Online Task-Scheduling for Fault-Tolerant Low-Energy Real-Time Systems*

Tongquan Wei†, Piyush Mishra†, Kaijie Wu‡ and Han Liang‡
†ECE Department, Michigan Technological University, Houghton MI 49931, USA
‡ECE Department, University of Illinois, Chicago IL 60607, USA
†{twei, mishra}@mtu.edu, ‡{kaijie, hliang4}@uic.edu

Abstract: **In this paper we investigate fault tolerance and Dynamic Voltage Scaling (DVS) in hard real time systems. We present two low-complexity fault-aware scheduling algorithms that combine feasibility analysis of Rate Monotonic Algorithm (RMA) schedules and DVS-based frequency scaling using exact characterization of RMA algorithm. These algorithms lay the foundation for highly efficient online schemes that minimize energy consumption by adapting DVS policies to runtime behavior of tasks and fault occurrences without violating the offline feasibility analysis. Simulation results demonstrate energy savings of up to 60% over low-energy offline scheduling algorithms [22].**

1. Introduction

Embedded systems are highly susceptible to faults due to increasing level of integration, reducing size of transistor features, and lowering of voltage levels in integrated circuits [2],[3]. Hard real time embedded systems used in mission-critical applications such as navigation system, process control, and system monitoring must finish all scheduled tasks on time and demand high fault tolerance with low overheads. Need for reliability is rising even in non-critical applications since significant progress in technology is driving wide-scale deployment of real time systems in daily life products which are prone to operate in harsher environments but have lower expectancy of failures. Common examples include mobile devices and sensor networks deployed in open fields which suffer frequent jolts and damage and are exposed to strong radiations [1].

Fault tolerance mechanisms used for detecting and correcting faults can be classified based on (1) place of deployment, (2) fault coverage, (3) fault or error latency, and (4) cost of recovery. Typically, fault tolerance in real-time systems is achieved via online fault detection [6] followed by a hardware-based checkpoint and rollback recovery mechanism. It allows processor to rollback to previously known valid state to resume normal execution by exploiting the available slack time [7],[8]. Traditionally, fault tolerance techniques try to maximize fault coverage while minimizing the fault detection latency and associated costs [12],[13],[14]. Cost is usually measured in terms of hardware and/or time overheads and is of considerable importance to real time embedded systems due to severe resource constraints. Owing to the fast evolving application of real-time systems in battery-powered portable devices, energy consumption has emerged as another important design constraint. In this paper we propose a systematic approach to energy-efficient fault-tolerant task scheduling techniques for hard real-time systems.

Although dynamic power management is an active area of research and several techniques have been proposed to minimize energy consumption at hardware level [3], software level [4], and system level [5], very little attention has been paid to exploit these techniques to design energy-efficient fault-tolerance schemes for real-time embedded systems. At hardware level power saving is achieved via various circuit design optimization strategies while at software level power efficient modes are selected using a software controller. At system level energy efficiency is achieved by dynamically reconfiguring active components of the system and selectively turning off system components when they are idle. One such promising power management technique is Dynamic Voltage Scaling (DVS) that exploits technological advances in power supply circuits to reduce energy consumption and has attracted considerable attention due to its effectiveness and ease of use [15],[16]. It reduces power consumption of processor by dynamically scaling down voltage at the cost of increased execution time [9],[10],[11].

1.1. Related work

Melham et al. proposed techniques to exploit slacks in task schedules to reduce energy consumption while tolerating faults based on DVS [18]. They make several simplifying assumptions such as a task is susceptible to at most one fault occurrence and the processor can scale its frequency in continuous range. Zhang et al. developed an online scheduling algorithm that combines checkpointing with DVS to tolerate faults in soft real-time uni-processor systems with periodic tasks [17]**.** In [19] they proposed a fixed priority offline scheme using Rate Monotonic Algorithm (RMA) to tolerate faults in hard real time systems based on simplified assumptions, such as fault-free checkpointing and rollback recovery and negligible DVS overheads and cost for restoration of system state. They proposed a computationally intensive algorithm to verify feasibility of task schedules while deterministically tolerating up to 'k' faults. It operates in two phases – the first phase performs feasibility analysis on checkpointing based scheme to tolerate 'k' faults and the second phase scales processor voltage to minimize energy consumption subject to timing constraints derived from feasibility analysis in phase 1. This scheme was improved in [22] to overcome simplistic assumptions, but at the cost of increased complexity in algorithm. For application level frequency scaling, where all tasks in a task set run at the same speed, the algorithm examines various processor frequency levels to determine the lowest frequency that satisfies timing constraints subject to 'k' faults. It has time complexity of $O(n^2RL)$, where n is the number of periodic tasks in a task set, R is the ratio of largest task period to smallest task period, and L is the number of frequency levels supported by the processor. For task level frequency scaling, where each task in a task set is assigned an individual frequency, the algorithm performs an exhaustive search to derive the optimal combination of speed assignments for each task such that all timing constraints are satisfied subject to 'k' faults. Time complexity in this case is $O(n^2RL^n)$. Due to uncertainties in task execution times and fault occurrences these offline scheduling algorithms do not adapt well to the runtime behavior of tasks. Further, due to their high complexity they are not suitable for online re-evaluation of task schedules or frequency assignments.

