Scheduling for Energy Efficiency and Fault Tolerance in Hard Real-Time Systems

Yu Liu*, Han Liang§ and Kaijie Wu*
*ECE, University of Illinois at Chicago
§Digital Media Group, Trident Microsystems, Inc.

Abstract—This paper studies the dilemma between fault tolerance and energy efficiency in frame-based real-time systems. Given a set of K tasks to be executed on a system that supports L voltage levels, the proposed heuristic-based scheduling technique minimizes the energy consumption of tasks execution when faults are absent, and preserves feasibility under the worst case of fault occurrences. The proposed technique first finds out the optimal solution in a comparable system that supports continuous voltage scaling, then converts the solution to the original system. The runtime complexity is only (LK^2). Experimental results show that the proposed approach produces near-optimal results in polynomial time.

I. MOTIVATION

Energy efficiency is crucial to many real-time systems due to their limited energy supply and severe thermal constraints of the operating environment. Among the power-management techniques proposed to tackle these challenges, Dynamic Voltage and Frequency Scaling (DVFS) has emerged as the most popular and widely deployed scheme [1][2]. DVFS dynamically adjusts the supply voltage of CMOS circuits to save power at the cost of extended circuit delays. On the other hand, aggressive technology scaling is making devices more susceptible to radiation-induced bit-flips—neutrons at the sea level are able to introduce faults in electronic systems [3][4]. For systems like surveillance, satellite, and life-support implanted devices that require both fault tolerance and energy efficiency, there is a lack of efficient solutions. Simply applying fault recovery techniques and energy minimization techniques one after the other only results in inferior quality. This is because minimizing energy first may not leave enough slack for fault recovery, and minimizing energy after fault recovery reservation treats normal task executions and re-executions (for fault recovery) equally, which is equivalent to optimizing the worst case that happens rarely.

In this paper we propose to study the trade-off between fault tolerance and energy efficiency of fault-prone real-time systems. Given a fault tolerance constraint and a set of tasks, we are interested in finding the execution order and the DVFS policy of tasks that lead to the maximum energy saving in the absence of faults and preserve feasibility even under the worst case of fault occurrences.

II. BACKGROUND

A. The frame-based real-time system

The system model studied in this work is referred to as a frame-based system where all tasks are released at time 0 of a time frame and should be finished before the end of the frame, as described in a recent survey [7]. Tasks could have dependences between them. An example of such system is the industrial application that involves collecting data points from thousands of sensors at several MHz rate, processing them (communication, encoding or decoding, CRC check, data fusion, and etc) and then feeding the result into a decision-machine for further analysis or visualization. The read-process-analyze cycle time is determined by the rate of incoming data-points, and strict task timelines need to be followed to ensure that no data-point is missed in any cycle. The control paradigm of such systems is categorized as *Time-Triggered* architecture. As opposed to *Event-Triggered* architecture where tasks are released independently according to their own periods and are executed according to priority, *Time-Triggered* architecture is more preferred in safety-critical systems due to its characteristics of synchronization, better hard real-time performance and easier incorporation of fault tolerance [8][9].

This paper studies uni-processor real-time systems with DVFS capability. Considering the nature of real-time workload and the huge time and energy overhead of turning on/off a system, we assume that systems will stay on after finishing the workload of the current frame until the start of the next frame. This study focuses on minimizing the energy consumption of processors.

B. Power and energy consumption of a processor

Since supply voltage is a function of operating frequency, the power consumption of an active processor can be determined by its operating frequency, as denoted by $P_A(f)$, and $P_A(f) = P_d(f) + P_{ind}$, where $P_d(f)$ and P_{ind} are frequency-dependent power and frequency- independent power respectively [12]. The energy consumed by executing a task with N_i CPU cycles at frequency f_i can then be computed as $P_A(f_i) \times N_i/f_i$. [1] The total active energy consumed by executing K tasks is then given below as E_{Active}. After finishing all scheduled tasks, the processor is assumed to go into idle/sleep state with the minimal static power ($P_{Idle} \geq 0$) only. The idle energy is shown below as E_{idle} where D is the duration of the frame.

$$E_{Active} = \sum_{i=1}^{K} [P_A(f_i) N_i / f_i], \; E_{Idle} = P_{Idle}(D - \sum_{i=1}^{K} N_i / f_i)$$

Therefore, the total energy consumed by executing K tasks in a time frame with duration D is:

$$E_{Total} = E_{Active} + E_{Idle} = \sum_{i=1}^{K} [P_d(f_i) + P_{ind} - P_{Idle}] \frac{N_i}{f_i} + P_{Idle} D$$

[1] We assume system voltage/frequency changes only at the start of a task and will stay until the end of the task.

978-3-9810801-6-2/DATE10 © 2010 EDAA

Let $E(f_i) = (P_d(f_i) + P_{ind} - P_{Idle})/f_i$. $E(f_i)$ represents the energy consumption per clock cycle, E_{Total} can be written as:

$$E_{Total} = \sum_{i=1}^{K} E(f_i) N_i + P_{Idle} D \qquad \textbf{Eq. 1}$$

$P_d(f)$ is believed to be a strictly increasing convex function of f, represented by a polynomial with an order between 2 and 3 [5]. Since P_{Idle} is the minimal static power consumed when the processor is idle, it is believed that $P_{Idle} \le P_{ind}$. Therefore $E(f)$ is a convex function and the minimum system energy is achieved when f takes the so-called *critical frequency*, f_c [13]. In this study, we assume $f_{\min} \ge f_c$.

