Optimistic Reliability Aware Energy Management for Real-Time Tasks with Probabilistic Execution Times*

Dakai Zhu
Univ. of Texas at San Antonio
San Antonio, TX, 78249
dzhu@cs.utsa.edu

Hakan Aydin
George Mason Univ.
Fairfax, VA 22030
aydin@cs.gmu.edu

Jian-Jia Chen
Swiss Federal Inst. of Technology
Zurich, Switzerland
chen@tik.ee.ethz.ch

Abstract

Reliability-aware power management (RAPM) schemes have been recently studied to save energy while preserving system reliability. The existing RAPM schemes, however, provision for worst-case execution scenarios and are rather conservative*. In this paper, by exploiting the probabilistic execution time information of real-time tasks, we develop an* optimistic *RAPM scheme. Instead of scheduling a* full *recovery for tasks whose executions are scaled down, the new scheme puts aside* just enough *slack to guarantee the required reliability leave while leaving more slack for energy management to achieve better energy savings. The problem is shown to be NP-hard and a novel heuristic algorithm is proposed and evaluated. The simulation results show that the optimistic RAPM scheme performs very well. It achieves energy savings comparable to that of the* ordinary (but reliability-ignorant) *power management scheme, while maintaining the system reliability as successfully as the conservative RAPM schemes.*

1 Introduction

Due to the proliferation of the battery-operated embedded devices, energy has been recognized as a first-class resource in computing systems and many software/hardware power management techniques have been studied. As one of the most commonly used power management techniques, *dynamic voltage and frequency scaling (DVFS)* scales down the supply voltage and operating frequency of modern processors to save energy [23]. However, applications generally take more time to complete at lower processor operating frequencies. Therefore, for real-time systems that usually have stringent timing constraints and the consequences of missing a deadline can be very serious, special provisions are needed when applying DVFS for energy savings. For various real-time task and system models, by exploiting the available static and/or dynamic *slack* in the system, several recent research studies explored the problem of minimizing energy consumption while meeting the timing constraints [2, 15, 20, 24, 25].

Traditionally, reliability and fault tolerance have been the major concerns in computer system design as faults may occur due to various reasons (such as soft errors due to cosmic ray radiations or electromagnetic interference). As CMOS technologies continue to scale down and design margins are reduced for higher performance, it is expected that, in addition to the systems that operate in electronics-hostile environments, practically all digital computing systems will be strikingly vulnerable to transient faults [8]. Moreover, as transient fault rates increase at lower supply voltages [5, 30], the negative effects of DVFS on reliability introduce additional complications. Therefore, *reliability-cognizant* energy management becomes a necessity – especially for safety-critical real-time systems (such as satellite and surveillance systems) where reliability is as important as energy efficiency.

Focusing on transient faults, a number of recent research efforts addressed energy efficiency and system reliability dimensions simultaneously [6, 16, 22, 26, 27]. However, most of the previous research either focused on tolerating a fixed number of faults [16, 22], or assumed constant transient fault rate [26]. To address the negative impact of DVFS on system reliability due to increased transient fault rates at lower supply voltages [5, 30], we proposed a *reliability-aware power management (RAPM)* scheme [28]. As opposed to the *ordinary (but reliability-ignorant)* power management schemes that exploit *all* the available slack for energy savings [2, 4, 17, 20], the central idea of the RAPM scheme is to put aside a portion of available slack before utilizing the remaining slack to scale down the execution of tasks for saving energy. For tasks whose executions are scaled down through DVFS, the reserved slack will be used to accomodate the needed

*This work was supported in part by NSF awards CNS-0720651, CNS-0720647, NSF CAREER Award CNS-0546244, Taiwan National Science Council Award NSC-096-2917-I-564-121, and the European Community's Seventh Framework Programme FP7/2007-2013 project Predator.

recovery tasks for reliability preservation in case that the transient faults occur during the scaled execution. The RAPM framework has been extended to various real-time task models and reliability requirements [29, 31, 32].

However, by considering the worst-case execution time (WCET) of tasks when reserving the slack for recovery tasks (which is invoked at the maximum processing speed only if the corresponding scaled tasks fail), the existing RAPM schemes adopt a rather *conservative* solution. For example, if the amount of available slack is not enough to accomodate a *full* recovery for the task to be dispatched under the worst-case scenario, the existing schemes will not scale down the task and the slack may be wasted [29]. In fact, real-time tasks typically only take a small fraction of their WCETs [9]. Further, the detailed probability distribution information of tasks' execution times can be obtained through either analysis or profiling [15, 25]. By utilizing such statistical information, several stochastic energy management schemes were proposed [15, 24, 25]; but none of them addressed the system reliability issues.

The main contribution of this paper is the introduction and analysis of an *optimistic* RAPM scheme for real-time tasks with given probability distributions of their execution times. By exploiting such statistical information, instead of scheduling a full recovery for tasks to be scaled down, the new scheme reserves only a small portion of the system slack that is *just sufficient* to preserve the required system reliability. Therefore, more slack is available for energy management, enabling additional energy savings.

The remainder of the paper is organized as follows. The system models are presented in Section 2. Section 3 first illustrates the problem with a motivational example and then analyzes the case with a single task. In Section 4, the optimistic RAPM problem for multiple tasks is formulated and shown to be NP-hard. Then, an efficient heuristic algorithm is proposed. The new scheme is evaluated in Section 5, and Section 6 concludes the paper.

2 System Models

2.1 Application Model

We consider applications with a set of n independent periodic real-time tasks $\{T_1, \ldots, T_n\}$. Task T_i ($i = 1, \ldots, n$) is represented by its period p_i and two associated vectors, $C_i = \{c_i^1, \ldots, c_i^{k_i}\}$ and $PR_i = \{pr_i^1, \ldots, pr_i^{k_i}\}$, which are used to represent the task's possible execution times and the corresponding probabilities, respectively. That is, there are k_i different possible execution times for executing task T_i and the probability for task T_i to take execution time c_i^j is pr_i^j ($j = 1, \ldots, k_i$). Here, for any task T_i, we have $\sum_{j=1}^{k_i} pr_i^j = 1$.

