optimization problem under stochastic uncertainties. One effective way to minimize such a function is to minimize the expected value of the function rather than the function itself [12]. Assuming that AT_i is

Table 1. The energy consumption and execution time of the benchmark tasks

Benchmark	Voltage, Frequency	Execution time (ms)	Energy Consumption (μJ)
qsort	1V,200MHz	453.93	14065.11
	0.58V,100MHz	881.56	11037.71
basicmath	1V,200MHz	707.61	20852.51
	0.58V,100MHz	1310.29	16379.83
bitcount	1V,200MHz	497.21	15883.70
	0.58V,100MHz	1009.17	12665.62
susan (smoothing)	1V,200MHz	258.68	8047.77
	0.58V,100MHz	503.35	6252.58
susan (edges)	1V,200MHz	18.89	588.03
	0.58V,100MHz	37.32	456.85
susan (corners)	1V,200MHz	10.96	337.56
	0.58V,100MHz	21.70	265.72

Energy manager task (Fig. 3)	1V,200MHz	0.0137	0.4190
	0.58V,100MHz	0.0267	0.3270

uniformly distributed, it can be shown that the expected value of *NE*(T$_i$) is:

$$E[NE(T_i)] = \begin{cases} \dfrac{WT_i^2}{2}\rho_i^2 & \dfrac{WT_i}{\rho_i} \le d_i \\ \dfrac{WT_i^2}{2}(\rho_i^2 + \dfrac{1}{\rho_i}) + \dfrac{d_i^2}{2}\rho_i & d_i < \dfrac{WT_i}{\rho_i} \end{cases} \quad (27)$$

In DVS-enabled processors the supply voltage can only take a value from a finite set of possible voltage values [20]. In our proposed online energy management method, at the start of each task, Eq. 27 is calculated for all the possible values of ρ_i, and then the parameter ρ_i is set to the voltage value which gives the least value for $E[NE(T_i)]$. It should be noted that most of the calculations required by Eq. 27 can be performed offline for each task and stored to be used at runtime. For this purpose, let x_i, y_i, w_i, and z_i be defined as follows:

$$x_i \overset{\Delta}{=} \frac{WT_i^2}{2}\rho_i^2,\ \ y_i \overset{\Delta}{=} \frac{WT_i^2}{2}(\rho_i^2 + \frac{1}{\rho_i}), \qquad w_i \overset{\Delta}{=} \frac{WT_i}{\rho_i},\ \ z_i \overset{\Delta}{=} \frac{\rho_i}{2} \quad (28)$$

The parameters x_i, y_i, w_i, and z_i can be calculated offline for each task as the parameter WT_i and the possible values of ρ_i are known at design time. Using these four parameters (x_i, y_i, w_i, and z_i), Eq. 27 can be rewritten as:

$$E[NE(T_i)] = \begin{cases} x_i & w_i \le d_i \\ y_i + z_i d_i^2 & d_i < w_i \end{cases} \quad (29)$$

Table 2. The energy consumption of the standby-sparing and time-redundancy systems*

Relaxed time constraints: Static Slack= the biggest WT (worst case execution time) in the schedule				
Distribution of the actual execution time	# of tasks in the schedule	Energy of Time-Redundancy system (J)	Energy of Standby-Sparing system (J)	Energy Ratio$^{\Psi}$
Uniform from 0 to WT	5	36.32	46.23	1.27
	10	60.94	59.21	0.97
	15	105.90	66.81	0.63
Exponential λ=3/WT	5	18.05	10.39	0.58
	10	38.72	28.17	0.73
	15	69.64	38.19	0.55
Normal μ=WT/2 σ=WT/4	5	38.28	34.73	0.91
	10	69.27	45.23	0.65
	15	105.31	59.17	0.56

Tight time constraints: Static Slack= 0				
Distribution of the actual execution time	# of tasks in the schedule	Energy of Time-Redundancy system (J)	Energy of Standby-Sparing system (J)	Energy Ratio$^{\Psi}$
Uniform from 0 to WT	5	36.32	74.88	2.06
	10	60.94	93.34	1.53
	15	105.90	131.52	1.24
Exponential λ=3/WT	5	18.05	20.87	1.16
	10	38.72	43.65	1.13
	15	69.64	56.31	0.81
Normal μ=WT/2 σ=WT/4	5	38.28	67.02	1.75
	10	69.27	82.63	1.19
	15	105.31	110.20	1.05