C. The high level strategy

While radiation-induced faults could be transient or permanent, it is reported that transient faults are the most common ones – three to five orders higher than permanent faults [4]. This paper focuses on transient faults, and assumes that transient faults are detected by concurrent error detection mechanisms built-in the systems, such as watchdogs [18], flow signatures checking [19], SRT architecture [20], etc. Fault detection latency is assumed to be included in tasks' worst case execution time. Upon detecting faults, the system could roll back to the latest checkpoint or just re-execute the faulty task. This study considers re-execution-based recovery schemes in general, and passive schemes in particular.

The workload of re-executions cannot be predicated as fault occurrences are accidental. Since the optimal DVFS must be determined based on a fixed workload, the uncertainty of total workload leaves us no choice but to optimize the most possible case which is zero fault occurrence in a frame. Therefore the high level strategy used in this work is to **minimize the energy consumption in the absence of faults and preserve the feasibility under the worst case**. Following this strategy, all re-executions will be budgeted on $f_{\max}$, and all the remaining schedule slack is devoted to slowing down normal task executions. A similar strategy is applied to checkpointing-based systems in [10]. The strategy maximizes energy saving when faults are absent and guarantees feasibility even if the worst case occurs.

D. Related work

Several energy-efficient fault tolerance techniques have been proposed for both checkpointing based systems and re-execution based systems. For the former, the major consideration is to adjust the intervals between checkpoints to minimize energy consumption while preserving feasibility. For the latter, one of the earliest works is on active-replica systems [14]. In active-replica systems, each task has a replica allocated to another processor and both copies are executed simultaneously. Unsal et al. suggested postponing replicas as late as possible so that if the primary copy succeeds, the replica can be aborted to save energy [14]. Recently, many works on passive-re-execution systems are proposed. In passive-re-execution systems, re-executions are performed only when faults are detected. In this paper, we consider the same system/recovery models. In 2004, Zhu et al. proposed an exponential fault rate model to capture the impact of DVFS on the rate of transient faults and showed that energy management through DVFS could reduce system reliability [5]. Based on this model they proposed to schedule an additional recovery task to recuperate the reliability loss due to DVFS for a single task model [15] and uni-core multiple tasks model [12].

Conventionally, re-execution based fault recovery schemes need to know the number of faults to tolerate in a frame. This paper focuses on the core problem of scheduling subject to a conventional fault tolerance constraint, as given below.

E. The optimization problem and contribution of this work

Given a uni-processor real-time system that supports a set of L frequency levels and the number of faults X to tolerate in a frame, the goal is to find the task-to-task execution order and the DVFS policy that lead to the minimal energy consumption when faults are absent and preserve feasibility when up-to X faults are present.

The optimization problem is at least NP-Hard since one of its sub-problem, finding the energy-optimal DVFS policy of a task set in a system supporting discrete scaling, is in fact a Multi-Choice Knapsack problem. The contribution of this work lies in two parts. First, we study a comparable "ideal" system that supports continuous scaling, and find the optimal solution in polynomial time. It is then converted to the discrete system using a simple heuristic. Secondly, we study the impact of execution order of tasks on energy efficiency and show that scheduling longer tasks earlier minimizes energy. Experimental results show that the average energy difference between the result obtained using the proposed approach and the optimal result is less than 1%.

III. CONTINUOUS SYSTEMS: THE OPTIMAL DVFS POLICY FOR A GIVEN EXECUTION ORDER

A. The basic concepts

The focus of this section is to derive the optimal DVFS policy for a given task-to-task execution order. Let us name the i^{th} task T_i, and its frequency f_i, and $f_{\min} \le f_i \le f_{\max}$. Formally, given a set of K tasks, $TSet = \{T_i, 1 \le i \le K\}$, we will find a policy, $Y = \{f_i \in [f_{\min}, f_{\max}], 1 \le i \le K\}$, that minimizes energy consumption when faults are absent, and maintains feasibility when up-to X faults occur.

Feasibility is guaranteed if and only if at any time instance there is always enough time to handle the worst case where the longest unfinished task incurs all the X faults. We therefore determine for each task T_i its deadline D_i and slack S_i as the following:

$$D_i = D - [\sum_{j=i+1}^{K} \frac{N_j}{f_{\max}} + X \frac{R_i}{f_{\max}}], 1 \le i \le K \qquad \textbf{Eq. 2}$$

$$S_i = D_i - \sum_{j=1}^{i} \frac{N_j}{f_j}, 1 \le i \le K \qquad \textbf{Eq. 3}$$

where $R_i = \max(N_i, N_{i+1} \ldots N_K)$, $1 \le i \le K$, and D is the duration of the time frame. $X \cdot R_i / f_{\max}$ is the maximum fault recovery time needed when T_i is being executed. S_i is determined by the assigned frequencies $f_1 \ldots f_i$. Therefore it is also denoted as $S_i(f_1, f_2, \ldots, f_i)$ when necessary. It is easy to see that **the sufficient and necessary condition for a policy Y to be feasible is for $\forall T_i \in TSet, S_i \ge 0$.**