In this paper we propose efficient offline algorithms that combine feasibility analysis and voltage scaling for hard real time systems based on exact characterization of RMA. These algorithms lay the foundation for efficient online schemes that minimize energy consumption by adapting DVS policies to the runtime behavior of tasks and fault occurrences.

*This work is in part supported by Center for Integrated Systems in Sensing, Imaging, and Communication (CISSIC) at MTU.

ICCAD'06, November 5-9, 2006, San Jose, CA

1.2. Outline

Rest of the paper is organized as follows. Section 2 describes the exact characterization of classic RMA algorithm (EC-RMA) used for scheduling hard real time tasks. Section 3 introduces the proposed EC-RMA based efficient offline schemes to verify feasibility of schedules and identify opportunities for DVS to save energy. Section 4 extends these schemes to dynamically re-evaluate DVS policies online to adapt to runtime behavior of tasks and fault occurrences. Section 5 presents the experimental results to demonstrate energy savings. Section 6 presents the conclusions of study and future research directions.

2. Scheduling algorithms for fault-tolerant hard real-time systems

Consider a task set consisting of n independent periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$. Timing parameters of task τ_i are defined as a tuple $\tau_i = \{T_i, D_i, C_i\}$, where T_i is the period, D_i is the deadline, and C_i is the worst case execution time in cycles under fault free conditions and without checkpointing overheads [22]. It is assumed that checkpointing intervals for a given task are equal but no assumption is made regarding relationship among the timing parameters. System parameters are defined as follows:

- Cs: time needed by checkpointing schemes to save system state, referred to as checkpointing cost.
- Cr: time needed to retrieve stored system state, referred to as state recovery cost.
- k: upper bound on the number of faults that must be tolerated during each task instance.

It is assumed that

- Processor supports L discrete frequency levels: $f_1 < f_2 < \ldots < f_L$.
- System parameters, Cs and Cr, are constant and independent of processor frequency scaling.
- Tasks are arranged in decreasing order of priorities according to fixed priority offline scheme RMA [20], that is, $T_1 < T_2 < , \ldots, < T_n$ such that period of τ_i is smaller than the period of τ_j for i<j.
- Task set is scheduled using RMA and the resulting schedule is feasible under fault free condition.
- Faults are detected as soon as they occur and online slack time is ready as soon as the task finishes execution.

It was shown in [22] that under the assumption that all checkpoint intervals for the same task are equal, optimal number of checkpoints m_i for task τ_i, that minimizes its worst case response time, is given by $m_i = \text{Max}(\| \sqrt{kC_i/Cs} -1\|, 0)$. Term $\| \sqrt{kC_i/Cs} -1\|$ denotes the value from pair $\{\lceil \sqrt{kC_i/Cs} -1 \rceil, \lfloor \sqrt{kC_i/Cs} -1 \rfloor\}$ that minimizes the worst case response time of task τ_i. Therefore, total fault-free execution time of task τ_i is the sum of its execution time and checkpointing overhead, that is, $C_i + m_iCs$.

Next sub-section provides an overview of the RMA algorithm and its exact characterization. Classical RMA analysis is conservative and the next sub-section shows that an exact characterization relaxes the constraints to tolerate faults without significant degradation in system performance.

2.1. Exact characterization of RMA (EC-RMA)

Rate Monotonic Algorithm (RMA), proposed by Liu and Layland in 1973, is an optimal fixed priority algorithm that schedules periodic tasks with hard deadlines by assigning higher priorities to tasks with shorter periods [20]. Original algorithm had strict requirements, such as tasks in a task set were independent and task deadlines were equal to or less than task periods, many of which were relaxed later to extend the scope of its application. Worst case behaviors of RMA occurs when all tasks in a task set are instantiated simultaneously and are ready for execution immediately after initiation. This time instant is called *critical instant* and it has been shown that a schedule of independent periodic tasks is feasible at critical instant if the first instance of each task is schedulable. The worst case utilization bound of a task set is given by $n(2^{1/n}-1)$, where n is the number of tasks in the task set.

J. Lehoczky et al. showed that the worst case analysis of RMA is a conservative estimate which can be relaxed based on an exact characterization of RMA to derive both sufficient and necessary conditions for feasibility analysis of a schedule [21]. They proved that a set of periodic tasks is schedulable for all task phasing if and only if the cumulative demand of each task on processor is equal to or less than the current available processor time. Specifically, let $W_i(t)=\sum_{(1 \le j \le i)} C_j \lceil t/T_j \rceil$ denote the cumulative demand of task τ_i on processor over [0, t] assuming 0 as the critical instant. The necessary and sufficient condition for a set of periodic tasks to be schedulable is given as

$$\text{Max}_{(1 \le i \le n)}\{\text{Min}_{(0 < t \le Ti)}[W_i(t)/t]\} \le 1 \quad (2.1)$$

It was shown that only a finite number of time instants, called scheduling points, need to be checked for feasibility analysis since normalized demand of task τ_i on processor, $W_i(t)/t$, is strictly decreasing except at scheduling points. Scheduling points S_i associated with task τ_i are defined as multiples of T_g for $T_g \le T_i$, that is,

$$S_i=\{hT_g | g=1,2,\ldots,i.\ h=1,2,\ldots,\lfloor T_i/T_g \rfloor\}. \quad (2.2)$$