Note that, the number of different execution times k_i for task T_i and the corresponding probabilities can be obtained through either analysis or profiling [15, 25]. We further assume that $c_i^1 < c_i^2 < \ldots, < c_i^{k_i}$, where c_i^1 and $c_i^{k_i}$ are the best-case execution time (BCET) and worst-case execution time (WCET) of task T_i, respectively. In addition, considering the DVFS feature of the systems, it is assumed that the execution times are given under the maximum processing frequency f_{max} and they scale linearly with reduced processing frequencies.

The utilization of task T_i is defined as $u_i = \frac{c_i^{k_i}}{p_i}$ and $U = \sum_{i=1}^{n} u_i$ is the system utilization. We use the preemptive *earliest deadline first (EDF)* scheduling policy and assume that $U \leq 1$. Moreover, it is assumed that the first job of each task is assumed to arrive at time 0 and the j'th job $J_{i,j}$ of task T_i arrives at time $(j-1) \cdot p_i$, with the deadline of $j \cdot p_i$ ($j \geq 1$).

2.2 Power Model

Although the power consumption of modern processors is dominated by their dynamic power [3], to incorporate the fast-growing static/leakage power and the power consumed by components other than processors, the system-wide power management has caught researchers' attention recently [1, 13]. In this paper, we adopt the system-level power model where the power consumption of a computing system is given by [30]:

$$P(f) = P_s + \hbar(P_{ind} + P_d) = P_s + \hbar(P_{ind} + C_{ef} f^m) \quad (1)$$

Here, P_s is the *static power*, which includes the power to maintain basic circuits, and to keep the clock running and the memory in sleep modes. P_s can be removed only by powering off the whole system. P_{ind} is a constant *frequency-independent active power*, which denotes the power consumed by off-chip devices such as main memory and external devices and can be efficiently removed by putting the system components to sleep states. P_d is the *frequency-dependent active power*, which includes any power that depends on the processing frequencies. The effective switching capacitance C_{ef} and the dynamic power exponent m (in general, $2 \leq m \leq 3$) are the system-dependent constants [3] and f is the processing frequency. $\hbar = 1$ when the system is *active* (i.e., computation is in progress); otherwise, $\hbar = 0$. Despite its simplicity, the above model captures the essential components of the system-level power.

Due to the excessive time/energy overhead associated with turning on/off a system [7], we assume that the system is always on. Therefore, P_s is not manageable and we will concentrate on the frequency-independent and frequency-dependent active power (i.e., P_{ind} and P_d) in our analysis and evaluation. Note that energy is the integral of power over time. From Equation (1),

we can see that the energy consumption due to P_d decreases with reduced frequency while the energy consumption from P_{ind} will increase due to extended execution time. Through simple algebraic transformation, one can obtain a minimal *energy-efficient frequency*, $f_{ee} = \left(\frac{P_{ind}}{(m-1)C_{ef}}\right)^{\frac{1}{m}} f_{max}$, below which DVFS ceases to be energy-efficient [30].

We use normalized frequencies in this paper and assume that $f_{max} = 1.0$. Moreover, we assume that there are k frequency/voltage levels available for the processor under consideration ($f_1 < f_2 < \ldots < f_k$) in the range of the minimal frequency f_{min} and f_{max}. That is, we have $f_1 = f_{min}$ and $f_k = f_{max}$.

2.3 Fault and Recovery Models

During the execution of an application, a fault may occur due to various reasons, such as hardware failure, the effects of electromagnetic interferences or cosmic ray radiations. In this paper, we focus on *transient* faults as they occur much more frequently than *permanent* faults [12], especially with the continued scaling of CMOS technologies and reduced design margins [8].

Traditionally, transient faults have been modeled as following the Poisson distribution with an average arrival rate λ [26]. Moreover, it has been recently shown that the average arrival rate of transient faults depends largely on the system supply voltage and processing frequency considering the fact that transient faults are more likely to occur at lower supply voltages (especially for the ones caused by cosmic ray radiations) [5, 30]. Therefore, for the DVFS-enabled embedded systems to be considered in this paper, the average arrival rate of transient faults at a reduced frequency f (and the corresponding voltage level) can be modeled as [30]:

$$\lambda(f) = \lambda_0 \cdot g(f) \tag{2}$$

where λ_0 is the average fault arrival rate at the maximum frequency f_{max} (and corresponding supply voltage V_{max}). That is, $g(f_{max}) = 1$. In general, for a lower frequency $f < f_{max}$, we have $g(f) > 1$.

Note that, transient fault rates are shown to be exponentially-related to the circuits *critical charge* (which is the smallest charge required to cause a soft error in a circuit node) [10] and the critical charge is proportional to the system supply voltage [21]. Focusing on the transient faults caused by cosmic ray radiations, we consider in our analysis and simulations a specific exponential fault rate model that is derived from historically data regarding transient faults and cosmic ray radiations [30]:

$$\lambda(f) = \lambda_0 g(f) = \lambda_0 10^{\frac{d(1-f)}{1-f_{min}}} \tag{3}$$

where d (> 0) is a constant, indicating the sensitivity of transient fault rates to voltage scaling. That is, reducing the supply voltage and frequency results in exponentially increased transient fault rates. The maximum average fault rate is assumed to be $\lambda_{max} = \lambda_0 10^d$, which corresponds to the minimum frequency f_{min} (and supply voltage V_{min}) [30].

Following the same approach that has been adopted in previous works [18, 26], we exploit temporal redundancy and use the *backward error recovery* techniques to recover from transient faults [19]. That is, when such transient fault(s) are detected at the end of a job's execution (using *sanity* or *consistency* checks [19]), the system's state is restored to a previous safe state and the job is re-executed. For simplicity, the overhead for fault detection and restoring system state is assumed to be negligible (or it can be incorporated into tasks' WCETs). Moreover, to preserve system reliability, the re-execution will be performed at the maximum processing frequency f_{max} [28].

3 The Case of a Single Task

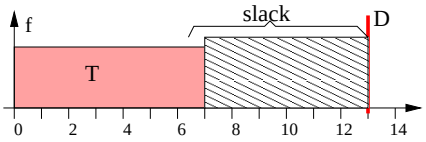
Before addressing the general problem for a set of periodic tasks, in this section, we focus on the case of a single task. In what follows, we first illustrate the problem with a concrete example.