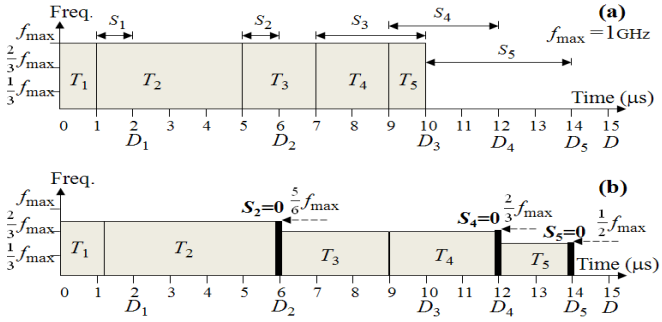

Figure 1. Example with $f_{\min} = 1/3\, f_{\max}$ and $X = 1$

An example is used to demonstrate the meanings of these notations. Consider a task set comprising of 5 tasks that are named according to the execution order: T_1, T_2, T_3, T_4, and T_5, and require 1k, 4k, 2k, 2k, 1k CPU cycles (N_i), respectively, as shown in **Figure**

1 (a). The processor has f_{max} = 1GHz and can scale down to f_{min} = 300MHz continuously. The system is supposed to tolerate at most 1 fault in a time frame. In **Figure 1**, a task is represented by a bar where the height (y-axis) is its frequency and the length (x-axis) is its execution time. The deadline of the task set, D, is set to 15 µs. The deadline of each task, D_i, can then be calculated as D_1 = 2µs, D_2 = 6µs, D_3 = 10µs, D_4 = 12µs and D_5 = 14µs. In **Figure 1**(a), all tasks are executed on f_{max}, and such a policy is feasible since all tasks finish before their corresponding deadlines, i.e. $S_i > 0$ for $1 \le i \le 5$. The goal is to find the optimal DVFS policy of the task set so that the energy is minimized without letting $S_i < 0$, for $1 \le i \le 5$.

We propose the following iterative procedure to find the optimal DVFS policy. The proof of optimality follows in the next subsection. I is set to 0 at the beginning:

1) Reduce the frequencies of all tasks scheduled AFTER the I^{th} task simultaneously until either f_{min} is reached or the slack of any task decreases to 0. If it is the former case, the procedure returns with a complete policy. Otherwise, go to 2).

2) Update I using the index of the task that runs out slack. If more than one tasks run out slack at the same time, I is set to the last one. Go to 1).

In the running example, T_2 is the first task that runs out slack, followed by T_4, then T_5. The complete policy is Y = {f_1=833MHz, f_2=833MHz, f_3=667MHz, f_4=667MHz, f_5= 500MHz}, as given in **Figure 1**(b).

B. Proof of optimality

From the previous subsection, it can be seen that the frequencies produced by the iterative procedure always have $f_i \ge f_{i+1}$. Lemma 1 and Theorem 1 show that this is a necessary condition for an energy-optimal DVFS policy.

Lemma 1: For *TSet* = {T_1, T_2}, if a policy $Y=(f_1, f_2)$ is feasible and energy-optimal, then $f_1 \ge f_2$.

Proof: The slack constraints and deadlines of the tasks are:

$$S_1(f_1) = D_1 - \frac{N_1}{f_1} \ge 0, \quad D_1 = D - \left(\frac{N_2}{f_{max}} + \frac{XR_1}{f_{max}} \right),$$

$$S_2(f_1, f_2) = D_2 - \left(\frac{N_1}{f_1} + \frac{N_2}{f_2} \right) \ge 0, \quad D_2 = D - \frac{XR_2}{f_{max}},$$

where $R_1 = \max(N_1, N_2)$ and $R_2 = N_2$. We divide the proof into two cases according to the value of $S_2(f_1, f_2)$.

Case 1: $S_2(f_1, f_2) = D_2 - (\frac{N_1}{f_1} + \frac{N_2}{f_2}) > 0$

$S_2 > 0$ indicates that not all slack has been devoted to decreasing the frequency of T_2. An energy optimal policy will not have $S_2 > 0$ unless $f_2 = f_{min}$. Therefore, $f_2 = f_{min} \le f_1$.

Case 2: $S_2(f_1, f_2) = D_2 - (\frac{N_1}{f_1} + \frac{N_2}{f_2}) = 0$

$S_2 = 0$ implies that whatever values taken by f_1 and f_2, the total execution time of T_1 and T_2, which is $N_1/f_1 + N_2/f_2$, shall remain unchanged as D_2. Further, since $E_{Total} = E(f_1)N_1 + E(f_2)N_2 + P_{Idle}D$ (Eq 1) and $E(f)$ is a convex function, E_{Total} will be minimized when $f_1 = f_2 = (N_1+N_2)/D_2 = \Delta_{S2}$. The proof is similar to Lemma 2 in [11]. Constraint $S_1 \ge 0$ sets the lower bound of f_1 as:

$$S_1(f_1) = D_1 - \frac{N_1}{f_1} \ge 0 \rightarrow f_1 \ge \frac{N_1}{D_1} = \Delta_{S1}$$

If $\Delta_{S1} \le \Delta_{S2}$, letting $f_1 = f_2 = \Delta_{S2}$ satisfies $S_1 \ge 0$, preserves the total execution time, and minimizes E_{Total}.

If $\Delta_{S1} > \Delta_{S2}$, since f_1 needs to be raised to at least Δ_{S1}, f_2 will be reduced below Δ_{S2} to preserve the total execution time. Therefore $f_1 \ge \Delta_{S1} > \Delta_{S2} \ge f_2$ in the energy optimal solution. Summarizing both cases proves Lemma 1. **Q.E.D**.

