# Reliability-Aware Dynamic Voltage Scaling for Energy-Constrained Real-Time Embedded Systems

Baoxian Zhao, Hakan Aydin
*Department of Computer Science*
*George Mason University*
*Fairfax, VA 22030*
*bzhao@gmu.edu, aydin@cs.gmu.edu*

Dakai Zhu
*Department of Computer Science*
*University of Texas at San Antonio*
*San Antonio, TX 78249*
*dzhu@cs.utsa.edu*

***Abstract*— The Dynamic Voltage Scaling (DVS) technique is the basis of numerous state-of-the-art energy management schemes proposed for real-time embedded systems. However, recent research has illustrated the alarmingly negative impact of DVS on task and system reliability. In this paper, we consider the problem of processing frequency assignment to a set of real-time tasks in order to maximize the overall reliability, under given time and energy constraints. First, we formulate the problem as a non-linear optimization problem and show how to obtain the static optimal solution. Then, we propose on-line (dynamic) algorithms that detect early completions and adjust the task frequencies at run-time, to improve overall reliability. Our simulation results indicate that our algorithms perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance.**

## I. Introduction

With the proliferation of battery-powered embedded computing devices, energy management has become critically important with the prospects of extending the battery life, which is typically limited. The *Dynamic Voltage Scaling (DVS)* technique [1], [2] is recognized as the basis of numerous energy management solutions. DVS exploits the fact that the dynamic power consumption is a strictly convex function of the CPU speed, and attempts to save energy by reducing the supply voltage and frequency at run-time. DVS has been well studied in many power management schemes that target real-time embedded systems with timing constraints [3], [4], [5], [6].

On the other hand, recent research [7], [8] has shown that DVS has a significant and negative effect on the *system reliability*, primarily because of significantly increased *transient fault rates* at low supply voltage and frequency levels. Therefore, there is a growing awareness about the need to apply DVS only after careful evaluation, especially for mission-critical real-time embedded applications where both high level of reliability and low energy consumption are important. Nevertheless, at present, there are only a few studies investigating the system reliability and energy efficiency requirements simultaneously [8], [9], [10], [11]. The trade-off between system reliability and energy consumption is first studied by Zhu et al. in [8]. In [8], two fault rate models for DVS settings are also suggested. [9] proposed a reliability-aware power management (RA-PM) scheme that dynamically schedules a recovery job at task dispatch time whenever DVS is applied, preserving the overall (original) reliability. The scheme is further extended to multiple tasks with a common deadline [10] and to periodic real-time tasks [11].

Previous research mostly focused on saving energy as much as possible through DVS while *maintaining original system reliability*, typically by scheduling recovery tasks for potential fault scenarios. Here, we consider a totally different case by addressing the settings where the system has a given *fixed energy budget* for its operation. The system is allowed to consume energy within this allowance. Our objective is to determine the task frequency (speed) assignments in order to maximize the system reliability (i.e. the probability of completing the application successfully) within the given energy budget and timing constraint. Or equivalently, we consider determining optimal energy allocations for each task, while precisely characterizing/quantifying the effects of DVS on the system reliability.

In this research effort, after presenting our system models and assumptions in Section II, we first address and solve the *static* version of the problem, where all tasks execute their worst-case workload (Section III). Then, we extend our framework to dynamic settings and propose three on-line reclaiming algorithms that detect early completions and adjust the task frequencies at run-time, with the objective of improving the application's reliability by making the best use of excess energy at run-time (Section IV). The experimental evaluation (presented in Section V) indicates that our dynamic algorithms perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance.

To the best of our knowledge, this research effort is the first to address the problem of maximizing the overall reliability of a real-time embedded application within the given energy and time constraints.

## II. Models and Problem Description

### A. Power Model

With DVS, the clock frequency is reduced by reducing the supply voltage [12] since there is an almost linear relationship between the supply voltage and operating frequency [13]. In this paper, we focus on the DVS technique and adopt the system-level power model proposed in [8] and

This work is supported by US National Science Foundation grants CNS-0720647, CNS-0720651 and CNS-546244 (CAREER Award)

978-1-4244-2658-4/08/$25.00 ©2008 IEEE

subsequently used in [9], [10]. Hence, the system power consumption $P$ is given by:

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

where $P_s$ is the static power, $P_{ind}$ is the frequency-independent active power and $P_d$ is the frequency-dependent active power. The static power, which may be removed only by powering off the whole system, includes the power to maintain basic circuits, keep the clock running and the memory in sleep modes [14]. $P_{ind}$ is a constant independent of processing frequencies (i.e. the power consumed by off-chip devices such as main memory and external devices) and can be efficiently removed by putting systems into sleep states [13], [15]. $P_d$ is the power mainly consumed by CPU, in addition to any power that depends on the frequencies [13]. $\hbar$ represents system states and indicates whether active powers are currently consumed in the system. Specifically, when the system is active, $\hbar$ = 1; otherwise, the system is in sleep modes or turned off and $\hbar$ = 0. The effective switching capacitance $C_{ef}$ and the dynamic power exponent $m$ (which is, in general, no smaller than 2) are system-dependent constants and $f$ is the processing frequency.