Therefore, equation (2.1) can be re-written as

$$\text{Max}_{(1 \le i \le n)}\{\text{Min}_{(t \in Si)}[W_i(t)/t]\} \le 1 \quad (2.3)$$

Proposed task scheduling techniques exploit exact characterization of RMA to efficiently verify the feasibility of an offline schedule in the presence of faults and voltage scaling and to adapt DVS policies online to variations in task execution time and fault occurrences. These techniques offer three advantages over previous techniques: (1) higher tolerance to fault overheads due to relaxed constraints of exact characterization (2) low-cost offline schedule feasibility analysis schemes, (3) efficient extension to online DVS policy re-evaluations due to relatively smaller number of scheduling points at which feasibility checks need to be performed.

2.2. Feasibility analysis of fault-tolerant low-energy RMA schedules

Fault correction using hardware-based checkpoints and rollback on fault detection incurs time overheads. A feasible RMA schedule may become infeasible in the presence of faults due to the extra cost of recovering from faults by rolling back and re-executing the faulty computations. Faults can occur during normal computation, checkpointing, or state restoration after a fault occurrence. The worst case recovery cost, corresponding to all k faults occurring during checkpoint saving, includes k re-executions, $kC_i/(m_i+1)$, and k checkpoint savings and state restorations, k(Cs+Cr). Therefore, total worst case cost of task τ_i with fault overheads is given by

$$TC_i=C_i+kC_i/(m_i+1)+m_iCs+k(Cs+Cr) \quad (2.4)$$

Of this, the first three terms are frequency dependent while the last term is frequency-independent according to the assumptions made. The necessary and sufficient condition for feasibility analysis of an RMA schedule shown in (2.3) becomes

$$\text{Max}_{(1 \le i \le n)}\{\text{Min}_{(t \in Si)}[\sum_{(1 \le j \le i)} TC_j \times \lceil t/T_j \rceil/t]\} \le 1 \quad (2.5)$$

Feasibility of an RMA schedule may also be affected adversely due to prolonging of clock period as a result of voltage scaling in DVS capable processors. In deep submicron VLSI power consumption of a microprocessor is directly proportional to $v_{dd}^2 f$, where v_{dd} is the supply voltage and f is the operating frequency [24]. Energy consumed by a task with N instruction cycles at clock period Clk is proportional to $(V_{dd}^2 \times f) \times (N \times Clk) = V_{dd}^2 N$. It is easy to observe that energy savings can be maximized by scaling voltage as early and as low as possible. However, circuit delay increases

almost linearly as voltage decreases. Let Clk_{FS} be the clock period at voltage V_{dd} and Clk_{new} at scaled voltage level V_{dd_new}. Thus, $Clk_{new} \approx Clk_{FS}V_{dd}/V_{dd\_new} = \alpha\ Clk_{FS}$, where $\alpha = V_{dd}/V_{dd\_new}$ is the stretching factor [24].

In a hard real-time system tasks are subject to timing constraints and must finish execution before their deadlines. These constraints impose an upper bound on α. In other words, $V_{dd\text{-}new}$ should be lower bounded to maintain a feasible schedule. Exact characterization of RMA provides a low-cost systematic approach to verify the feasibility of a schedule while tolerating faults for enhanced reliability and scaling voltage for energy-efficiency.

3. Low-complexity energy-efficient offline scheduling

Two techniques are proposed for energy-efficient fault-tolerance in hard real-time systems. First technique selects a common frequency for all tasks of an application (a task set) while second technique selects individual frequencies for each task in the task set in order to minimize energy consumption. Both techniques integrate fault-tolerance and DVS policy evaluations to systematically search for the energy-optimum fault-tolerant schedule for a given set of tasks. It is assumed that the task set is schedulable in the absence of faults at a given frequency. Without loss of generality, it is assumed to be the lowest frequency level and can be easily modified to accommodate other frequency levels.

Proposed techniques scale voltage as early and as low as possible while providing sufficient slack times to tolerate up to k faults. It is shown that the proposed algorithms systematically produce the most energy-efficient schedule as the first feasible solution and have lower complexity compared to previous techniques. Further, they support low-cost online DVS policy re-evaluations to reduce energy consumption by adapting to variations in task execution times and fault occurrences, as discussed in section 4. Experimental results presented in section 5 show that online re-evaluation of DVS policies saves considerable energy compared to offline scheduling.

3.1. Application level voltage scaling (A-DVS)

A-DVS algorithm, shown in Figure 1, starts by calculating the scheduling points S_i for each task τ_i and then iteratively performs feasibility analysis using EC-RMA to select optimum DVS strategy while tolerating k faults in each task instance. It uses a flag, *Schedulable*, which is set to '0' whenever a task is found un-schedulable at the current common frequency level and a buffer, *Demand*, to hold the cumulative workload demand on processor at scheduling points at the current frequency level. Inputs to the algorithm are an RMA schedule which is feasible at the lowest frequency level *l*=1, maximum number of faults each task should tolerate = k, checkpoint saving cost = Cs, and checkpoint recovery cost = Cr.