Theorem 1 extends the study to K tasks.

Theorem 1: For *TSet* = {T_i, $1 \le i \le K$}, if a policy $Y=(f_i, 1 \le i \le K)$ is feasible and energy-optimal, then $f_i \ge f_{i+1}$.

Proof: We randomly pick a task, T_i and the task scheduled right after it, T_{i+1}. By considering T_i as the first task and T_{i+1} as the second task, this theorem is proved by Case 2 of Lemma 1, that is, finding the optimal policy for T_i and T_{i+1} so that $S_i \ge 0$ and the total execution time of T_i and T_{i+1} remains unchanged. Lemma 1 tells that the necessary condition for an optimal policy is to have $f_i \ge f_{i+1}$. This proves Theorem 1. **Q.E.D**.

Task T_i is called a breakpoint (**BP**) task if $f_i > f_{i+1}$. In the example given in **Figure 1**(b), T_2 and T_4 are BP tasks. Those BP tasks are of special interest to us since they define the complete policy of all tasks. To find the BP tasks, a Δ value is defined for each task:

$$\Delta_i = \sum_{k=1}^{i} N_k / D_i, \text{ for } 1 \le i \le K \qquad \textbf{Eq. 4}$$

Δ_i is similar to the "intensity" defined in [17] for event-triggered systems. It can be seen as a hypothetical frequency for task T_i in the way explained in Lemma 2.

Lemma 2: The slack constraint of T_i will be satisfied if T_i and all the tasks before T_i are executed on frequencies equal to or higher than Δ_i, i.e. $S_i \ge 0$ if $f_k \ge \Delta_i$ for $1 \le k \le i$; The slack constraint of T_i will be violated if at least one task, either T_i or any task before T_i, is executed on a frequency lower than Δ_i and none of the other tasks is executed on frequencies higher than Δ_i, i.e. $S_i < 0$ if $f_k \le \Delta_i$ for $1 \le k \le i$ and $\exists j$, $1 \le j \le i$, $f_j < \Delta_i$.

Proof: The proof is straightforward.

If $f_k \ge \Delta_i$ for $1 \le k \le i$,

$$S_i(f_1, ..., f_i) = D_i - \sum_{j=1}^{i} \frac{N_j}{f_j} \ge D_i - \sum_{j=1}^{i} \frac{N_j}{\Delta_i} = D_i - \frac{\sum_{j=1}^{i} N_j}{\Delta_i} = 0$$

If $f_k \le \Delta_i$ for $1 \le k \le i$, and $\exists j$, $1 \le j \le i$, $f_j < \Delta_i$,

$$S_i(f_1, ..., f_i) = D_i - \sum_{j=1}^{i} \frac{N_j}{f_j} < D_i - \sum_{j=1}^{i} \frac{N_j}{\Delta_i} = D_i - \frac{\sum_{j=1}^{i} N_j}{\Delta_i} = 0$$

Q.E.D.

Lemma 3 and Theorem 2 show that the first BP task is the one with the largest Δ value among all the tasks.

Lemma 3: For a feasible and energy-optimal policy $Y=(f_i, 1 \le i \le K)$, if task T_I is the first BP task and $\Delta_I > f_{min}$, then $f_k = \Delta_I$ for $1 \le k \le I$.

Proof: If task T_I is the first BP task, all the tasks scheduled before T_I are on the same frequency as T_I, i.e. $f_k = f_I$ for $1 \le k \le I$. Next, Lemma 2 tells that the frequency of these tasks cannot be less than

Δ_I due to slack constraints, i.e. $f_k \geq \Delta_I$ for $1 \leq k \leq I$. Therefore we only need to show that f_k is no larger than Δ_I for $1 \leq k \leq I$.

We then make a counter statement that $f_k > \Delta_I$ for $1 \leq k \leq I$. Especially, $f_I > \Delta_I$. If T_I is the last task, i.e. $I = K$, the original policy cannot be optimal because f_I can always be reduced without affecting all the other tasks. If T_I is not the last task, since it is assumed that the processor supports continuous scaling, one can always find a pair of new frequencies, f_I', f_{I+1}', for tasks T_I and T_{I+1} respectively such that

$$f_I' > f_{I+1}', \text{ and } f_I' = f_I - \varepsilon > \Delta_I, \varepsilon > 0$$

$$\frac{N_I}{f_I} + \frac{N_{I+1}}{f_{I+1}} = \frac{N_I}{f_I'} + \frac{N_{I+1}}{f_{I+1}'} \qquad \textbf{Eq. 5}$$

The new policy is still feasible because:

1) The slack constraints of tasks from T_1 to T_{I-1} are not affected. Further, since the total execution time of T_I and T_{I+1} remains unchanged, the slack constraints of tasks from T_{I+1} to the last task are not affected either.

2) Since the frequencies of task T_I and all the tasks executed before T_I have the following property: $f_1 = \ldots = f_{I-1} > f_I' > \Delta_I$, according to Lemma 2, the slack constraint of task T_I is satisfied, i.e. $S_I' \geq 0$.

Since $E(f)$ is a convex function and the total execution time of T_I and T_{I+1} remains unchanged, according to Lemma 2 in [11], executing tasks T_I and T_{I+1} using f_I' and f_{I+1}' always consumes less energy than using the original policy. This disproves the original statement thereby proving this lemma. **Q.E.D**.