Since there exists an excessive time and energy overhead associated with turning on/off a system [16], we assume that the system is always active. As $P_s$ is not manageable, we will ignore the static power and concentrate on the frequency-independent active power $P_{ind}$ and frequency-dependent active power $P_d$ in our analysis.

Although DVS can reduce energy consumption because of the low frequency-dependent active power $P_d$ at reduced processing frequencies, the application will take more time to complete at low frequencies. As a result, increased total energy may be consumed because of the prolonged device active times (due to the frequency-independent active power $P_{ind}$). Therefore, considering the system-level power, lower frequencies may be not be always best for energy saving and it is shown in [14] that there exists a minimum energy-efficient voltage/frequency pair. In [17], the energy-efficient frequency for our power model is given as:

$$f_{ee} = \sqrt[m]{\frac{P_{ind}}{(m-1)C_{ef}}} \quad (2)$$

Hence, it is not energy-efficient to run any task at a frequency below $f_{ee}$; doing otherwise will result in more energy consumption. Note that if $f_{ee}$ exceeds the maximum available frequency level $f_{max}$, then the system should not reduce its speed below $f_{max}$ [18]. In the following discussion, we use the dynamic power exponent $m$ as 3.

### B. Fault Model

During the execution of an application, a fault may occur due to various reasons, such as hardware failure, software errors and the effect of electromagnetic interference and cosmic ray radiations. Since *transient* faults occur much more frequently than *permanent* faults [19], [20], [21], in this paper, we focus on transient faults, and develop feasible DVS solutions with a given energy budget to maximize overall reliability.

Traditionally, transient faults have been modeled through Poisson distribution with an average arrival rate $\lambda$ [22]. However, considering the effects of voltage scaling on transient faults [7], [8], the average arrival rate $\lambda$ will depend on the system processing frequency and supply voltage. Therefore, the fault rate at frequency $f$ (and its corresponding voltage level) can be *generally* modeled as

$$\lambda(f) = \lambda_0 \cdot g(f) \quad (3)$$

where $\lambda_0$ is the average fault rate corresponding to the maximum frequency $f_{max}$ = 1 (and supply voltage $V_{max}$). That is, $g(f_{max})$ = 1.

In general, transient fault rates are exponentially-related to the circuit's *critical* charge (which is the smallest charge required to cause a soft error in a circuit node) [23]. In our analysis and simulations, we focus on the exponential fault rate model proposed in [8]:

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

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

### C. Application Model

In this work, we consider a real-time application that consists of a set of $n$ independent tasks: $T_1, T_2, ..., T_n$. All tasks in the application should complete their executions by the common deadline $D$. Note that, if the application is periodic, $D$ can also represent the period (or, *frame length*).

The worst-case execution cycles (WCC) of task $T_i$ is denoted by $c_i$. We consider a system with DVS capability where the clock frequency can vary from a minimum available frequency $f_{min}$ to a maximum frequency $f_{max}$ (normalized to 1.0). The execution time of task $T_i$ under the frequency $f_i$ is given by $\frac{c_i}{f_i}$. The *utilization* $U$ of the task set is given as $\sum \frac{c_i}{D \cdot f_{max}} = \sum \frac{c_i}{D}$; in other words, it corresponds to the *load* under maximum frequency.

In our model, each task $T_i$ is allowed to have a different frequency-independent power figure $P_{ind_i}$, since each task may require access to different subsets of external devices. We assume that the system is *energy-constrained* in the sense that it has a fixed energy budget $E$, which is not replenishable during execution and cannot be exceeded in any interval of length $D$.

The *reliability* of a task is defined as the probability of completing the task successfully (i.e. without encountering errors triggered by transient faults) [8], [9], [10]. Assuming that the transient faults follow a Poisson distribution, the reliability of the task $T_i$ with its WCC $c_i$ is $R_i(f_i) = e^{-\lambda(f_i) * \frac{c_i}{f_i}}$ [8], where $f_i$ is its execution frequency and $\lambda(f_i)$ is defined as in (4). The reliability of a real-time system depends on

the correct execution of all tasks in an application [17]. In our application model, which consists of $n$ tasks, the system reliability is therefore, $R = \prod_{i=1}^{n} R_i(f_i)$.