* For all the three distributions, it was assumed that the task worst-case execution times (i.e., WT) are uniformly distributed from 20ms to 1500ms.
Ψ Energy Ratio = Energy of the proposed system / Energy of the time-redundancy system

Clearly, the online calculation of Eq. 29 imposes less overhead as compared to Eq. 27. Fig. 3 shows the pseudo code of the proposed online energy management method. This code is executed at the start of each task T$_i$, and determines the parameters d_i and ρ_i. In this code, we first determine the parameter d_i (line 1) using Eq. 24. Then we start from the minimum possible value of ρ_i (calculated in line 3) and for each possible supply voltage we use Eq. 29 to calculate the expected normalized energy (lines 7 and 10). Finally, we set the parameter ρ_i to the voltage which gives the least value for the expected normalized energy (line 11). It should be noted that although Eq. 27 is derived with the assumption that AT_i is uniformly distributed, we will show in Section 5, through simulation experiments, that this method is quite effective to reduce the energy consumption of the proposed standby-sparing system even when AT_i has other distributions (e.g., normal and exponential distributions).

5. SIMULATION RESULTS

To evaluate the proposed method, we have conducted several experiments using MiBench benchmarks (Auto./Industrial set) [22], and numerous synthetic schedules. MPARM [21] (cycleaccurate simulator for ARM7TDMI processor proposed in [23]) were used to obtain the power consumption and execution times reported in the paper. The first set of experiments was conducted in order to investigate the energy and execution time overhead of the proposed online energy management method. In the experiments, the processor could have five different supply voltages: 1V(200MHz), 0.86V(167MHz), 0.76V(143MHz), 0.69V(125MHz), 0.58V(100MHz). To execute the benchmarks, we used the RTEMS embedded operating system [24]. Table 1 shows the energy consumption and execution time of the benchmark tasks when executed at the supply voltages 1V, and 0.58V (the maximum and minimum values of the supply voltage). It can be seen from Table 1 that, as compared to the MiBench benchmarks, the energy and execution time overhead of the proposed online energy management method is always less than 0.15%, which is quite negligible.

To evaluate the effectiveness of the proposed method, we conducted another sets of experiments where we compared our proposed standby-sparing system with a time-redundancy system which use rollback-recovery (re-execution) to tolerate faults. It was assumed that the time-redundancy system exploits dynamic slack through DVS to reduce the energy consumption. It was also assumed that the time-redundancy system does not use its dynamic slacks to tolerate faults and only uses its static slack for fault tolerance (re-execution). This assumption is reasonable, because unlike the static slack, the available amount of dynamic slack is not known at design time and hence dynamic slacks cannot be used in a fault-tolerant static schedule. To compare the two systems, 99 static schedules similar to the schedule of Fig. 1a were generated randomly and used in the experiments. Out of these 99 random schedules, one third were generated with 5 tasks and one deadline, one third with 10 tasks and 2 deadlines, and one third with 15 tasks and three deadlines. To generate random schedules, the worst-case execution times of the tasks were generated randomly using uniform distribution. It was assumed that the worst-case execution times of the tasks could be any value from 20ms to 1500ms. For example, when we wanted to generate a static schedule with 4 tasks, we obtained the random numbers: WT_1=299ms, WT_2=50ms, WT_3=328ms, and WT_4=142ms. These numbers form the static schedule which is shown in Fig. 4.

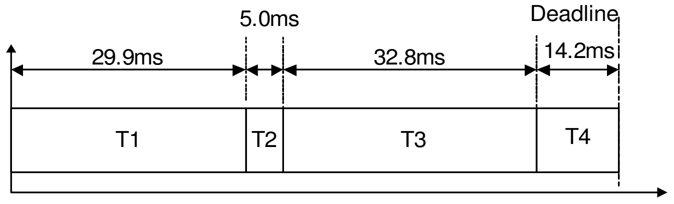

Figure 4. An example static schedule which has been generated randomly

With respect to the static slack time we considered two cases: 1) relaxed time constraints: when the static slack is equal to (or bigger than) the biggest worst case execution time in the schedule, in this case the time-redundancy system will have enough time to re-execute any of the tasks in the schedule if a fault occurs, 2) tight time constraints: when the static slack is so small that the time-redundancy system cannot perform any re-execution.