Lines 2 to 4 of A-DVS algorithm iteratively compute scheduling points of n tasks with time complexity of $O(n^2R)$, where n is the number of tasks in the task set and R is the ratio of the largest period to the smallest period. Rest of the algorithm operates in 2 phases which are iterated for each task in the task set. Phase 1, consisting of lines 6 to 9, derives the optimum number of checkpoints at current frequency level to compute the worst case total cost TC of current task (τ_i) if the previous task was found schedulable, or the current and all higher priority tasks (τ_j, $1 \le j < i$) if current task was found un-schedulable in the previous iteration. Phase 2, consisting of lines 10 to 22, verifies the schedulability of current task using equation (2.5) and performs frequency scaling if required. For each scheduling point of the current task lines 11 to 14 compute the cumulative workload and lines 15 to 16 set the *Schedulable* flag appropriately. If the current task is found schedulable algorithm proceeds to the next task, else it scales the common frequency up by one level and re-evaluates the schedulability of the current task. This process continues until a feasible schedule is found for all tasks or the highest frequency level is reached without satisfying equation (2.5), in which case the task set schedule is deemed infeasible.

Procedure A-DVS(k, Cs, Cr)
1. *l*=1; Schedulable=0; Demand=0; i=j=p=q=1;
2. **for** τ_i, $1 \le i \le n$ **do**
3. $S_i = \{h \times T_g | g=1,2,...,i.\ h=1,2,...,\lfloor T_i/T_g \rfloor\}$;
4. **end for**
5. **for** τ_i, $1 \le i \le n$ **do**
   *//**Phase 1--**Compute total cost TC_j*
6. **for** τ_j, $j \le i$ **do**
7. $m_j = \max\{\| \sqrt{k \times C_j/(f(l) \times Cs)} - 1\|, 0\}$;
8. $TC_j = (C_j + k \times C_j/(m_j+1))/f(l) + m_j \times Cs + k \times (Cs+Cr)$;
9. **end for**
   *//**Phase 2--**Feasibility analysis and frequency scaling*
10. **for** $s_{iq} \in S_i$ **do**
11. Demand = 0;
12. **for** τ_p, $1 \le p \le i$ **do**
13. $Demand = Demand + TC_p \times \lceil s_{iq}/T_p \rceil$;
14. **end for**
15. **if** $Demand \le s_{iq}$ **then** Schedulable=1; **break**;
16. **else** Schedulable=0; **end if**
17. **end for**
18. **if** Schedulable =0 **then**
19. i--; j=1;
20. **if** $l < L$ **then** *l*++;
21. **else print** "Infeasible Schedule"; **break; end if**
22. **else** j=i+1; **end if**
23. **end for**

Figure 1 Application level voltage scaling algorithm

For example, assume that during iteration i task τ_j, for $1 \le j < i$, are schedulable at frequency level *l*. A-DVS first computes TC_i for task τ_i at frequency level *l*. Then, it evaluates equation (2.5) at each of the task's scheduling points, s_{iq}, to check if τ_i is schedulable. If the condition is satisfied at any s_{iq}, task τ_i is schedulable at frequency level *l* and algorithm continues on to verify the next task τ_{i+1}, else it scales the frequency up by one level to f_{l+1}, re-computes TC_j for $1 \le j \le i$ and repeats the process. Note that it is not required to re-evaluate higher priority tasks τ_j ($1 \le j < i$) since scaling the common frequency up does not have any adverse affect on their schedulability.

Time complexities of phase 1 and phase 2 of the algorithm are $O(n^2L)$ and $O(Rn^2L)$ respectively. Therefore, the overall time complexity of this algorithm for a set of n tasks is $O(Rn^2L)$ and the average time complexity per task is O(RnL), which is of the same order as the application level technique proposed in [22] but it supports efficient online adaptation of DVS policies to exploit runtime uncertainties in task execution times and fault occurrences to save energy, as shown in section 4.1.

3.2. Task level voltage scaling (T-DVS)

T-DVS algorithm, shown in Figure 2, is similar to A-DVS except that whenever a task is found un-schedulable it iteratively increases the frequencies of equal and higher priority tasks with lowest frequency level until the current task becomes schedulable or the highest frequency level for all tasks is reached and the task set is found infeasible. The approach is guided by the principle that maximum energy-efficiency is achieved by maintaining as low frequency for as long as possible.

T-DVS calculates the scheduling points S_i for each task τ_i and then iteratively performs feasibility analysis to select optimum

DVS strategy while tolerating k faults in each task instance. Besides the variables *Schedulable* and *Demand* defined in section 3.1 three new variables $Freq_i$ and $FreqLevel_i$ are introduced to denote the frequency value and frequency level of task τ_i, and RefID to denote the index of task with minimum frequency. Inputs to the algorithm are an RMA schedule which is feasible at the lowest frequency level *l*=1, maximum number of faults each task instance should tolerate = k, checkpoint saving cost = Cs, and checkpoint recovery cost = Cr.

Lines 2 to 4 iteratively compute the scheduling points of n tasks with time complexity of $O(n^2R)$. Rest of the algorithm operates in 2 phases which are iterated for each task in the task set. Phase 1, consisting of lines 6 to 7, derives the optimum number of checkpoints at current frequency level to compute the worst case total cost TC_{RefID} for current task (τ_{RefID}) if the previous task was found schedulable, or the task with lowest frequency among all equal and higher priority tasks (τ_{RefID}, $1 \le RefID \le i$) if the previous task was found un-schedulable. Unlike A-DVS, phase 1 of T-DVS takes constant time.