## III. Reliability-Aware DVS: Static Solution

In this section, we formalize and optimally solve the problem of finding task-level frequency assignments to maximize the system reliability, with a given energy budget $E$ during a period $D$. Recall that, the probability of completing the task $T_i$ without a fault (that is, its *reliability*) at the processing frequency $f_i$ is $R_i(f_i) = e^{-\lambda(f_i)*\frac{c_i}{f_i}}$ , where $\lambda(f_i)$ is given by (4). Hence, $R_i(f_i)$ is a strictly concave and increasing function of $f_i$.

The total (i.e. frequency-dependent and frequency-independent) energy consumption of $T_i$ at the frequency $f_i$ can be expressed as [9]:

$$E_i(f_i) = P_{ind_i}\frac{c_i}{f_i} + C_{ef}c_i f_i^2 \tag{5}$$

Notice that $E_i(f_i)$ is a strictly convex function and is minimized when $f_i = f_{ee_i}$ (Section II-A).

Let $\varphi_i(f_i) = \lambda(f_i)\cdot\frac{c_i}{f_i}$. Our problem can be formally stated as *to find* $f_i$ $(1 \le i \le n)$ *values so as to maximize*:

$$R = \prod_{i=1}^{n} R_i = e^{-\sum_{i=1}^{n}\varphi_i(f_i)} \tag{6}$$

Subject to:

$$\sum_{i=1}^{n}\frac{c_i}{f_i} \le D \tag{7}$$

$$\sum_{i=1}^{n} E_i(f_i) \le E \tag{8}$$

$$f_{min} \le f_i \le f_{max}\ (1 \le i \le n) \tag{9}$$

Above, the inequality (7) corresponds to the deadline constraint, while (8) encodes the hard energy constraint. The constraint set (9) gives the range of feasible frequency assignments.

Considering the well-known features of the exponential functions, we can re-express our objective as to *minimize*

$$\sum_{i=1}^{n}\varphi_i(f_i) \tag{10}$$

subject to the constraints (7), (8) and (9). In the rest of the paper, this optimization problem will be called *Energy-Constrained Reliability Management* (ECRM) problem.

Now, let $E_{limit}$ be the minimum energy that must be allocated to the given task system to allow their completion before or at the deadline (period boundary) $D$. Given the task parameters, $E_{limit}$ can be computed by the polynomial-time algorithm developed in [18]. As a by-product, the same algorithm yields also the optimal task-level frequency assignments $(fl_1, fl_2, ..., fl_n)$ when the total energy consumption is *exactly* $E_{limit}$. Obviously, if $E < E_{limit}$, then there is no solution to our problem, since the system would lack the minimum energy needed for timely completion. Also, let $E_{max} = \sum_{i=1}^{n} E_i(f_{max})$ be the energy consumption of the task set when all tasks run at $f_{max}$. As another boundary condition, when $E \ge E_{max}$, executing all tasks at the maximum frequency is the optimal solution (since $R_i(f_i)$ is monotonically increasing with $f_i$ and the system has sufficient energy to run at $f_{max}$ continuously). Therefore, in the remaining of the paper, we will focus exclusively on settings where $E_{limit} \le E < E_{max}$.

*Lemma 1:* In the optimal solution to ECRM, $\forall i\ f_i \ge f_{ee_i}$, where $f_{ee_i} = (\frac{P_{ind_i}}{2C_{ef}})^{\frac{1}{3}}$ is the energy-efficient frequency for $T_i$.

*Proof:* This follows from the observation that executing $T_i$ at a speed lower than $f_{ee_i}$ would result in *more energy* consumption for $T_i$ (Section II.A). As a result, if a task were to execute at a speed lower than $f_{ee_i}$ in the optimal solution, then increasing its speed to $f_{ee_i}$ would actually *decrease* its energy consumption (still satisfying (8)) and *increase* the overall reliability considering the positive impact of higher speeds on task reliability – giving a contradiction. ■

Thanks to Lemma 1, the constraint (9) can be re-written as:

$$f_{low_i} \le f_i \le f_{max}\ (1 \le i \le n) \tag{11}$$

where $f_{low_i}$ is $\max(f_{min}, f_{ee_i})$.

*Lemma 2:* If $E_{limit} \le E < E_{max}$, in the optimal solution to ECRM, the total energy consumption $\sum_{i=1}^{n} E_i(f_i)$ must be exactly equal to $E$.