Theorem 2: For a feasible and energy-optimal $Y=(f_i, 1\leq i\leq K)$, T_I is the first BP task if $\Delta_I > f_{\min}$, and $\Delta_I > \Delta_k$ for $I<k\leq K$, and $\Delta_I \geq \Delta_k$ for $1\leq k<I$.

Proof: The theorem assumes T_I, the task with the largest Δ frequency, is not the last task, i.e. $I<K$. We make a counter statement that the first BP task is T_J, $J\neq I$, $J\leq K$.

If $J < I$, then $f_1 = \ldots = f_J > f_{J+1} \geq \ldots \geq f_I$. According to Lemma 2, f_J has to be greater than Δ_I. Together with the condition that $\Delta_I \geq \Delta_k$ for $1\leq k<I$, it can be derived that $f_J > \Delta_J$.

If $J > I$, then $f_1 = \ldots = f_I = \ldots = f_J > f_{J+1}$. According to Lemma 2, $f_I \geq \Delta_I$, hence $f_J = f_I \geq \Delta_I$. Together with the condition that $\Delta_I > \Delta_k$ for $I<k\leq K$, it can be derived that $f_J > \Delta_J$.

Therefore the counter statement leads to $f_J > \Delta_J$. This contradicts Lemma 3 thereby proving Theorem 2. **Q.E.D.**

If T_I is the first BP task, letting $f_i = \Delta_I$ for $1\leq i\leq I$ will lead to the feasible and energy-optimal policy. However, consider the second BP task denoted as T_J ($I < J$), one CANNOT simply let $f_j = \Delta_J$, for $I < j \leq J$. This is because Δ_J is calculated by assigning Δ_J to all the first J tasks. But it has been proved that all the first I tasks will be executed on Δ_I, a frequency higher than Δ_J. Therefore to determine a new BP task, one may consider a new frame that starts after the most recent BP task and a new task set that consists of all tasks after it. The Δ frequency of these tasks will be updated as following assuming T_I is the most recent BP task:

$$\Delta_i = \left(\sum_{k=I+1}^{i} N_k\right) \Big/ \left(D_i - D_I\right), \text{ for } I+1 \leq i \leq K$$

Figure 2 shows the pseudo code to find all the BP tasks and their frequencies. The complexity of finding I is same as that of a search algorithm, $O(K)$. Hence, the complexity of this algorithm is $O(K^2)$.

```
Function: Find_Optimal_DVFS_Policy
Input: (N1, N2, …, NK), D, fmin
Output: the energy-optimal Y=(f1, f2, …, fK)
  Calulate R1, R2, …, RK; j = 1;
  while (j <= K) do
    Calculate the set Δ = (Δj, …, ΔK)
    Find I that ΔI>Δk for I<k≤K and ΔI ≥Δk for j≤k<I;
    if (ΔI ≤fmin) then
      let fj = … = fK = fmin; break;
    else
      let fj = … = fI = ΔI; D = D – DI; j = I + 1;
    end if
  end while
```

Figure 2. Find the optimal DVFS policy for a given order

IV. CONTINUOUS SYSTEMS: THE OPTIMAL EXECUTION ORDER

In this section we investigate the impact of task-to-task execution order on energy consumption. More formally, *given a set of tasks, one wants to find the energy-optimal schedule (including task-to-task execution order and DVFS policy) that preserves data dependencies between tasks, consumes the least energy when faults are absent, and maintains feasibility when up-to X faults are present*. Since the system needs to reserve time for the worst case—the longest unfinished task incurs all the faults. The reservation cannot be released until the longest unfinished task is executed successfully. If the longest unfinished task is executed earlier, the reservation can be released earlier and devoted to slowing down more tasks. So to minimize energy consumption, an intuition is to schedule longer tasks earlier.

Theorem 3: For a set of independent tasks, always execute longer tasks earlier is a sufficient condition towards the minimal energy consumption.

Proof: Consider a pair of adjacent tasks T_i and T_{i+1} in the order $O = (T_1 \ldots T_i, T_{i+1} \ldots T_K)$, and $N_i < N_{i+1}$, Assume the most recent BP task before T_i is T_I with a deadline D_I, and I=0 indicates T_i is before the first BP task. The following shows that the Δ frequency of the shorter task T_i is always smaller than that of the longer task T_{i+1}:

$$\Delta_i = \frac{\sum_{k=I+1}^{i} N_k}{\left(D - D_I\right) - \left(\sum_{k=i+1}^{K} \frac{N_k}{f_{\max}} + X\frac{R_i}{f_{\max}}\right)}$$

$$\leq \frac{\sum_{k=I+1}^{i} N_k + N_{i+1}}{\left(D - D_I\right) - \left(\sum_{k=i+1}^{K} \frac{N_k}{f_{\max}} + X\frac{R_{i+1}}{f_{\max}}\right) + \frac{N_{i+1}}{f_{\max}}} = \Delta_{i+1}$$

Note that $R_i = R_{i+1}$ since $N_i < N_{i+1}$. This means that in the current order O, T_i cannot be the next BP task. Assuming no data dependency between T_i and T_{i+1}, we then switch T_i and T_{i+1} so that the new order (denoted as O^*) has T_{i+1} being executed right before T_i. We first prove that O^* is still feasible if the original policy Y is kept.