For generating random static schedules we used uniform distributions for worst-case execution times WT_i, as we wanted all schedules to be equally probable to be considered. However, for a specific schedule, the actual execution times of the tasks AT_i may have different probability distributions based on the system application [13]. In the context of real-time systems, some research works have considered the uniform, normal, or exponential distributions for the actual execution times of the tasks [13][14]. Similarly, in our experiments, we considered these three distributions for the actual execution times AT_i. It should be noted that in the experiments the same static schedules were used for all the three distributions. Indeed at first we randomly generated 99 static schedules and then we used these static schedules with various distributions for the actual execution times AT_i. In all the experiments, the tasks in the synthetic schedules were selected from the MiBench benchmarks; however as we wanted to evaluate the impact of AT_i distribution, each task T_i was executed only for a duration of AT_i (AT_i / ρ_i when voltage scaling is used (Section 2)) which was generated randomly by one of the three distributions. Table 2 shows the energy consumption of the synthetic schedules when executed on the proposed standby-sparing and time-redundancy systems.

The following three interesting observations can be made from Table 2:

- The results show that for tight deadlines, the proposed standby-sparing system consumes in average 32% more energy than the time-redundancy system. However, in this case, the time-redundancy system has not enough time for fault tolerance (re-execution) and hence can tolerate no faults, while the proposed standby-sparing system is still fault tolerant (Section 2).
- For relaxed time constraints, the proposed system provides up to 24% energy saving as compared to the time-redundancy system. This is because, in this case, the time-redundancy system does not exploit its static slack for energy saving and reserves the static slack for fault tolerance (re-execution). However, in the proposed standby-sparing system, fault tolerance is decoupled from the slack time (Section 2), hence the static slack is exploited to reduce the energy consumption. It should also be noted that, for relaxed time constraints, the spare can be usually kept idle; hence the spare consumes very little energy.
- Although for all the three distributions we used exactly the same static schedules, both the standby-sparing and time-redundancy systems consume less energy for the exponential distribution as compared to the other two distributions. This is because our study shows that for the exponential distribution the average amount of dynamic slack was about 570ms, while for the normal and uniform distributions the average amount of dynamic slack was about 400ms. Therefore, as both the standby-sparing and time-redundancy systems exploit dynamic slacks to save energy, the exponential distribution results in less energy consumption.
- It can be seen from Table 2 that as the number of tasks in a schedule increases, the energy saving which is achieved by the proposed standby sparing system increases. This is because our study shows that as the number of tasks in a schedule increases the average amount of dynamic slack increases. On the other hand, the proposed standby-sparing

system benefits from this increase in slack more than the time-redundancy system, because when enough slack exists, the proposed standby-sparing system is able to keep the spare idle almost all the time which results in considerable energy saving. The reason why an increase in the number of tasks results in an increase in the average amount of dynamic slack is that for the first few tasks of all schedules, the amount of available dynamic slack is small. In fact, for the first task of a schedule, no dynamic slack is available. The second task can only use the dynamic slack that is leftover from the first task. The third task can only use the dynamic slack which is leftover from the first and second tasks, and so forth. Therefore, the first few tasks of a schedule have lower chance to obtain dynamic slack time as compared to the tasks that come later in the schedule. However, as the number of tasks in a schedule increase, this slack shortage that only exists for the first few tasks becomes proportionately negligible. For example, in the experiments, we observed that when the distribution of actual execution times is uniform, the average available dynamic slack for each task is as follows:

Average dynamic slack for T_1=0 ms
Average dynamic slack for T_2=364 ms
Average dynamic slack for T_3=388 ms
Average dynamic slack for T_4=396 ms
Average dynamic slack for the subsequent tasks ≈ 400 ms