*Proof:* Assume that the statement is false. Since $E < E_{max}$ by assumption, there must be a speed $f_i < f_{max}$. In this case, it should be possible to increase $f_i$ by $\varepsilon > 0$ such that $(f_i + \varepsilon) \le f_{max}$ and $\sum_{j\neq i} E_j(f_j) + E_i(f_i + \varepsilon) \le E$. It is clear that the deadline and energy constraints are still satisfied after this modification. Further, the overall system reliability has obviously improved, due to the execution of $T_i$ at a speed higher than $f_i$. Thus, the proposed solution cannot be optimal and we reach a contradiction. ■

Lemma 2 allows us to conclude that if $E_{limit} \le E < E_{max}$, then we can re-write the constraint (8) as an equality.

$$\sum_{i=1}^{n} E_i(f_i) = E \tag{12}$$

Consequently, we obtain a new non-linear (convex) optimization problem ECRM', defined as *to find* $f_i$ $(1 \le i \le n)$ *values so as to minimize*:

$$\sum_{i=1}^{n}\varphi_i(f_i) \tag{13}$$

Subject to:

$$\sum_{i=1}^{n}\frac{c_i}{f_i} \le D \tag{14}$$

$$\sum_{i=1}^{n} E_i(f_i) = E \tag{15}$$

$$f_{low_i} \le f_i \le f_{max}\ (1 \le i \le n) \tag{16}$$

The problem ECRM' can be solved, for instance, by *Quasi-Newton* techniques developed for constrained non-linear optimization [24]. The technique exploits the well-known Kuhn-Tucker optimality conditions for non-linear

programs [25] in an iterative fashion by transforming the original problem to a quadratic programming problem [24]. A theoretical complication with this approach is that it is practically impossible to *express* the maximum number of iterations as a function of the number of *unknowns* which, in this case, corresponds to the number of tasks $n$. However, in our experience, the algorithm is rather fast: for instance on a 1 GHz CPU with 1 GB memory, our implementation was able to return the optimal solution in less than 1.8 seconds even for task sets with 1000 tasks.

However, we also developed a heuristic algorithm that runs in polynomial-time. This algorithm, named ECRM-LU, satisfies the deadline, energy and frequency range constraints. Further, it yields solutions that are extremely close to the optimal solution. ECRM-LU proceeds as follows. We temporarily ignore the deadline constraint (14) and solve the problem ECRM' only by considering the energy constraint (15) and frequency range constraints (16). Notice that, by excluding the deadline constraint, the problem is transformed to a separable convex optimization problem with $n$ unknowns, $2n$ inequality constraints and a single equality constraint. This problem, in turn, can be solved in time $O(n^3)$ through iteratively manipulating the Kuhn-Tucker optimality conditions in a way similar to the technique illustrated in algorithm given in [18]. Now, if this solution satisfies also the deadline constraint (14), obviously, it is also the solution to ECRM'. Otherwise, we re-write the constraint set (16) as:

$$fl_i \leq f_i \leq f_{max} \ \ (1 \leq i \leq n) \tag{17}$$

where $fl_i$ is the frequency assignment to task $T_i$ in the solution where all tasks complete *at exactly D* and with energy allocation $E_{limit}$. Recall that $\{fl_i\}$ values can be also computed in time $O(n^3)$. By enforcing the constraint set (17), we make sure that the final speed assignments satisfy also the deadline constraint. Once again, this version of the problem where the deadline constraint is handled implicitly by enforcing the lower bounds on frequency assignments can be solved in time $O(n^3)$. Hence, the overall time complexity of ECRM-LU is also $O(n^3)$. Interestingly, our extensive simulation studies show that ECRM-LU performs very well compared to the optimal solution through the almost entire spectrum: the reliability figures yielded by ECRM-LU are close to the optimal one by a margin of 0.03%, when $\frac{E}{E_{limit}} \geq 1.02$. In a tiny portion of the interval where $1.0 \leq \frac{E}{E_{limit}} < 1.02$, we observed a difference of at most 1%.

## IV. Dynamic Reliability-Aware Scheduling

The static solution presented in the previous section is optimal under the assumption that all tasks will present their worst-case workload (i.e. their WCCs). While provisioning for worst-case scenarios is imperative in real-time systems, in practice, many real-time tasks complete early without consuming their WCCs. In fact, numerous DVS studies published in recent past were based on detecting and reclaiming unused CPU time (i.e. the dynamic *slack*) to enhance energy savings by reducing the processing frequency at run-time [3], [26], [27], [5]. A similar opportunity exists here: the *excess energy* that arises from early completions of tasks, can be used to *increase* the speeds of tasks at run-time, to improve the system reliability. Clearly, utmost care must be exercised to make sure that the system remain within its energy allowance (budget), before making such adjustments.