$$S_i^*(f_1, \ldots, f_{i-1}, f_{i+1}, f_i)$$

$$= D - \left(\sum_{j=i+2}^{K} \frac{N_j}{f_{\max}} + X\frac{R_i^*}{f_{\max}}\right) - \sum_{j=1}^{i+1} \frac{N_j}{f_j}$$

$$\geq D - \left(\sum_{j=i+2}^{K} \frac{N_j}{f_{\max}} + X\frac{R_{i+1}}{f_{\max}}\right) - \sum_{j=1}^{i+1} \frac{N_j}{f_j}$$

$$= S_{i+1}(f_1, \ldots, f_{i-1}, f_i, f_{i+1}) \geq 0$$

$$
\begin{aligned}
&S_{i+1}^*(f_1,\ldots,f_{i-1},f_{i+1})\\
&= D-\left(\frac{N_i}{f_{\max}}+\sum_{j=i+2}^{K}\frac{N_j}{f_{\max}}+X\frac{R_{i+1}^*}{f_{\max}}\right)-\left(\sum_{j=1}^{i-1}\frac{N_j}{f_j}+\frac{N_{i+1}}{f_{i+1}}\right)\\
&= D-\left(\frac{N_i}{f_{\max}}+\sum_{j=i+2}^{K}\frac{N_j}{f_{\max}}+X\frac{R_{i+1}}{f_{\max}}\right)-\left(\sum_{j=1}^{i-1}\frac{N_j}{f_j}+\frac{N_{i+1}}{f_{i+1}}\right)\\
&\geq D-\left(\sum_{j=i+2}^{K}\frac{N_j}{f_{\max}}+X\frac{R_{i+1}}{f_{\max}}\right)-\sum_{j=1}^{i+1}\frac{N_j}{f_j}\\
&= S_{i+1}(f_1,\ldots,f_{i-1},f_i,f_{i+1})\geq 0
\end{aligned}
$$

Note that in this deduction, $R_i^* = \max(N_i, N_{i+2},\ldots,N_K) \leq R_{i+1} = \max(N_{i+1}, N_{i+2},\ldots,N_K)$, and $R_{i+1}^* = \max(N_{i+1}, N_i, N_{i+2},\ldots,N_K) = R_{i+1} = \max(N_{i+1}, N_{i+2},\ldots,N_K)$ since $N_i < N_{i+1}$. This indicates that O^* consumes the same energy as O if Y is kept. We then prove that Y is not the optimal policy for O^* by showing that the new Δ frequencies of T_i and T_{i+1} are always smaller than Δ_{i+1}:

$$
\begin{aligned}
\Delta_i^* &= \frac{\sum_{k=I+1}^{i+1}N_k}{(D-D_I)-\left(\sum_{k=i+2}^{K}\frac{N_k}{f_{\max}}+X\frac{R_i^*}{f_{\max}}\right)}\\
&\leq \frac{\sum_{k=I+1}^{i+1}N_k}{(D-D_I)-\left(\sum_{k=i+2}^{K}\frac{N_k}{f_{\max}}+X\frac{R_{i+1}}{f_{\max}}\right)} = \Delta_{i+1}
\end{aligned}
$$

$$
\begin{aligned}
\Delta_{i+1}^* &= \frac{\sum_{k=I+1}^{i+1}N_k - N_i}{(D-D_I)-\left(\sum_{k=i+2}^{K}\frac{N_k}{f_{\max}}+\frac{N_i}{f_{\max}}+X\frac{R_{i+1}^*}{f_{\max}}\right)}\\
&= \frac{\sum_{k=I+1}^{i+1}N_k - N_i}{(D-D_I)-\left(\sum_{k=i+2}^{K}\frac{N_k}{f_{\max}}+X\frac{R_{i+1}}{f_{\max}}\right)-\frac{N_i}{f_{\max}}}\\
&\leq \frac{\sum_{k=I+1}^{i+1}N_k}{(D-D_I)-\left(\sum_{k=i+2}^{K}\frac{N_k}{f_{\max}}+X\frac{R_{i+1}}{f_{\max}}\right)} = \Delta_{i+1}
\end{aligned}
$$

In the case where T_{i+1} was a BP task in O, the optimal policy of O^*, denoted as Y^*, will take advantage of the lower Δ frequencies of T_i and T_{i+1} and result in lower energy consumption than Y. In the other case where T_{i+1} was not a BP task, such switch will not change DVFS policy since both Δ_i^* and Δ_{i+1}^* are smaller. Therefore it is a sufficient condition, but not a necessary one. **Q.E.D.**

```
Function: Find_Optimal_Order (TSet)
  USet ← TSet
  while USet <> ∅ do
    Find the longest task in USet
    if the task has unscheduled ancestors then
      Find_Optimal_Order(Fan in tree of the task)
    end if
    Schedule the task, and Remove the task from USet
  end while
```

Figure 3. Find the Optimal Order

Theorem 3 can be extended to a set of dependent tasks. For a set of dependent tasks there are situations where a task cannot be executed until all of its ancestor tasks are finished. Thus one needs to find the As-Early-As-Possible schedule for the longest tasks. The pseudo code is given in Figure 3. Initially, all tasks are unscheduled and are listed in *USet*. Each of the following **While** iteration schedules the longest task in *USet* and the tasks on which it depends. The total time complexity is $O(K^2)$.