3.1 Motivational Example

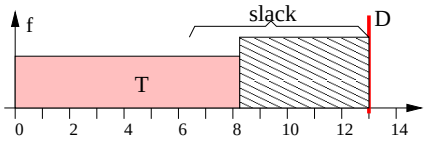
Suppose that the application under consideration has only one task T with a period of 13 time units. Its execution time vector is $C = \{2, 4, 6\}$ and the corresponding probability vector is $PR = \{0.1, 0.8, 0.1\}$. That is, the probability that the task will take its WCET (i.e., 6) is only 10%. The probabilities of having execution time as 4 and 2 time units are 80% and 10%, respectively.

Define the *original reliability* of a task as the probability of completing its execution without incurring transient faults when there is no power management (i.e., when it runs at f_{max}), without any recovery. Suppose that the average fault arrival rate at f_{max} is $\lambda_0 = 10^{-6}$. Taking the probability distribution of task T's execution times into consideration, T's *original reliability* can be calculated as 0.999996 [30]. To preserve the task's reliability, the existing *conservative RAPM (C-RAPM)* scheme considers only the WCETs of tasks [28]. For illustration purposes, suppose that the processing frequency can be varied continuously from f_{min} (e.g., 0.2) to f_{max}.

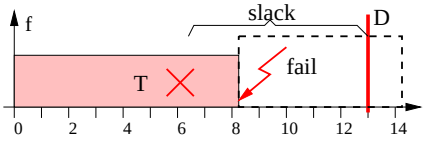
As shown in Figure 1a, C-RAPM reserves 6 time units for the recovery (to accomodate the worst-case scenario when re-executing the task) and utilizes the remaining 1 unit of slack to scale down the *primary* execution of task T to the frequency of 0.86. In this case, the reliability achieved for task T will be the summation of the probability that T's scaled execution incurs no faults and the probability of T's scaled execution fails but its recovery succeeds. From [28], we know that the C-RAPM scheme preserves the task T's original reliability.



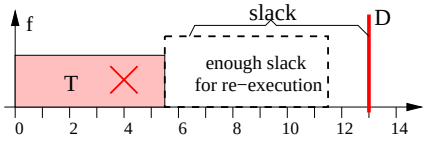
a. Conservative RAPM with scaled frequency $f = 0.86$



b. Optimistic RAPM with scaled frequency $f = 0.73$



c. T takes its WCET with scaled frequency $f = 0.73$ and fails



d. T takes $\frac{2}{3}$ of its WCET with with scaled frequency $f = 0.73$

Figure 1. An Example: Comparison of Optimistic RAPM with Conservative RAPM.

Instead of reserving 6 units of slack to accommodate the WCET for re-execution, less slack may be reserved for recovery to preserve the task's original reliability considering the fact that task T may use less time than its WCET and complete earlier. For instance, considering the probability distribution of execution times for task T, as shown in Figure 1b, only 4.75 units of slack are reserved for the recovery and the remaining slack is used to scale down task T to the frequency of 0.73. For simplicity, we assume that the recovery (i.e., re-execution) is activated only if there is enough slack for their WCETs. When task T takes its WCET (with only 10% probability) at run time and fails, the reserved slack is not enough for re-executing task T at f_{max}, and thus leads to a failure (as shown in Figure 1c). However, when T takes no more than 4 time units at run time (with 90% probability), there will be enough slack for the recovery (as shown in Figure 1d). Assuming that $d = 2$ (i.e., $\lambda_{max} = 10^{-4}$ at $f_{min} = 0.2$), the achieved reliability under this *optimistic RAPM* scheme will be 0.99999603, which is better than T's original reliability.

Moreover, by optimistically using the slack, better energy savings can be expected. Assume that $P_s = 0$, $P_{ind} = 0.01$, $C_{ef} = 1$ and $m = 3$. The optimistic RAPM scheme saves 46% energy when compared to the no power management (NPM) case, while the C-RAPM scheme saves only 26%, which is a significant improvement. Therefore, through judicious selection of the scaled frequency and the amount of slack for recovery, we can reserve less recovery slack than that of the conservative RAPM scheme for reliability preservation, while using more slack to scale down the execution of tasks for more energy savings.

3.2 The Impact of Time Allocation and Frequency Level

The above example illustrates that the amount of slack reserved for recovery and the scaled frequency of the primary execution of a task are critical to preserve the task's reliability and to maximize energy savings. In what follows, focusing on a single task with given probability distribution of its execution times, we investigate how to obtain the scaled frequency that leaves *just enough slack* to preserve its reliability.

Original Reliability and Expected Energy: When T_i is executed at f_{max} under NPM and no recovery being considered, the *original reliability* for T_i can be calculated as (recall that the transient faults follow Poisson distribution and the average fault arrival rate is λ_0 at f_{max}):

$$R_i^0 = \sum_{j=1}^{k_i} (pr_i^j \cdot R_i^j) = \sum_{j=1}^{k_i} (pr_i^j \cdot e^{-\lambda_0 c_i^j}) \qquad (4)$$

Note that the above equation takes the probability distribution of task's execution times into consideration. Similarly, the expected energy consumption for task T_i under NPM can be calculated as:

$$E_i^0 = \sum_{j=1}^{k_i} (pr_i^j \cdot E_i^j) = \sum_{j=1}^{k_i} (pr_i^j \cdot P(f_{max}) c_i^j) \qquad (5)$$

Scaled Execution with Recovery: Suppose that the amount of available time for task T_i within its period p_i is D_i ($c_i^{k_i} \leq D_i \leq p_i$), the amount of available slack for task T_i is $D_i - c_i^{k_i}$, which can be used to scale down the execution frequency of task T_i for energy savings and/or to schedule a recovery to enhance/preserve the task's reliability. Suppose that the *single* scaled frequency[1] for task T_i is f ($\geq \frac{c_i^{k_i}}{D_i}$). Considering the probability distribution