Phase 2, consisting of lines 8 to 24, verifies the schedulability of current task using equation (2.5) and performs frequency scaling at task level, if required. For each scheduling point of the current task lines 9 to 12 compute the cumulative workload of tasks τ_p ($1 \le p \le i$) and lines 13 to 15 set the *Schedulable* flag appropriately. In lines 17 to 18 if the current task is found schedulable algorithm proceeds to the next task, else task with minimum frequency among all equal and higher priority tasks is found. In line 19 to 23 if the minimum frequency level is found equal to the highest frequency level the task set is deemed infeasible, else the frequency of this task is raised by one level and schedulability of task τ_i is re-evaluated using the updated cumulative workload. This process continues till a feasible schedule is found for all tasks or the highest level of frequency is reached without satisfying equation (2.5), in which case the task set schedule is deemed infeasible. This approach ensures that as many tasks run at as low frequency as possible to save energy.

Unlike the task level offline feasibility analysis algorithm of [22] T-DVS does not need to exhaustively explore all L^n possible combinations of tasks and frequency levels. The first feasible schedule generated by the algorithm is energy-optimum since it consists of lowest possible frequency combinations for each task. Time complexity of T-DVS is also dominated by the time complexity of feasibility analysis and frequency scaling as compared to the cost $O(n^2R)$ for deriving scheduling points. Time complexity of feasibility analysis per task is determined by a nested loop consisting of nR iterations. A popular MINIMUM() algorithm with cost of O(n) [23] is used during frequency scaling to determine the task with minimum frequency level, $FreqLevel_j$ ($1 \le j \le i$). In the worst case, nL frequency scaling is performed for each task and each scaling involves up to (nR+n) iterations to perform feasibility analysis at scheduling points and to find the minimum frequency. Therefore, feasibility analysis and frequency scaling involve nL(nR+n) iterations for each task and time complexity of T-DVS per task is $O(Rn^2L)$ which is order of magnitude better than previous techniques. Even though the time complexity of T-DVS is much higher than A-DVS, its online re-evaluation of DVS policies is considerably simpler, as shown in section 4.2.

4. Re-evaluation of DVS policies

Offline scheduling assumes that tasks exhibit worst case execution times and all 'k' faults occur during the checkpointing of tasks. However, runtime behavior of task execution times and occurrences of faults can vary significantly [25]. Hence, online re-evaluation of DVS policies that adapt to runtime characteristics of tasks and faults can save significant energy. Proposed algorithms A-DVS and T-DVS provide efficient mechanisms to exploit the slack time generated during runtime to further slow down the processor and save energy. Note that feasibility analysis carried out by A-DVS or T-DVS guarantees that offline schedules meet all timing constraints. Dynamic DVS policies proposed in this section operate within these bounds to ensure that the feasibility of resultant modified schedules is maintained.

Procedure T-DVS(k, Cs, Cr)

1. RefID=1;$Freq_i$=f(1) and $FreqLevel_i$=1 for $1 \leqslant i \leqslant n$;
2. **for** τ_i, $1 \leqslant i \leqslant n$ **do**
3. $S_i = \{h \times T_g | g=1,2,...,i.\ h=1,2,...,\lfloor T_i / T_g \rfloor\}$;
4. **end for**
5. **for** τ_{RefID}, $1 \leqslant RefID \leqslant n$ **do**

   //***Phase 1***--*Compute total cost* TC_i
6. $m_{RefID} = \max\{\lfloor \sqrt{k \times C_{RefID}/(Freq_{RefID} \times Cs)} - 1 \rfloor, 0\}$;
7. $TC_{RefID} = (C_{RefID} + kC_{RefID}/(m_{RefID}+1))/Freq_{RefID} + m_{RefID}Cs + k(Cs+Cr)$;

   //***Phase 2***--*Feasibility analysis and frequency scaling*
8. **for** $s_{iq} \in S_i$ **do**
9. Demand=0;
10. **for** τ_p, $1 \leqslant p \leqslant i$ **do**
11. $Demand = Demand + TC_p \times \lceil s_{iq}/T_p \rceil$;
12. **end for**
13. **if** Demand $\leqslant s_{iq}$ **then**
14. Schedulable = 1; **break**;
15. **else** Schedulable = 0; **end if**
16. **end for**
17. **if** Schedulable =1 **then** i++; RefID=i-1; **continue**;
18. **else**
19. Find $Min_{(1 \le j \le i)}Freq_j$ and $Min_{(1 \le j \le i)}FreqLevel_j$;
20. **if** $Min_{(1 \le j \le i)}FreqLevel_j < L$ **then**
21. $Min_{(1 \le j \le i)}FreqLevel_j$++;
22. $Min_{(1 \le j \le i)}Freq_j$++; RefID=j-1;
23. **else print** "Infeasible Schedule"; **break**; **end if**
24. **end if**
25. **end for**

Figure 2 Task level voltage scaling algorithm

4.1. Re-evaluation of DVS policies at application level

If A-DVS generates a feasible schedule at frequency level *l* for a given task set it implies that one or more tasks failed to satisfy equation (2.5) at frequency level (*l*-1). Slack time generated online due to less than expected number of faults and due to better than expected execution time of a task could be used to dynamically scale down the processor speed. At the end of execution of each task online DVS policies determine whether the generated slack time is sufficient to scale down the processor speed by comparing the amount of time needed for the schedule of remaining tasks to be feasible at f_{l-1} with the slack time generated.