V. CONVERT TO DISCRETE SYSTEM

If the frequency of task T_i is between two frequency levels, T_i will be assigned the higher one. This guarantees feasibility. Further, there might be some room to decrease the frequencies of some tasks without violating feasibility. The tasks are chosen according to a simple heuristic – the task that has the largest $N_i \cdot dE(f)/d(f)|_{f=f_i}$. Denote this value as the Energy Gradient (***EG***) of task T_i at f_i. The algorithm attempts to reduce the frequency of the task with the largest *EG* to its next lower level. The attempt is successful if the new policy is feasible. Otherwise, the task is masked out and the task with the second largest *EG* will be considered. The complexity of one such iteration is $O(K)$. The search stops when no task can run at its next lower level. In the worst case, the algorithm stops when the frequency levels of at most *K-1* tasks are reduced from $f_{\max}$ to $f_{\min}$ which gives a complexity $O(L \cdot K)$. Therefore the complexity of this algorithm is $O(L \cdot K^2)$. The actual runtime is much less since the input of this algorithm is already close to the optimal policy.

The complete approach therefore includes three steps: find the optimal execution order (Figure 3), then find the optimal continuous DVFS policy of the order (Figure 2), and then convert the policy to the discrete system. The aggregated time complexity is $O(L \cdot K^2)$.

VI. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the proposed approach on energy consumption and runtime overhead, we developed three processor simulators based on the published data of Intel Pentium M 765, Intel PXA255, and AMD Athlon64 4000+. All of these processors support discrete DVFS and their data sheets are available online at their manufactures' websites. The specifications of these processors are listed in Table 1. For each processor, given a maximum number of faults to tolerate, we randomly generated 600 task sets with different numbers of data dependencies. The task number in a task set is set to 7 because the Integer Linear Programming (ILP) based approach described below failed to find the optimal result in days if the number is higher. Tasks are randomly generated with cycles ranging from 0.5 to 20 million, and the utilization of a task set ranging from 10%-70%. Note that this utilization does not include the utilization caused by re-executions. The simulations are performed in a Linux PC with two 3.4GHz CPUs and 2GB RAM.

Table I. PROCESSORS USED IN SIMULATION

Processor	Pentium M 765	Athlon64-4000+	PXA255
Discrete Freq. / Voltage Levels	(2.1G, 1.340V) (1.8G, 1.276V) (1.6G, 1.228V) (1.4G, 1.180V) (1.2G, 1.132V) (1.0G, 1.084V) (0.8G, 1.036V) (0.6G, 0.988V)	(2.4G, 1.400V) (2.2G, 1.350V) (2.0G, 1.300V) (1.8G, 1.250V) (1.0G, 1.100V)	(0.4G, 1.65V) (0.3G, 1.43V) (0.2G, 1.32V)

Two results are obtained for comparison. The first one is produced by the proposed approach. The second one is the optimal results obtained by solving an ILP formulation using GNU Linear Programming Kit [16]. To the best of our knowledge, no other techniques address the same problem.

In the ILP formulation, $L \times K$ binary variables, $x_{k,l}$, ($1 \leq k \leq K$ and $1 \leq l \leq L$) are defined for each possible order of execution. $x_{k,l} = 1$ if T_k is assigned frequency level l, $x_{k,l} = 0$ if otherwise. Since each

task can only be on one frequency level, $\sum_{1\le l\le L} x_{k,l} = 1$. For each $x_{k,l}$,there is a constant $T_{k,l}$ which is the execution time of the k^{th} task if it is assigned to frequency level l. Then the feasibility constraint for the k^{th} task is $\sum_{1\le i\le k} (\sum_{1\le l\le L} x_{i,l} \cdot T_{i,l}) \le D_k$, and D_k is pre-computed using Eq. 2. Similarly, for each $x_{k,l}$,there is a constant $E_{k,l}$, which is the energy consumption of the k^{th} task if it is assigned to frequency level l. The object function to minimize is the total energy consumed by the tasks using the selected DVFS policy, as $\sum_{1\le i\le K}(\sum_{1\le l\le L} x_{i,l} \cdot E_{i,l}) + P_{idle} D$.

Comparisons are made on the energy difference between the results produced by the proposed approach and the ILP approach, and are reported in Figure 4 to Figure 6 with X being set to 1 and 2. In these figures, x-axis is the utilization and y-axis is the energy difference. As one may see, most of the times the proposed approach finds the optimal solution. Another observation is that the difference increases slightly as utilization increases. This is because when utilization is high, the tasks tend to be scheduled on high frequencies where a small difference in DVFS policy may lead to a large difference in energy consumption.