[1]By exploiting the task's probabilistic execution time information, better energy savings can be obtained through intra-task DVFS that gradually increases the frequency of a task during its execution [15, 24, 25]. However, exploring such possibilities is well beyond the scope of this paper and is left as future work. For simplicity, we assume that a single scaled frequency will be determined for each task.

of task's execution times and the possible recovery, the reliability achieved $R_i(f, D_i)$ for task T_i will be:

$$R_i(f, D_i) = \sum_{j=1}^{k_i} pr_i^j R_i^j(f, D_i) \quad (6)$$

where $R_i^j(f, D_i)$ is the probability of task T_i completing its execution correctly in time D_i when it is executed at a scaled frequency f and the scaled execution takes $\frac{c_i^j}{f}$ time units. $R_i^j(f, D_i)$ is defined as follows:

$$R_i^j(f, D_i) = e^{-\lambda(f)\frac{c_i^j}{f}} + (1 - e^{-\lambda(f)\frac{c_i^j}{f}}) RR_i(D_i - \frac{c_i^j}{f}),$$

$$RR_i(t) = \begin{cases} 0 & t < c_i^{k_i} \\ R_i^0 & t \geq c_i^{k_i} \end{cases}.$$

Here, the first term on the right hand side of the expression $R_i^j(f, D_i)$ is the probability that task T_i's scaled execution will incur no faults. The second term represents the probability of having a failed scaled (primary) execution but a successful recovery (i.e., re-execution). Recall that, the recovery is assumed to be activated only if the remaining slack is no less than the WCET of the task.

Similarly, the expected energy consumption $E_i(f, D_i)$ for task T_i can be obtained as:

$$E_i(f, D_i) = \sum_{j=1}^{k_i} pr_i^j \left(P(f)\frac{c_i^j}{f} + (1 - e^{-\lambda(f)\frac{c_i^j}{f}}) RE_i(D_i - \frac{c_i^j}{f}) \right) \quad (7)$$

where $RE_i(t)$ is the expected energy consumption from the re-execution, given as follows:

$$RE_i(t) = \begin{cases} 0 & t < c_i^{k_i} \\ E_i^0 & t \geq c_i^{k_i} \end{cases}$$

Therefore, to maximize the expected energy savings while preserving the task's original reliability R_i^0, we need to find out the optimal scaled frequency f to

$$\text{minimize } E_i(f, D_i) \quad (8)$$

subject to

$$R_i(f, D_i) \geq R_i^0, \quad (9)$$

$$f \geq \max\{\frac{c_i^{k_i}}{D_i}, f_{ee}\} \quad (10)$$

$$f \in \{f_1, \ldots, f_k\}. \quad (11)$$

Intuitively, when task T_i runs at lower frequencies, it needs more execution time for DVFS and more energy savings can be obtained. But, at lower frequencies, the scaled execution of T_i is more likely to fail and more slack is needed for recovery to preserve its reliability. However, it is not hard to see that, for a task with given time allocation within its period, finding the close formula for the optimal scaled frequency to minimize the task' expected energy consumption (i.e., to maximize energy savings) while satisfying the above equations is extremely challenging.

Note that, when the reserved slack is no less than a task's WCET, the task's reliability will definitely be preserved [28]. That is, we have $f \leq \min\left\{\frac{c_i^{k_i}}{D_i - c_i^{k_i}}, f_{max}\right\}$. After eliminating the frequency levels that are infeasible, the optimal frequency level that minimizes the expected energy consumption can be found efficiently (recall that modern CPUs have typically small number of speeds).

4 Optimistic RAPM Scheme

4.1 Time/Energy Tables for Tasks

Following an approach similar to the one given in the last section, we can also find out the minimum time allocation D_i^j needed for a task T_i to run at a given scaled frequency f_j ($f_1 \leq f_j \leq f_k$), *while preserving its original reliability*. Note that, if the time allocation for task T_i is no less than what is needed for T_i to run at f_j but still has slack for a full recovery in the worst-case scenario (which actually is the time requirement for the *conservative* RAPM), we know that task T_i's reliability will be preserved. That is, we have $D_i^j \leq \frac{c_i^{k_i}}{f_j} + c_i^{k_i}$. In addition, for feasibility, the time allocation D_i for the task T_i should be at least enough for it to complete with its WCET at the scaled frequency f_j. Therefore, we get:

$$\frac{c_i^{k_i}}{f_j} \leq D_i^j \leq \frac{c_i^{k_i}}{f_j} + c_i^{k_i} \quad (12)$$

Note that the achieved reliability for a task increases monotonically as more time is allocated to it. Through binary search, D_i^j can be found efficiently using Equations 4 and 6. For example, within 20 iterations, the error for D_i^j can be reduced to 0.0001%.

Hence, for each real-time task with given probability distribution of its execution times, we can compute *offline* a **time-energy table** that contains the *minimum time requirement* for the task to run at any scaled frequency (while preserving its original reliability) and the corresponding *expected energy savings*. For systems with k frequency levels, the table will have k entries.

Table 1 shows an example table for task T_i. As the first entry in the table, when the task T_i is expected to run at the maximum frequency $f_k = f_{max}$, the time allocation D_i^k will be the same as its WCET $c_i^{k_i}$. However, no energy saving is expected. For a lower frequency f_j ($1 \leq j < k$), the minimum time allocation

Table 1. Minimum Time Requirement versus Expected Energy Savings for Task T_i at Different Frequency Levels.

Frequency	Time requirement	Energy savings
f_k	$D_i^k = c_i^{k_i}$	0
f_{k-1}	D_i^{k-1}	ES_i^{k-1}
...	...	...
f_1	D_i^1	ES_i^1

requirement D_i^j for the task to preserve its original reliability can be calculated following the above-mentioned process. Moreover, the expected energy savings for the task to run at the scaled frequency can be obtained as $ES_i^j = E_i(f_j, D_i^j) - E_i^0$, where E_i^0 and $E_i(f_j, D_i^j)$ are defined as in Equations (5) and (7), respectively.