Execution time overflow, $ovfl_{ij}$, models the additional time required by task τ_i to be schedulable at frequency level j. It is set to '0' if task τ_i is schedulable at frequency level j, else is computed as the difference between worst case response time R_{ij} of task τ_i including fault overhead at frequency level j and its deadline D_i, as shown below:

$$ovfl_{ij} = \begin{cases} R_{ij} - D_i & R_{ij} \ge D_i \\ 0 & R_{ij} < D_i \end{cases} \quad (4.1)$$

This straightforward approach is highly computation intensive. We propose an alternate simple, yet energy-efficient, approach to

compute $ovfl_{ij}$. For each task τ_i $ovfl_{ij}$ is the minimum of difference between its cumulative time demand on processor at frequency level j and each scheduling point. Therefore, for task τ_i ($1 \le i \le n$) at each frequency level f_j ($1 \le j \le L$), its corresponding $ovfl_{ij}$ is given by

$$ovfl_{ij}=Max\{Min_{(t1 \in Si)}(t_2 - t_1), 0\} \quad (4.2)$$

$s_{ij} \in S_i$ are the scheduling points for task τ_i and t_2 is given by

$$t_2=\sum_{(1 \le p \le i)}[(C_p+kC_p/(m_p+1))/f_j+k(Cs+Cr)+m_pCs]\lceil s_{ij}/T_p \rceil.$$

Proposed scheme incurs constant time overhead since both t_1 and t_2 can be pre-computed during offline feasibility analysis of A-DVS algorithm and stored in system memory. A task set of n periodic tasks feasible at frequency level *l* would require $(n-1)\times(l-1)$ memory. In general, the number of frequency levels supported by a processor is small and the number of tasks in a task set is also limited. For example, DVS capable Intel XScale processor supports only 3 voltage levels – 200, 300, and 400 MHz [22] and real life GAP task set contains only 17 tasks [26]. Therefore, memory overhead to store execution time overflows is quite small. Since the proposed scheme computes minimum value of overflow for each task, it provides better opportunity to scale the processor frequency.

Let slk_{ij} denote the accumulated slack time after task τ_i finishes execution at frequency level j. slk_{ij} may include unutilized slack time generated by higher priority tasks and is assumed to be updated and ready as soon as task τ_i finishes its execution. In a fixed-priority real-time system slack time generated by task τ_i can only be used by tasks of lower priority since tasks with equal and higher priority finish execution before this slack is generated. If slk_{ij} is greater than the sum of execution time overflows of all remaining lower priority tasks, that is, $slk_{ij} \ge \sum_{(i+1 \le p \le n)} ovfl_{p,(j-1)}$, processor speed is scaled down to frequency (j-1).

Example: Consider a task set of four periodic tasks running on a processor which supports 3 frequency levels, as shown in Table 1. Execution time overflows for each task at frequency level 3 are zero, denoting a feasible schedule at frequency level 3. At frequency level 2, task τ_1 and τ_2 are schedulable while task τ_3 and τ_4 are not schedulable since $ovfl_{12} = ovfl_{22} = 0$ but $ovfl_{32} = 1$ and $ovfl_{42} = 2$. Therefore, processor runs at frequency level 3 to maintain the feasibility of schedule. However, if the slack generated during runtime $slk_{23} \ge (ovfl_{32} + ovfl_{42}) = 3$ processor can be scaled down to level 2 without violating the feasibility of the schedule.

Table 1 runtime DVS re-evaluation

Speed level / Tasks	1	2	3
τ_1	$ovfl_{11}=0$	$ovfl_{12}=0$	$ovfl_{13}=0$
τ_2	$ovfl_{21}=3$	$ovfl_{22}=0$	$ovfl_{23}=0$
τ_3	$ovfl_{31}=4$	$ovfl_{32}=1$	$ovfl_{33}=0$
τ_4	$ovfl_{41}=5$	$ovfl_{42}=2$	$ovfl_{43}=0$

4.2. Re-evaluation of DVS policies at task level

Dynamic re-evaluation of DVS strategies at task level is simpler since T-DVS statically derives optimum combination of frequency allocation to tasks and each frequency can be scaled individually. For example, after task τ_{i-1} finishes execution scheduler checks if the accumulated slack time from tasks τ_j ($1 \le j \le i-1$), slk_{i-1}, is large enough to scale down frequency $Freq_i$ for task τ_i by one or more levels. This is achieved by comparing the overflow for task τ_i at lower frequency levels with the generated slack. If slk_{i-1} is more than the overflow, scale down processor frequency $Freq_i$ and update slk_{i-1} by subtracting execution time overflow, else accumulate the slack time for tasks with lower priorities. For example, assume that task τ_i is scheduled to execute at frequency level *l*. After task τ_{i-1} finishes its execution and the corresponding slack time slk_{i-1} is ready, add slk_{i-1} to the total cost TC_i of task τ_i and compute new frequency level f_l' as follows:

$$f_l'=f_l\times TC_i/(slk_{i-1}+ TC_i) \quad (4.3)$$

Comparing f_l' with the lower frequency levels supported by processor determines whether or not to scale the frequency for this task. This scheme takes constant time.