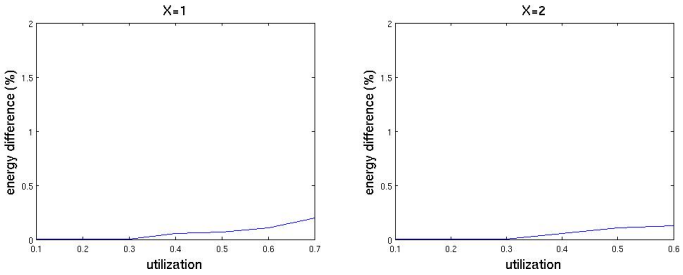

Figure 4 Simulation using Pentium M

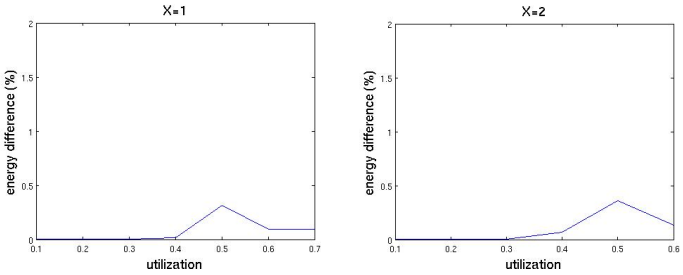

Figure 5 Simulation using Athlon 64

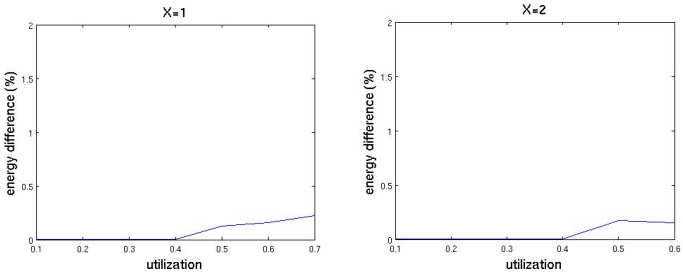

Figure 6 Simulation using PXA255

Speed wise, the practical runtime of the proposed approach is on the order of milliseconds. No noticeable runtime increase is observed for the task sets consisting of up to 1000 tasks. The ILP solver on the other hand spends around ten minutes on average for small-scale task sets that consist of only 8 tasks, and its runtime increases dramatically as the number of tasks and/or the number of frequency levels increase.

VII. CONCLUSIONS

This paper studies the trade-off between energy efficiency and fault tolerance in scheduling real-time tasks. Recognizing that the problem in discrete systems is NP-hard, the proposed approach investigates a comparable continuous system and finds the optimal solution in polynomial time. The solution is then converted to the discrete system. For a set of K tasks scheduled on a system supporting L voltage levels, the time complexity is $O(L \cdot K^2)$. The energy difference between the result obtained using the proposed approach and the optimal result obtained by solving an ILP formulation is less than 1%.

REFERENCES

[1] V. Gutnik and A. Chandrakasan, "An efficient controller for variable supply-voltage low power processing", *Symposium on VLSI Circuits*, pp. 158-159, 1996.

[2] W. Namgoong, M. Yu, and T. Meng, "A high-efficiency variable-voltage cmos dynamic dc-dc switching regulator", *IEEE International Solid-State Circuits Conference*, pp. 380-381, 1997.

[3] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, Vol. 43, No.6, pp. 2742-2750, Dec. 1996.

[4] T. Langley, R. Koga, T. Morris, "Single-event effects test results of 512MB SDRAMs," *IEEE Radiation Effects Data Workshop*, pp. 98-101, Jul. 2003.

[5] J. Hong, G. Qu, M. Potkonjak and M. Srivastava, "Synthesis techniques for low-power hard real-time systems on variable voltage processors", *The 19th IEEE Real-Time Systems Symposium* (*RTSS*), pp. 178-187, Dec. 1998.

[6] D. Zhu, R. Melhem, and D. Mosse, "The effects of energy management on reliability in real-time embedded systems," *IEEE International Conference on Computer Aided Design* (*ICCAD*), pp. 35-40, 2004.

[7] J. Chen and C. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (DVS) platforms," *International Conference on Embedded and Real-Time Computing Systems and Applications*, 2007.

[8] H. Kopetz, "Why time-triggered architectures will succeed in large hard real-time systems," *IEEE Workshop on Future Trends of Distributed Computing Systems*, 1995

[9] H. Kopetz, R. Obermaisser, "Temporal composability," *Computing & Control Engineering Journal*, Vol. 13, No. 4, Pp. 156 – 162, 2002

[10] R. Melhem, D. Moss´and E.Elnozahy, "The interplay of power management and fault recovery in real-time systems"', *IEEE Transactions on Computers*, Vol. 53, No. 2, pp. 217-231, Feb 2004.

[11] T. Ishihara and H. Yasuura. "Voltage scheduling problems for dynamically variable voltage processors," *International Symposium on Low Power Electronics and Design*, pp. 197-202, 1998.

[12] D. Zhu and H. Aydin, "Energy management for real-time embedded systems with reliability requirements," *IEEE International Conference on Computer Aided Design* (*ICCAD*), pp. 528-534, 2006.

[13] R. Jejurikar, C. Pereira and R. Gupta, "Leakage aware dynamic voltage scaling for real-time embedded systems," *IEEE Design Automation Conference* (*DAC*), pp. 275-280, 2004.

[14] O. S. Unsal, I. Koren, and C. M. Krishna, "Towards energy-aware software-based fault tolerance in real-time systems," *International Symposium on Low Power Electronics and Design*, pp. 124-129, 2002.

[15] D. Zhu, "Reliability-aware dynamic energy management in dependable embedded real-time systems," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 397-407, 2006

[16] GNU Linear Programming Kit, online at www.gnu.org /software/glpk/

[17] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *IEEE Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pp. 374–382, 1995.

[18] A. Mahmood, E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors-A Survey," *IEEE Transactions on Computers*, Vol. 37, No. 2, pp. 160-174, 1988.

[19] N. Oh, P. Shirvani, and E. McCluskey, "Controlflow checking by software signatures," *IEEE Transactions on Reliability*, Vol. 51, no 2, pp. 111-122, Mar. 2002.

[20] S. K. Reinhardt, S. S. Mukherjee, "Transient-fault detection via simultaneous multithreading," International Symposium on Computer Architecture, pp. 25–36, June Jun. 2000.