In general, at lower frequencies, energy savings tend to increase. However, if the transient fault rates are very sensitive to scaled voltages/frequencies (e.g., large value of d; see Section 2), the probability of the scaled primary execution of the task at lower frequencies incurring transient fault(s) could be relatively high. Thus, the energy overhead from the recovery (i.e., re-execution of the task at f_{max}) could offset the energy gains by executing the task at a low frequency. That is, it is possible that, $\exists j \in \{1, \ldots, n\}$, such that $ES_i^j < ES_i^{j+1}$ (see Section 5 for the analysis results). Therefore, for energy efficiency purposes, such frequency levels should be excluded from the management schemes.

Moreover, we can see that, the minimum time requirement for a task to preserve its original reliability at low frequency is more than what is needed at a higher frequency. That is, we have $D_i^{j+1} < D_i^j$ ($j = 1, \ldots, k-1$). If the minimum time requirement for task T_i to run at the scaled frequency f_j is more than the size of its period (i.e., $D_i^j > p_i$), it is not possible for the task to complete its execution in time. That, in turn, would mean that f_j is an *infeasible* frequency for task T_i. Suppose that the smallest feasible frequency level for task T_i is a_i (i.e., $D_i^{a_i} \leq p_i < D_i^{a_i-1}$). In what follows, for task T_i, we consider only the frequency levels with $f_j \geq f_{a_i}$.

4.2 Problem Formulation

For a system with a set of independent periodic real-time tasks, the system reliability relies on the reliability of each task. For simplicity, to preserve overall system reliability, we focus on preserving the reliability for individual tasks. Considering the preemptive *earliest deadline first (EDF)* scheduling, it is well known that the task set is schedulable if the system utilization $U \leq 1$ [14]. Define the spare CPU capacity as $SC = 1 - U$. For task sets with $SC > 0$, the **static optimistic RAPM problem** to be addressed in this work is: **how to allocate the spare CPU capacity SC to the tasks and find out their corresponding scaled frequencies such that their reliabilities are preserved and the total energy savings are maximized.**

We first define the *inflated* utilization for task T_i to run at frequency f_j as $iu_i^j = \frac{D_i^j}{p_i}$, where D_i^j is the minimum time allocation needed for task T_i to preserve its original reqliability at the scaled frequency f_j (as shown in Table 1). The least common multiple of all tasks' periods is denoted as LCM. The static optimistic RAPM problem can be formally presented as: finding the assignment of x_i^j for task T_i ($i = 1, \ldots, n; j = a_i, \ldots, k$), so as to:

$$\text{maximize } \sum_{i=1}^{n} \sum_{j=a_i}^{k} (ES_i^j \cdot x_i^j) \frac{LCM}{p_i} \tag{13}$$

subject to:

$$\sum_{i=1}^{n} \sum_{j=a_i}^{k} (iu_i^j \cdot x_i^j) \leq 1 \tag{14}$$

$$\sum_{j=a_i}^{k} x_i^j = 1, \ \forall i = 1, 2, \ldots, n \tag{15}$$

$$x_i^j \in \{0, 1\}, \ \forall i = 1, \ldots, n; \forall j = a_i, \ldots, k. \tag{16}$$

Here, the first condition ensures that the scaled task set is feasible under EDF. The last two conditions state that only one frequency level can be selected for any task.

Note that the special case of the above problem with $k_i = 1$ and $pr_i^1 = 1$ ($i = 1, \ldots, n$) is essentially the static RAPM problem studied in [29], which has been proved to be NP-hard. Therefore, the static optimistic RAPM problem is also intractable. In what follows, we propose an efficient heuristic scheme for this intractable problem.

4.3 Heuristic with Energy-Slack Ratio

Intuitively, when spare CPU capacity exists, to maximize the expected energy savings, we would like to scale down the task with the maximum expected energy savings and at the same time with the minimum increased utilization requirement. To quantify such property of tasks, we define the *energy-slack ratio (ESR)* as follows. For task T_i, its ESR_i^j at the frequency level f_j ($j = a_i, \ldots, k-1$) is defined as:

$$ESR_i^j = \frac{ES_i^j - ES_i^{j+1}}{(iu_i^j - iu_i^{j+1}) \cdot p_i} \tag{17}$$

Here, ESR_i^j quantifies how much energy can be saved per time unit when T_i is scaled down from f_{j+1} to f_j.

Algorithm 1 summarizes the steps for the ESR-based greedy heuristic. Here, after obtaining the time-energy table for all tasks based on the probability distributions of their execution times, ESRs are calculated for all tasks at every frequency level. Initially, all tasks are assumed

to run at the maximum frequency $f_k = f_{max}$ (line 2), which is assumed to be schedulable under EDF. Then the spare CPU capacity is calculated. After that, based on the ESR values for tasks at their next lower frequency levels, the task with the highest ESR value (provided that the additional utilization requirement is no more than the remaining CPU capacity SC) will be chosen to use the spare CPU capacity (line 5). After updating the remaining spare CPU capacity and the frequency selection (lines 6 and 7), the above steps will be repeated until the spare CPU capacity is used up or no task can further utilize the remaining spare CPU capacity.

Algorithm 1 : The ESR-based Greedy Algorithm

1: For all tasks, calculate the time-energy tables and ESR_i^j;
2: For all tasks, set frequency selection $z_i = k$;
3: $SC = 1 - U = 1 - \sum_{i=1}^{n} u_i$; //spare CPU capacity
4: **while** $SC > 0$ and $\exists i$ with $iu_i^{z_i-1} - iu_i^{z_i} \leq SC$ **do**
5: Find task T_y with $ESR_y^{z_y-1} > ESR_i^{z_i-1}$ and $iu_y^{z_y-1} - iu_y^{z_y} \leq SC$;
6: $SC = SC - (iu_y^{z_y-1} - iu_y^{z_y})$;
7: $z_y = z_y - 1$;
8: **end while**
9: return $(z_1, \ldots, z_n)$;

Note that, if the spare CPU capacity is large enough, the maximum number iterations of the *while-loop* in Algorithm 1 will be $n \cdot k$, where n is the number of tasks in a task set and k is the number of frequency levels in the system. Moreover, finding the task with the maximum ESR during each iteration (line 5) can be done efficiently in time $O(n)$. Therefore, the overall complexity of the algorithm is $O(k \cdot n^2)$.