The proposed simple scheme is not energy-optimal since available slack time is utilized only by the immediate next task rather than proportionally distributed among all remaining tasks. Since fault occurrences during remaining tasks determine the total energy consumption, optimal energy consumption cannot be calculated until all the tasks are executed. Trying to find an energy-optimal scaling scheme during execution of a schedule may not bring any benefit at all.

5. Experimental results

Proposed schemes were validated for energy efficiency and fault tolerance using extensive simulation experiments. Real life task set benchmarks from [26] were used to compare energy savings with offline scheduling schemes proposed in [22]. Benchmarks consist of 6 to 17 tasks per set with significantly varying timing characteristics. Number of faults for each task was generated randomly between 0 to k and simulation was run 10000 times to compensate for the stochastic property in fault occurrences. Two DVS-capable processors, Transmeta Crusoe which supports 5 voltage and frequency levels (V, MHz) – (1.2, 300), (1.225, 400), (1.35, 533), (1.5, 600), and (1.6, 667) – and Intel XScale PXA260 which supports 3 voltage and frequency levels (V, MHz) – (1.0, 200), (1.1, 300), and (1.3, 400) – were used for energy consumption estimation [22].

Due to lack of space we present results for only two benchmarks – Computer Numerical Control (CNC) and an Inertial Navigation System (INS) – under the assumption that given task execution times correspond to maximum CPU speed. It is also assumed that energy consumption of checkpointing and data retrieval is 160μJ and energy for a single DVS transition is 30μJ, as suggested in [22], and online energy savings correspond only to the variation in fault occurrences. Simulation results are presented for both A-DVS and T-DVS techniques and compared with the JFTC, JFTA and JFTT techniques proposed in [22]. These techniques refer to offline constant frequency, application-level frequency scaling, and task-level frequency scaling respectively. $E_{13} = (E_1-E_3)/E_1\times100\%$ denotes energy savings of A-DVS and T-DVS over JFTC and $E_{23} = (E_2-E_3)/E_2\times100\%$ denotes energy savings of A-DVS over JFTA and T-DVS over JFTT. "NF" denotes that the schedule is infeasible.

Table 2 and Table 4 show that the proposed application level technique saves 22 to 52% and 30 to 52% energy over JFTC and 22 to 50% and 30 to 56% energy over JFTA on Crusoe and XScale processors respectively. Similarly,

Table 3 and Table 5 show that the proposed task level technique saves 22 to 58% and 30 to 52% energy over JFTC and 18 to 57% and 30 to 60% over JFTT on Crusoe and XScale processors respectively. It is also shown (e.g. for k = 5 for CNC and k = 4 for INS) that proposed techniques have higher fault tolerance capabilities due to the relaxed constraints of exact characterization of RMA based approaches.

Table 2 Application level for Transmeta Crusoe

Task set	k	JFTC E_1(mJ)	JFTA E_2(mJ)	A-DVS E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	18.1	14.6	10.1	44.2	30.8
	2	24.3	21.2	12.9	46.9	39.2
	3	29.8	26.6	15.1	49.3	43.2
	4	34.9	33.6	16.7	52.1	50.3
	5	NF	NF	18.6	-	-
INS	1	6050.7	5467.2	3986.0	34.1	27.1
	2	6735.1	6735.1	5246.4	22.1	22.1
	3	7300.2	7300.2	5617.5	23.1	23.1
	4	NF	NF	5935.8	-	-

Table 3 Task level for Transmeta Crusoe

Task set	k	JFTC E_1(mJ)	JFTT E_2(mJ)	T-DVS E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	18.1	14.9	10.0	44.8	32.9
	2	24.3	21.1	12.9	46.9	38.9
	3	29.8	26.7	13.4	55.0	49.8
	4	34.9	34.1	14.5	58.5	57.5
	5	NF	NF	13.1	-	-
INS	1	6050.7	5457.6	4087.9	32.4	25.1
	2	6735.1	6222.1	5070.4	24.7	18.5
	3	7300.2	7284.1	5637.4	22.8	22.6
	4	NF	NF	5961.6	-	-

Table 4 Application level for Intel XScale

Task set	k	JFTC E_1(mJ)	JFTA E_2(mJ)	A-DVS E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	7.6	8.2	3.6	52.6	56.1
	2	12.8	13.8	7.1	44.5	48.6
	3	17.6	18.8	9.0	48.9	52.1
	4	22.2	22.2	10.5	52.7	52.7
	5	NF	NF	12.9	-	-
INS	1	1326.2	1326.2	923.8	30.3	30.3
	2	1853.6	1853.6	1248.4	32.6	32.6
	3	2298.2	2298.2	1510.6	34.3	34.3
	4	NF	NF	1758.8		