In this section, we develop on-line (dynamic) reliability-aware schemes for reclaiming the excess energy at run-time. In the following algorithms, we assume that $n$ tasks in the real-time embedded application are executed in the order $T_1, T_2, ..., T_n$. The three dynamic algorithms that we developed are:

- *BR*: dynamic *basic reclaiming* algorithm. In this solution, an initial speed assignment is made by solving the static problem presented in the preceding section, assuming worst-case workload for each task. However, at task completion points, the excess energy that may be available (due to early completions) is effectively re-cycled within the system: a new speed (frequency) assignment is made for the *remaining* tasks by considering the remaining (updated) energy budget and time to deadline. This new assignment is again obtained by invoking our optimal solution to the problem ECRM.
- *GRE*: dynamic *greedy* algorithm. Although *BR* satisfies the energy and deadline constraints, it is pessimistic in the sense that it assumes WCCs for all tasks when re-distributing the excess energy. An alternative approach may be to allocate the excess energy *entirely* to the next task[1] $T_{next}$ at task completion points, relying on the fact that $T_{next}$ is also likely to complete early and release excess energy for the use of the remaining tasks. In the mean time, the reliability of $T_{next}$ will be significantly improved due to execution at higher speeds. *GRE* still preserves feasibility in terms of timing and energy constraints: compared to the initial static solution obtained by ECRM, the task speeds never decrease (guaranteeing the timely completion) and only the excess energy that is obtained at run-time is re-assigned.
- *AGR*: dynamic *aggressive* algorithm. This scheme represents the most speculative solution, in the sense that it counts on *probable* early completions before execution, and makes speed assignments accordingly. The main idea is to *aggressively give sufficient energy to the current task by still leaving minimum required energy for the tasks to follow, based on their WCCs*. It is speculative, because under a worst-case scenario (where most of the tasks present high workload), many tasks in the chain would be forced to execute at low speeds to guarantee the completion within the energy budget, significantly lowering the overall reliability. However, in settings where the actual workload is likely to deviate from the worst-case with high probability, this strategy

[1]Note that if the next task cannot be assigned the entire excess energy due to the maximum frequency limitations, then the following task(s) will be able to reclaim energy at the next task completion points.

will (and, as we show in the performance evaluation section, *does*) pay off. Specifically, *AGR* is implemented through the following steps: First, $E_{limit}$ (which is defined as the minimum energy needed to complete all the tasks by their deadline (Section III)) is computed. Next, a speed assignment $(fl_1, fl_2, \ldots, fl_n)$ based on WCCs of all tasks but with energy allocation equal to $E_{limit}$ is computed. $T_2, \ldots, T_n$ are *tentatively* assigned the frequencies $(fl_2, \ldots, fl_n)$ and their energy consumption with these assignments and WCCs ($E_{reserve}$) are evaluated. Then, the entire remaining energy (i.e. $E - E_{reserve}$) is allocated to the first task $T_1$, allowing it to execute as fast as possible within the given constraints. At task completion points, the above steps are repeated by considering the early completions and *actual* remaining energy for remaining tasks. The fact that this solution preserves the energy and deadline constraints follows from the properties of the speed assignments that corresponds to $E_{limit}$, which corresponds to the minimum energy needed for a feasible solution.

## V. Simulation Results and Discussion

To evaluate the performance of our dynamic algorithms under varying workload conditions, we designed a discrete-event simulator in C. In our simulator, we implemented five schemes:

- The *static* scheme which computes the processing frequencies using the optimal solution to problem ECRM assuming the worst-case workload for each task. *Static* does not use any on-line component in the sense that no dynamic speed adjustment is performed, regardless of the actual workload.
- The *basic reclaiming scheme (BR)*, that re-allocates unused energy at task completion points, by re-invoking the algorithm to optimally solve the problem ECRM for the remaining tasks.
- The *greedy reclaiming scheme (GRE)*, that re-allocates unused energy entirely to the next task whenever possible, at task completion points.
- The *aggressive scheme (AGR)*, that aggressively allocates the maximum possible amount of energy for the current task, while still leaving a minimal energy for the remaining tasks to allow timely completion.
- The *clairvoyant scheme (Bound)*, that knows the *actual* workload of each task in advance and computes the optimal speed assignments to maximize the reliability by solving the problem ECRM accordingly. Obviously, *Bound* is not a practical scheme (since it assumes the knowledge of the actual workload in advance); however, it characterizes the upper bound on the performance of *any* static or dynamic algorithm.