5 Results and Discussions

In this section, we present an experimental evaluation of the proposed optimistic RAPM scheme for both expected energy savings and reliability preservation by comparing it to our previous conservative RAPM schemes [29], through extensive simulations. We first explain the parameters used in the evaluations.

Focusing on the active power, we assume that $P_s = 0$, $P_{ind} = 0.01$, $C_{ef} = 1$ and $m = 3$. Normalized frequency is used with $f_{max} = 1$ and the energy efficient frequency can be calculated as $f_{ee} = 0.17$ (see Section 2). We further assume that the minimum available frequency is $f_{min} = 0.2$ ($> f_{ee}$). The effects of different numbers of frequency levels are evaluated in Section 5.1.

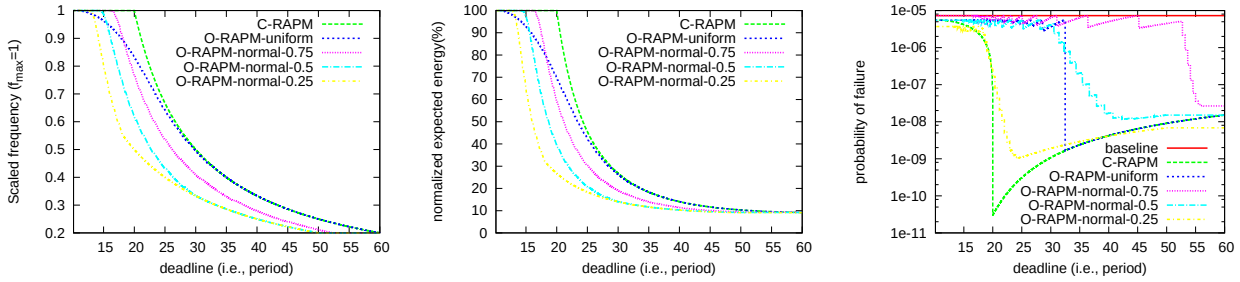
Transient faults are assumed to follow the Poisson distribution [26]. The average fault rate at the maximum frequency f_{max} (and corresponding supply voltage) is assumed to be $\lambda_0 = 10^{-6}$, which corresponds to $100,000$ FITs (failure in time, in terms of errors per billion hours of use) per megabit and is a reasonable fault rate as reported [10]. Taking the effects of DVFS on transient fault rates into consideration, the exponent in the exponential fault model (see Section 2) is assumed to be $d = 2$ (or 3). That is, the average fault rate is assumed to be 100 (or 1000, respectively) times higher at f_{min} (and corresponding supply voltage).

The WCET of tasks are generated randomly within the range of $[10, 100]$. The best-case execution time (BCET) for tasks is assumed to be 10% of their WCETs. Suppose that the number of different execution times within BCET and WCET (inclusive) of a task is ck (which is different for the tasks considered). We consider two different probability distributions regarding tasks' execution times. For *uniform* distribution, the probability of a task to take any one of its execution time is the same and equal to $\frac{1}{ck}$. The second one is the *modified discrete* normal distribution. We consider three cases with the average of $BCET + \frac{(WCET-BCET)}{4}$, $\frac{BCET+WCET}{2}$, and $BCET + \frac{3(WCET-BCET)}{4}$ (which are represented as *normal*-0.25, *normal*-0.5, and *normal*-0.75, respectively). For the cases with multiple tasks, the periods of tasks are generated such that their least common multiple is 7200. The execution times of tasks are normalized according to desired system utilizations [17].

5.1 Single Task

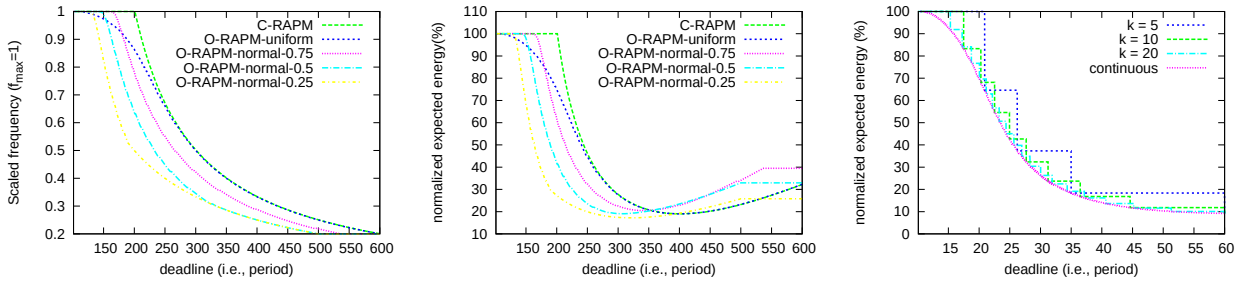
To investigate the relationship between the scaled processing frequency, expected energy savings and reliability for tasks with different probability distributions of their execution times, we first evaluate the case with a single task. Here, for the task under consideration, there are $WCET = 10$, $BECT = 1$ and $ck = 100$. Taking the probability distribution of the task's execution times into consideration, Figure 2a first shows the minimum frequency for the task to preserve its original reliability under the optimistic RAPM (O-RAPM) scheme, when the task has different deadlines (i.e., periods). Here, large values of deadlines indicate that more slack is available. The curve labeled with *O-RAPM-uniform* presents the the minimum frequency for the task to preserve its reliability under O-RAPM when the execution times of the tasks follow uniform distribution between $BCET$ and $WCET$. A similar labelling is used for normal distributions. For comparison, the frequency setting under the conservative RAPM (C-RAPM) is also shown. We start with continuous frequency (i.e., $k = \infty$) within $f_{min} = 0.2$ and $f_{max} = 1.0$. The effects of different discrete frequency levels are shown in Figure 3c.

From the results, we can see that, the O-RAPM scheme can generally set a lower frequency than that of the C-RAPM scheme, especially for the cases where the amount of available slack is limited. Therefore, less slack will be used for reliability preservation. Moreover, for different probability distributions of the task's execution times, the minimum frequency for the task to preserve its



a. minimal scaled frequency b. normalized expected energy c. probability of failure

Figure 2. Frequency selection of the Optimistic RAPM with corresponding energy savings and achieved reliability for a single task; here, $BCET = 1$, $WCET = 10$, $ck = 100$ and $d = 2$.