Table 5 Task level for Intel XScale

Task set	k	JFTC E_1(mJ)	JFTT E_2(mJ)	T-DVS E_3 (mJ)	E_{13} (%)	E_{23} (%)
CNC	1	7.6	9.1	3.6	52.6	60.4
	2	12.8	14.5	7.5	41.4	48.3
	3	17.6	18.5	9.5	46.0	48.6
	4	22.2	22.5	11.0	50.5	51.1
	5	NF	NF	10.8	-	-
INS	1	1326.2	1327.5	932.5	30.3	30.4
	2	1853.6	1855.9	1292.6	30.3	30.4
	3	2298.2	2299.0	1496.5	34.9	34.9
	4	NF	NF	1726.5	-	-

According to [22] DVS on XScale is ineffective since overheads are comparable to energy savings due to low processor power consumption (178, 283, and 411 mW) and support for small number of voltage levels [22]. On other hand, proposed techniques obtain significant energy savings on both processors due to online re-evaluation of DVS policies. For example, for k = 3 and benchmark CNC, JFTT consumes more energy than JFTC (18.5mJ > 17.6mJ) while T-DVS consumes only 9.5 mJ since average number of runtime faults per task instance ranged from 0 to 1. Note that energy savings reported here correspond only to online variations in fault occurrences and overall savings will be much higher if variations in task execution times are also taken into account.

Time complexity of proposed techniques is very small. For example, CNC benchmark consists of 6 independent tasks whose execution times vary from 35 to 720 µS and periods vary from 2400 to 4800 µS. Therefore, n = 8, R = 4800/2400 = 2, and L = 3 for XScale and 5 for Crusoe.

6. Conclusion

We have presented efficient scheduling algorithms that combine feasibility analysis and DVS based on exact characterization of RMA and can dynamically adapt to runtime behavior of tasks and fault occurrences to minimize energy consumption. Proposed online DVS policy re-evaluation schemes are low-cost and can save up to 60% more energy compared to offline scheduling algorithms. As part of future work, proposed techniques will be extended to task sets with dependent tasks and non-critical phasing.

7. Reference

[1] I. Akyildiz, W Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: A survey," *IEEE Communications Magazine*, 2002

[2] S. Reinhardt and S. Mukherjee, "Transient fault detection via simultaneous multithreading," *ACM SIGARCH Computer Architecture News*, 2000

[3] A. Chandrakasan, S. Sheng and R. Brodersen, "Low-power CMOS digital design," *IEEE Journal of Solid-State Circuits*, Vol. 27(4). Apr 1992

[4] J. Lorch and A. J. Smith, "Software strategies for portable computer energy management," *IEEE Personal Communication Magazine*, Vol. 5 (3), pp. 60-73, Jun 1998

[5] L Benini, A Bogliolo, and G. Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on VLSI Systems*, Vol. 8(3), Jun 2000

[6] K. Shin and Y. Lee, "Error detection process-model, design and its impact on computer performance," *IEEE Transactions on Computers*, Vol. C(33), pp. 529-540, Jun 1984

[7] K. Chandy, J. Browne, C. Dissly and W. Uhrig, "Analytic models for rollback and recovery strategies in data base systems," *IEEE Transactions on Software Engineering*, Vol. 1, pp. 100-110, Mar 1975

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

[9] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proceedings, ISLPED,* Aug 1998

[10] Y. Shin, K. Choi and T. Sakurai, "Power optimization of real time embedded systems on variable speed processors," *Proceedings, ICCAD,* pp. 365-368, Jun 2000

[11] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," *Proceedings, DAC,* pp. 828-833, Jun 2001

[12] K. Shin, T. Lin and Y. Lee, "Optimal checkpointing of real-time tasks," *IEEE Transactions on Computers*, Vol. 36(11), pp. 1328-1341, Nov 1987

[13] A. Ziv and J. Bruck, "An on-line algorithm for checkpoint placement," *IEEE Transactions on Computers*, Vol. 46(9), pp. 976-985, Sep 1997

[14] S. Kwak, B. Choi and B. Kim, "An optimal checkpointing strategy for real-time control systems under transient faults," *IEEE Transactions on Reliability*, Vol. 50(3), pp. 293-301, Sep 2001

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

[16] 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

[17] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," *Proceedings, DATE,* pp. 918-923, 2003

[18] 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(2), pp. 217-231, Feb 2004

[19] Y. Zhang and K. Chakrabarty, "Energy-aware fault tolerance in fixed-priority real-time embedded systems," *Proceedings, ICCAD,* pp. 209-214, 2003

[20] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of the ACM*, Vol. 20(1), pp. 46-61, 1973

[21] J. Lehoczky, L. Sha and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," *Proceedings, IEEE Real-Time Systems Symposium*, pp. 166-171, 1989

[22] Y. Zhang and K. Chakrabarty, "Task feasibility analysis and dynamic voltage scaling in fault-tolerant real-time embedded systems," *Proceedings, DATE,* Vol. 2, 2004

[23] T. Cormen, C. Leiserson, R. Rivest and C. Stein, *Introduction to Algorithm,* The MIT Press, 2001

[24] J. Rabaey, A. Chandrakasan and B. Nikolic, *Digital Integrated Circuits: a Design Perspective*, Prentice Hall, 2002

[25] Y.Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," *Proceedings, DAC,* pp. 134-139, 1999

[26] A. Bruns, K. Tindell and A. Sellings, "Effective analysis for engineering real-time fixed priority schedulers," *IEEE Transactions on Software Engineering*, Vol. 21(5), May 1995