We considered task sets with 8 tasks, with the frame/period length of $D = 1000$. The total *utilization (U)* of each task set under maximum frequency is varied from 0.2 to 1.0 (full load). The worst-case number of cycles *(WCC)* of each task is randomly generated. To model the variations in the actual workload, we use the ratio $\frac{ACC}{WCC}$, which denotes the ratio of the *average-case number of cycles (ACC)* to the worst-case number of cycles. The lower this ratio, the more the actual workload deviates from the worst-case. For each task set and utilization value, $\frac{ACC}{WCC}$ is changed from 0.2 to 1.0. The actual number of cycles of each task is generated randomly, using normal distribution with mean *ACC*. For each point in the graphs, 1000 task sets are used and the results that are shown are the average of all runs.

We model a DVS-enabled CPU where the normalized processing frequencies can change from $f_{min} = 0.1$ to $f_{max} = 1.0$. We use a cubic frequency-dependent power component $P_d$ which is equal to unity at $f_{max}$. The frequency-independent power component $P_{ind}$ for each task is normalized with respect to $P_d$ and is generated according to uniform distribution in the range $[0,2]$. To analyze the impact of system's energy budget $E$ on the performance, we varied $E$ from $E_{limit}$ (minimum energy needed to meet the deadline, see Section III) to $E_{max}$ (energy consumption at $f_{max}$). The ratio $\frac{E}{E_{limit}}$ shown in the plots, is a measure of the available energy in the system; for example, when $\frac{E}{E_{limit}} = 1.2$ the system has 20% more energy than the minimum needed to meet the deadline. We assume that the transient faults' occurrence is determined by Poisson distribution and given by Equation (4), where $\lambda_0 = 10^{-9}$ and $d = 3$.

Fig. 1 shows the relationship between the probability of failure when $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$. As expected, the probability of failure (defined as $1 - reliability$) generally decreases with increasing energy budget ($\frac{E}{E_{limit}}$ ratio), since more energy enables the system to use higher processing speeds with improved reliability. The clairvoyant *Bound* scheme achieves a constant probability of failure, because even when $E/E_{limit} = 1$, all tasks can be executed at $f_{max}$ thanks to a priori knowledge of actual execution times, which are, on the average, half of the worst-case in these experiments – since processing speeds beyond $f_{max}$ are not available, giving more energy to *Bound* does not further help. We observe that, among the other schemes, the static scheme (which does not perform any dynamic energy reclamation) performs worst and *AGR* is the best, with very close performance to *Bound* when ($\frac{E}{E_{limit}} > 1.2$). This result indicates that aggressively giving maximum energy to the tasks that will execute early typically pays off in these settings, since these tasks are also likely to generate excess energy due to early completions, which can be allocated to the later tasks. *BR* is a little worse than *GRE*; but both perform consistently better than *Static*, verifying the benefits of dynamic reclaiming. Observe that once the available energy is 30% or more compared to $E_{limit}$, all dynamic schemes perform almost the same.

Fig. 2 illustrates how the probability of failure changes as a function of task utilization $U$, for $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$. Again, the relative ordering of the schemes remains the same. However, we observe an interesting phenomenon: as we increase the utilization towards the range of $0.5 - 0.6$, the probability of failure first increases. After that threshold, it starts to decrease. This pattern can be explained as follows:

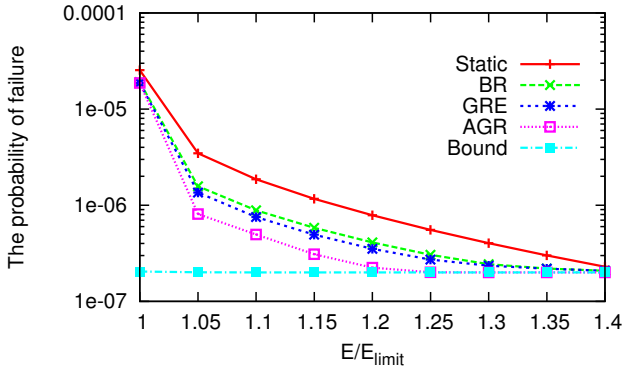


Fig. 1. The probability of failure vs. $E/E_{limit}$ with $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$

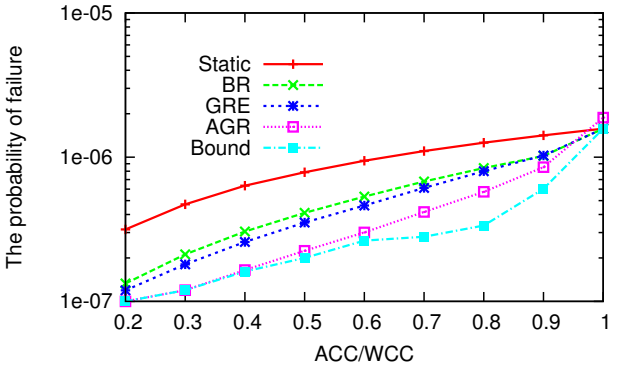


Fig. 3. The probability of failure vs. the $\frac{ACC}{WCC}$ ratio with $U = 0.4$ and $E/E_{limit} = 1.2$