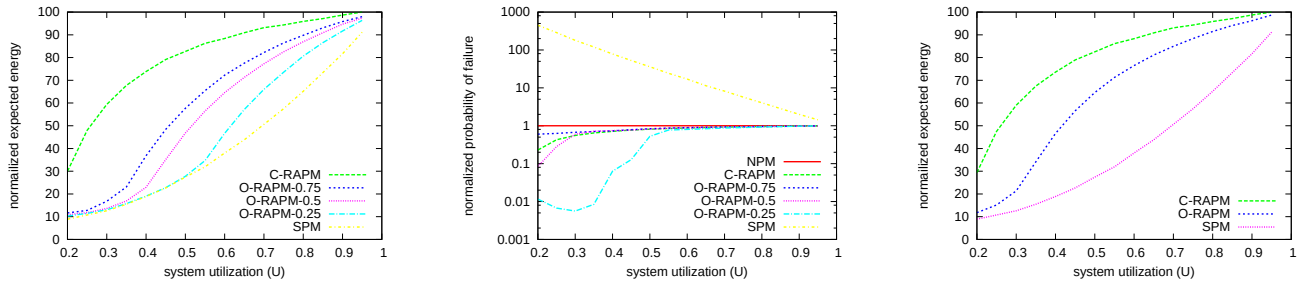
a. minimal scaled frequency b. normalized expected energy c. effects of discrete frequencies

Figure 3. Optimistic RAPM results for a large task with $BCET = 10, WCET = 100$ and $d = 3$; and the effects of discrete frequency levels for the small task with $BCET = 1$ and $WCET = 10$.

reliability under O-RAPM is different. In general, normal distributions with low average execution times lead to lower frequency settings and thus result in more energy savings as shown in Figure 2b. Note that, when the amount of available slack is less than WCET (i.e., when deadline is less than 20), C-RAPM will not scale down the execution of the task and no energy savings could be obtained. However, the new O-RAPM scheme could scale down the task even when the available slack is less than its WCET by considering the probability distribution of task's execution times. For instance, when the deadline (i.e., period) equals 18, the frequency setting under O-RAPM for the distribution *'normal-0.25'* is about 0.55, which results in around 70% of energy savings.

Figure 2c further shows the *probability of failure* (i.e., $1 - reliability$) for the schemes. Here, lower values for the probability of failure mean better reliabilities. We can see that the O-RAPM scheme can preserve the task's original reliability, while C-RAPM generally achieves much better reliability by conservatively reserving the full recovery (i.e., WCET). Here, the sharp drops in the probability of failure come from the fact that, full recovery is actually reserved even under O-RAPM when the amount of available slack is large enough.

However, we want to underline that the expected energy consumption does not decrease monotonically with reduced scaled frequency when more slack is available. For instance, for a larger task with $BCET = 10$, $WCET = 100$ and $ck = 100$, the minimum scaled frequency for reliability preservation is shown in Figure 3a and the corresponding expected energy consumption of the task is shown in Figure 3b. Here, we assume that $d = 3$. Note that, the expected energy consumption starts to increase after a certain point when the deadline (i.e., period) increases. The reason is that, for larger tasks and large values of d, the scaled execution of the primary task is more likely to fail, especially at low frequencies. Further, the energy consumed by the recovery (i.e., re-execution at f_{max}) will be more likely to offset the energy savings obtained from the scaled execution. Therefore, there exists a *reliability-aware energy efficient frequency*, which relies on both task's execution time and the transient fault rate increases. This frequency is normally larger than the energy efficient frequency (f_{ee}). For energy efficiency with reliability preservation requirements, a task should not be scaled below this threshold.



a. energy with normal distribution b. probability of failure with normal distribution c. energy with uniform distribution

Figure 4. The performance of O-RAPM under different system utilizations.

For the small task discussed above, Figure 3c further shows the effects of the number of frequency levels (k). From the figure, we can see that, in accordance with previous studies, the expected energy consumption with 10 frequency levels is close to that of the continuous frequency. In what follows, we assume that there are $k = 10$ frequency levels from f_{min} to f_{max} (inclusive).

5.2 Multiple Tasks

To analyze the performance of the O-RAPM heuristic scheme for multiple periodic tasks, we show the evaluation results for task sets with 20 periodic tasks. In addition to the task-level C-RAPM scheme [29], we also show the results for the ordinary static power management (SPM), which scales down all tasks uniformly to the frequency of $f = U$ (where U is the system utilization of the task set) [2, 17]. For cases where f falls between two frequency levels, we use these two frequency levels to approximate f following the similar idea in [11]. Each result point in the figure corresponds to the average of 100 randomly generated task sets.

For the execution times of tasks in task sets following normal distributions, Figure 4a first shows the normalized expected energy consumption for different schemes under different system utilizations. Here, smaller utilizations indicate that more spare CPU capacity is available for energy/reliability management. From the results, we can see that, at moderate system load (i.e., $U = 0.5$), the O-RAPM scheme can save up to 50% more energy compared to that of the C-RAPM scheme. Moreover, when execution times of tasks follow *'normal-0.25'* (i.e., on average, tasks use around only 25% of their WCETs), the normalized expected energy consumption under O-RAPM is almost the same as that of ordinary SPM scheme when $U < 0.5$. Note that SPM is reliability-ignorant and will result in dramatically decreasing in system reliability (i.e., increasing in probability of failure) as shown in Figure 4b.

Here, for ease of presentation, Figure 4b shows the *normalized* probability of failure with the one achieved under no power management (NPM) as the baseline. We can see that, as with the C-RAPM scheme, system reliability can be preserved under the O-RAPM scheme. In addition, when execution times of tasks follow *'normal-0.25'*, more tasks can be managed under O-RAPM, which leads to better system reliability.

Figure 4c further shows the normalized expected energy consumption for tasks when their execution times follow uniform distribution. With tasks being more likely to use their WCETs, the expected energy savings obtained by O-RAPM reduces compared to that of the normal distributions. However, it still can save up to 20% energy compared to that of C-RAPM when $U = 0.5$. The reliability can also be preserved; but specific plots are omitted due to space limitations.