6. SUMMARY AND CONCLUSION

The use of hardware-redundancy techniques for real-time systems is necessary when high reliability is the primary concern. However, hardware-redundancy techniques can excessively increase the energy consumption. In this paper, we propose a hardware-redundancy technique with low energy-overhead which uses standby-sparing to achieve fault tolerance for hard real-time systems. In the proposed standby-sparing system, DVS is used to reduce the energy consumption of the primary unit and DPM is used to reduce the energy consumption of the spare. Indeed, DVS is not used for the spare unit to avoid degrading the reliability of the spare. Through an analytical approach, we have developed an online energy management method for the proposed standby-sparing system which exploits dynamic slacks to reduce the energy consumption. The experimental results show that the energy and execution time overhead of the proposed online energy management method when applied to MiBench benchmarks (Auto./Industrial set) is always less than 0.15%, which is quite negligible. The results also show that for relaxed time constraints, the proposed system consumes about 24% less energy than the time-redundancy system. For tight deadlines when the time-redundancy system can tolerate no faults, the proposed system is still fault tolerant but consumes about 32% more energy than the time-redundancy system.

APPENDIX A

In this appendix, we prove the following theorems about the proposed standby-sparing system:

Theorem 1: For each task T_i, the minimum possible value of the normalized supply voltage ρ_i is $WT_i/(d_i+WT_i)$.

Proof: Since we have considered a hard real-time system, for each task T_i, the normalized supply voltage ρ_i should be determined so that the deadline is guaranteed to be met. The time at which the task T_i starts running is ST_i (given by Eq. 2) and in the worst case the execution of this task takes time WT_i/ρ_i. Therefore, in the worst case, when the task T_i finishes at the time ST_i+WT_i/ρ_i, the time which is left before the deadline is:

$$RMT_i = deadline - ST_i - \frac{WT_i}{\rho_i} \tag{30}$$

To guarantee that the deadline will not be missed, this time should be enough to execute all the remaining tasks (T_{i+1} through T_n) at the maximum voltage V_{MAX}. Hence:

$$\begin{aligned} &\sum_{j=i+1}^{n} WT_j \le RMT_i \quad \Rightarrow \\ &\sum_{j=i+1}^{n} WT_j \le deadline - ST_i - \frac{WT_i}{\rho_i} \;\Rightarrow \\ &\frac{WT_i}{\rho_i} \le deadline - \sum_{j=1}^{i-1}\frac{AT_j}{\rho_j} - \sum_{j=i+1}^{n} WT_j \end{aligned} \tag{31}$$

Using Eq. 4, Inequality 31 can be rewritten as:

$$\frac{WT_i}{\rho_i} \le d_i + WT_i \tag{32}$$

However, this inequality can be rearranged to:

$$\rho_i \ge \frac{WT_i}{d_i + WT_i} \tag{33}$$

and the theorem is proved. ■

Theorem 2: For each task T_i, the delay d_i is not less than (AT_i/ρ_i)-AT_i.

Proof: It can be simply shown that:

$$WT_i \ge AT_i \quad \Rightarrow \quad \frac{WT_i}{d_i + WT_i} \ge \frac{AT_i}{d_i + AT_i} \tag{34}$$

Based on Inequalities 33 and 34, we have:

$$\rho_i \ge \frac{AT_i}{d_i + AT_i} \tag{35}$$

This inequality can be rearranged to:

$$d_i \ge \frac{AT_i}{\rho_i} - AT_i \tag{36}$$

and the theorem is proved. ■

Theorem 3: The optimum value of ρ_i which minimizes the energy $NE(T_i)$ (given by Eq. 26) cannot be calculated at the start of the task T_i.

Proof: Let $\hat{\rho}_i$ be the optimum value of ρ_i which minimizes the energy $NE(T_i)$. Using calculus, we can conclude from Eq. 26 that the optimum value $\hat{\rho}_i$ is:

$$\hat{\rho}_i = \begin{cases} \frac{AT_i}{d_i} & \frac{WT_i}{d_i + WT_i} \le \frac{AT_i}{d_i} < \sqrt[3]{1/2} \\ \sqrt[3]{1/2} & \sqrt[3]{1/2} \le \frac{AT_i}{d_i} \end{cases} \tag{37}$$

It can be seen from Eq. 37 that the optimum value $\hat{\rho}_i$ depends on the actual execution times AT_i, however the actual execution time is random and not known at the start of the task T_i. Hence, it is impossible to calculate the optimum value $\hat{\rho}_i$ at the start of the task T_i. ■

7. REFERENCES

[1] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Scheduling of Fault-Tolerant Embedded Systems with Soft and Hard Timing Constraints", in *Proc. Design, Automation and Test in Europe* (DATE '08), pp. 915-920, March 2008.