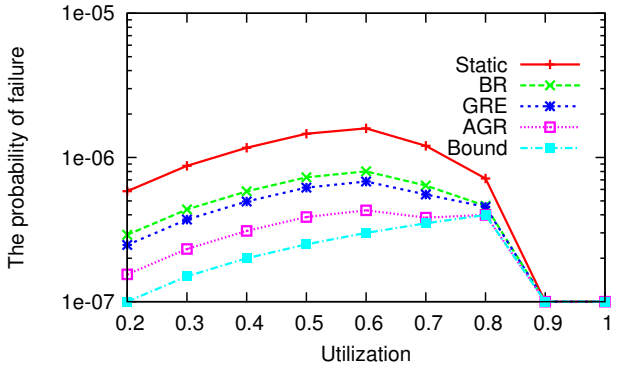


Fig. 2. The probability of failure vs. utilization($U$) with $\frac{ACC}{WCC} = 0.5$ and $E/E_{limit} = 1.15$

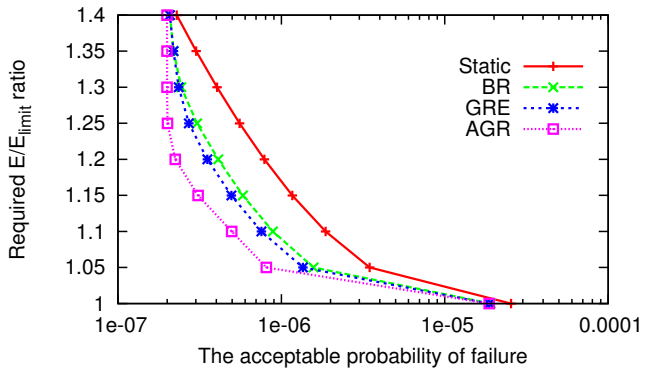


Fig. 4. The acceptable probability of failure vs. the required $E/E_{limit}$ ratio

increasing the utilization has direct effects on two important factors on system reliability, but in *reverse* directions: first, increasing the utilization results in increased execution times, which means increased probability of being subject to transient faults (under comparable fault rates). In fact, as $U$ approaches 0.5, this factor dominates and the probability of failure increases. However, higher utilization values force the system to adopt higher frequencies in order to meet the deadline and the positive impact of this on reducing the fault rates becomes the primary factor after a certain point. In fact, after $U = 0.8$, all tasks are executed at speeds close to $f_{max}$ and then the probability of failure drops sharply to a minimal value.

Fig. 3 shows the impact of the variability in the actual workload (i.e. the $\frac{ACC}{WCC}$ ratio) on the probability of failure, with $E/E_{limit} = 1.2$ and $U = 0.4$. In general, we find that the probability of failure increases with the increased ratio of $\frac{ACC}{WCC}$. This is to be expected, because with the increased ratio of $\frac{ACC}{WCC}$, at run-time, tasks execute longer and they are subject to transient faults with higher probabilities. However, observe that the dynamic schemes are able to significantly reduce the probability of failure compared to *Static* thanks to online reclaiming features, especially at low $\frac{ACC}{WCC}$ ratios. When $\frac{ACC}{WCC} = 1.0$, the probabilities of failure of *Static*, *BR* and *GRE* converge to that of *Bound*, since there are no early completions or excess energy at run-time. But, it is interesting to note that the probability of failure of *AGR* is slightly higher at that point. This is because, when all tasks take their WCC, the expected early completions do not occur, while the aggressive nature of *AGR* still forces the later tasks to execute at relatively low speeds, causing a loss in reliability.

These patterns can be also used to *establish guidelines* for system designers who need to figure out the *minimum amount of energy supply* that must be provided to the system, to achieve a certain target probability of failure. Fig. 4 establishes these thresholds for $\frac{ACC}{WCC} = 0.5$ and $U = 0.4$. For example, the figure suggests that 15% additional energy (beyond $E_{limit}$) must be provided to the system to achieve a probability of failure of $10^{-6}$, if *Static* is the scheme to be used. However, if dynamic schemes (e.g. *BR* or *GRE*) are available, then 7% additional energy is sufficient. It is also worthwhile to note that the difference between the schemes becomes very important at *low* values for the target probability of failure (e.g. smaller than $10^{-6}$); indicating that in safety- or time-critical applications, the available energy budget may be a prime factor for the achievable reliability.