6 Conclusions

Several reliability-aware power management (RAPM) schemes have been studied recently to address the negative effects of DVFS system reliability and to save energy while preserving reliability. However, the existing RAPM schemes are rather *conservative*. In this work, exploiting the probabilistic execution time information of real-time tasks, we studied the *optimistic* RAPM scheme. The main idea is to reserve the slack that is *just enough* for the recovery to achieve the required reliability. Therefore, more slack can be left for energy management with additional energy savings on the average.

We first analyzed the optimal frequency setting for a single task by considering the probability distribution of its execution times. Then, the static optimistic RAPM problem for a set of periodic real-time tasks is formulated and shown to be NP-hard. A novel heuristic algorithm is proposed and evaluated. The results show that the probability distribution of tasks' execution times has great impacts on the performance of the new optimistic RAPM scheme. In general, the optimistic RAPM scheme performs very well on both energy savings and reliability preservation. It achieves energy savings comparable to that of the *ordinary (but reliability ignorant)* power management while maintaining the system reliability as successfully as the conservative RAPM schemes.

References

[1] H. Aydin, V. Devadas, and D. Zhu. System-level energy management for periodic real-time tasks. In *Proc. of The* 27^{th} *IEEE Real-Time Systems Symposium (RTSS)*, Piscataway, NJ, USA, Dec. 2006. IEEE CS Press.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *Proc. of IEEE Real-Time Systems Symposium*, 2001.

[3] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *Proc. of The HICSS Conference*, Jan. 1995.

[4] J.-J. Chen and L. Thiele. Expected system energy consumption minimization in leakage-aware dvs systems. In *Proc. of the Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 315–320, 2008.

[5] V. Degalahal, L. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. Soft errors issues in low-power caches. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 13(10):1157–1166, Oct. 2005.

[6] A. Ejlali, M. T. Schmitz, B. M. Al-Hashimi, S. G. Miremadi, and P. Rosinger. Energy efficient seu-tolerance in dvs-enabled real-time systems through information redundancy. In *Proc. of the Int'l Symposium on Low Power and Electronics and Design (ISLPED)*, 2005.

[7] E. M. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. In *Proc. of Power Aware Computing Systems*, 2002.

[8] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N. S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE Micro*, 24(6):10–20, 2004.

[9] R. Ernst and W. Ye. Embedded program timing analysis based on path clustering and architecture classification. In *Proc. of The Int'l Conference on Computer-Aided Design*, pages 598–604, 1997.

[10] 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.

[11] T. Ishihara and H. Yauura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. of The Int'l Symposium on Low Power Electronics and Design*, 1998.

[12] R. Iyer, D. J. Rossetti, and M. Hsueh. Measurement and modeling of computer reliability as affected by system activity. *ACM Trans. on Computer Systems*, 4(3):214–237, Aug. 1986.

[13] R. Jejurikar and R. Gupta. Dynamic voltage scaling for system wide energy minimization in real-time embedded systems. In *Proc. of the Int'l Symposium on Low Power Electronics and Design (ISLPED)*, pages 78–81, 2004.

[14] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM*, 20(1):46–61, 1973.

[15] J. Lorch and A. Smith. Improving dynamic voltage scaling algorithms with pace. *SIGMETRICS Perform. Eval. Rev.*, 29(1):50–61, 2001.

[16] R. Melhem, D. Mossé, and E. M. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. on Computers*, 53(2):217–231, 2004.

[17] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. of* 18^{th} *ACM Symposium on Operating Systems Principles*, Oct. 2001.

[18] P. Pop, K. Poulsen, V. Izosimov, and P. Eles. Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In *Proc. of the 5th IEEE/ACM Int'l Conference on Hardware/software codesign and System Synthesis (CODES+ISSS)*, pages 233–238, 2007.

[19] D. K. Pradhan. *Fault Tolerance Computing: Theory and Techniques*. Prentice Hall, 1986.

[20] S. Saewong and R. Rajkumar. Practical voltage scaling for fixed-priority rt-systems. In *Proc. of the* 9^{th} *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[21] N. Seifert, D. Moyer, N. Leland, and R. Hokinson. Historical trend in alpha-particle induced soft error rates of the alpha^{TM} microprocessor. In *Proc. of the* 39^{th} *Annual International Reliability Physics Symposium*, 2001.

[22] O. S. Unsal, I. Koren, and C. M. Krishna. Towards energy-aware software-based fault tolerance in real-time systems. In *Proc. of The Int'l Symposium on Low Power Electronics Design*, 2002.

[23] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proc. of The First USENIX Symposium on Operating Systems Design and Implementation*, Nov. 1994.

[24] R. Xu, C. Xi, R. Melhem, and D. Mosse. Practical pace for embedded systems. In *Proc. of the 4th ACM Int'l conference on Embedded software*, pages 54–63, 2004.

[25] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *Proc. of the* 19^th *ACM symposium on Operating systems principles (SOSP)*, pages 149–163, 2003.

[26] Y. Zhang, K. Chakrabarty, and V. Swaminathan. Energy-aware fault tolerance in fixed-priority real-time embedded systems. In *Proc. of Int'l Conference on Computer Aided Design*, Nov. 2003.

[27] B. Zhao, D. Zhu, and H. Aydin. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2008.

[28] D. Zhu. Reliability-aware dynamic energy management in dependable embedded real-time systems. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[29] D. Zhu and H. Aydin. Reliability-aware energy management for periodic real-time tasks. In *Proc. of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2007.

[30] D. Zhu, R. Melhem, and D. Mossé. The effects of energy management on reliability in real-time embedded systems. In *Proc. of the Int'l Conf. on Computer Aidded Design*, 2004.

[31] D. Zhu, X. Qi, and H. Aydin. Priority-monotonic energy management for real-time systems with reliability requirements. In *Proc. of the IEEE International Conference on Computer Design (ICCD)*, 2007.

[32] D. Zhu, X. Qi, and H. Aydin. Energy management for periodic real-time tasks with variable assurance requirements. In *Proc. of the IEEE Int'l Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2008.