[2] R. Melhem, D. Mosse, and E. Elnozahy, "The interplay of power management and fault recovery in real-time systems," *IEEE Trans. Computers*, vol. 53, no. 2, pp. 217-231, 2004.

[3] Y. Zhang and K. Chakrabarty, "Dynamic adaptation for fault tolerance and power management in embedded real-time systems," *ACM Tran. Embedded Computing Systems*, vol. 3, no. 2, pp. 336-360, 2004.

[4] F. Liberato, R. Melhem, and D. Mosse, "Tolerance to multiple transient faults for aperiodic tasks in hard real-time systems," *IEEE Trans. Computers*, vol. 49, no. 9, pp. 906-914, 2000.

[5] P. Eles, V. Izosimov, P. Pop, and Z. Peng, "Synthesis of Fault-Tolerant Embedded Systems", in *Proc. Design, Automation and Test in Europe* (DATE '08), pp. 1117-1122, March 2008.

[6] A. Ejlali, B.M. Al-Hashimi, M.T. Schmitz, P. Rosinger, and S.G. Miremadi, "Combined Time and Information Redundancy for SEU-Tolerance in Energy-Efficient Real-Time Systems", *IEEE Trans. VLSI Sys.*, vol. 14, no. 4, pp. 323-335, April 2006.

[7] I. Koren, and C. M. Krishna, *Fault-Tolerant Systems*, Morgan Kaufmann, Elsevier, 2007.

[8] Y. Zhang and K. Chakrabarty, "A Unified Approach for Fault Tolerance and Dynamic Power Management in Fixed-Priority Real-Time Embedded Systems", *IEEE Trans. CAD*, vol. 25, no. 1, pp. 111-125 JAN. 2006.

[9] A. M. K. Cheng, *Real-Time Systems, Scheduling, Analysis, and Verification*, John Wiley & Sons, 2002.

[10] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles, *System-Level Design Techniques for Energy-Efficient Embedded Systems*, Norwell, MA: Kluwer, 2004.

[11] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, no. 11, pp. 1571–1580, Nov. 2000.

[12] K. Marti, *Stochastic Optimization Methods*, Second Edition, Springer, 2008.

[13] P. Li, and B. Ravindran, "Fast, Best-Effort Real-Time Scheduling Algorithm", *IEEE Trans. Copuuters*, vol. 53, no. 9, Sept. 2004.

[14] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks", *IEEE Trans. Computers*, vol. 53, no. 5, May 2004.

[15] D. Zhu, R. Melhem, D. Mosse, and E. Elnozahy, "Analysis of an energy efficient optimistic TMR scheme", in *Proc. 10th Int'l Conf. Parallel and Distributed Systems* (ICPADS 2004), pp. 559-568, July 2004.

[16] S. Poledna, *Fault-tolerant real-time systems: The problem of replica determinism*, Kluwer Academic Publishers, 1996.

[17] H. Kopetz, *Real-time systems: Design principles for distributed embedded applications*, Kluwer Academic Publishers, 2002.

[18] D.K. Pradhan, *Fault-tolerant computer system design*, Prentice-Hall, 1996.

[19] R. Jejurikar, and R. Gupta, "Dynamic slack reclamation with procrastination scheduling in real time embedded systems", in *Proc. 42nd Design Automation Conference* (DAC 2005), pp. 111-116, June 2005.

[20] "*TM5400/TM5600 Data Book*", Transmeta Corp., Santa Clara, CA, 2000.

[21] http://www-micrel.deis.unibo.it/sitonew/research/mparm.html

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst,T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", in *Proc. IEEE 4th annual Workshop on Workload Characterization*, pp. 83-94, 2001.

[23] L. Benini, D. Bertozzi, A. Bogoliolo, F. Menichelli, and M. Olivieri., "MPARM: Exploring the Multi-Processor SoC Design Space with SystemC", *The Journal of VLSI Signal Processing*, vol. 41, no. 2, pp. 169-182, 2005.

[24] http://www.rtems.com