## VI. Conclusions

In this paper, we considered a real-time application consisting of multiple tasks and showed how to compute energy allocations (which translate to frequency assignments) to maximize overall reliability, while considering a hard energy constraint. Both static optimal and on-line (dynamic)

schemes are developed in this paper. Our simulation results indicate that our algorithms perform comparably to a clairvoyant optimal scheduler that knows the exact workload in advance. To the best of our knowledge, this problem was not addressed in the research literature in the past.

In the past DVS research, it has been shown [28] that executing tasks in different orders may give rise to different energy savings, because of the different workload variability exhibited by tasks. In a similar vein, it is likely that the execution order will also have an impact on overall reliability, in dynamic settings. We will consider this as a future research direction.

## REFERENCES


[1] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced cpu energy," *Proceedings of the 1st USENIX conference on Operating Systems Design and Implementation (OSDI'94)*, 1994.

[2] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, 1995.

[3] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Dynamic and aggressive power-aware scheduling techniques for real-time systems," *Proceedings of the 22th IEEE Real-Time Systems Symposium (RTSS'01)*, 2001.

[4] D. Mossé, H. Aydin, B. R. Childers, and R. Melhem, "Compiler-assisted dynamic power-aware scheduling for real-time applications," *Proceedings of Workshop on Compiler and OS for Low Power*, 2000.

[5] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for lowpower embedded operating systems," *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001.

[6] S. Saewong and R. Rajkumar, "Practical voltage scaling for fixed priority rt-systems," *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'03)*, 2003.

[7] 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*, vol. 24, no. 6, pp. 10–20, 2004.

[8] D. Zhu, R. Melhem, and D. Mossé, "The effects of energy management on reliability in real-time embedded systems," *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'04)*, 2004.

[9] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, 2006.

[10] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD'06)*, 2006.

[11] ——, "Reliability-aware energy management for periodic real-time tasks," *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'07)*, 2007.

[12] T. Pering, T. Burd, and R. Brodersen, "The simulation and evaluation of dynamic power-aware scheduling for real-time applications," *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPLED'00)*, 2000.

[13] T. Burd and R. Brodersen, "Energy efficient cmos microprocessor design," *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95)*, 1995.

[14] X. Fan, C. Ellis, and A. Lebeck, "The synergy between power-aware memory systems and processor voltage," *Proceedings of Workshop on Power-Aware Computer Systems (PACS'03)*, 2003.

[15] "Mobile pentium iii processor-m datasheet," Order Number: 298340-002,Oct 2001.

[16] E. M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," *Proceedings of Workshop on Power-Aware Computer Systems (PACS'02)*, 2002.

[17] D. Zhu, R. Melhem, D. Mossé, and E. Elnozahy, "Analysis of an energy efficient optimistic tmr scheme," *Proccedings of International Conference on Parallel and Distributed Systems (ICPADS'04)*, 2004.

[18] H. Aydin, V. Devadas, and D. Zhu, "System-level energy management for periodic real-time tasks," *Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06)*, 2006.

[19] X. Castillo, S. McConnel, and D. Siewiorek, "Derivation and calibration of a transient error reliability model." *IEEE Trans. on Computers*, vol. 31, no. 7, pp. 658–671, 1982.

[20] R. K. Iyer and D. J. Rossetti, "A measurement-based model for workload dependence of cpu errors," *IEEE Trans. on Computers*, vol. 33, pp. 518–528, 1984.

[21] R. K. Iyer, D. J. Rossetti, and M. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Trans. on Computer Systems*, vol. 4, no. 3, pp. 214–237, Aug. 1986.

[22] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," *Proceedings of Design, Automation and Test in Europe (DATE'03)*, 2003.

[23] P. Hazucha and C. Svensson, "Impact of cmos technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. on Nuclear Science*, vol. 47, no. 6, pp. 2586–2594, 2000.

[24] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, "Nonlinear programming: Theory and algorithms (third edition)," *A John Wiley and Sons, INC.*, pp. 576–585, 2005.

[25] D. Luenberger, "Linear and nonlinear programming," *Addison- Wesley, Reading Massachusetts*, 1984.

[26] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez, "Power-aware scheduling for periodic real-time tasks," *IEEE Transactions on Computers*, vol. 53, no. 10, pp. 584–600, 2004.

[27] K. Choi, R. Soma, and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," *Proceedings of Design, Automation and Test in Europe (DATE'04)*, 2004.

[28] F. Gruian and K. Kuchcinski, "Uncertainty-based scheduling: energy-efficient ordering for tasks with variable execution time," *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED'03)*, 2003.