Appeared in: Proceedings of the 18th 18th Euromicro Conference on Real-Time Systems
(ECRTS 06), Dresden, Germany, July 2006, pp. 161-170.

Reactive Speed Control in Temperature-Constrained Real-Time Systems

Shengquan Wang and Riccardo Bettati
Texas A&M University
College Station, TX 77840, USA
{swang,bettati}@cs.tamu.edu

Abstract

In this paper, we study temperature-constrained real-time systems, where real-time guarantees must be met without exceeding safe temperature levels within the processor. We give a short review on temperature issues in processors and describe how speed control can be used to trade-off task delays against processor temperature. In this paper, we describe how traditional worst-case execution scenarios do not apply in temperature-constrained situations. As example, we adopt a simple reactive speed control technique. We show how this simple reactive scheme can improve the processor utilization compared with any constant-speed scheme.

1 Introduction

In recent years, power density in processors has increased exponentially [1]. Due to this significantly high power density, processors are prone to overheating caused by the large energy consumption. It's generally assumed that over 50% of electronic failures are temperature-related, as the circuit reliability is exponentially dependent on the operating temperature [2]. Temperature therefore is becoming one of the big concerns in system design. For example, the Pentium Extreme Edition processor requires a thermal solution to maintain safe temperatures levels [3].

Naturally, power management plays a key role in maintaining temperature. There is an extensive literature on power management on processors both in general-purpose and embedded applications. However, the majority of this literature focuses on power management for the purpose of saving energy, *not* for maintaining safe temperature levels [4, 5, 6, 7, 8, 9, 10]. While energy and temperature are closely related, power control mechanisms for energy and temperature are quite different. It has been shown that many energy-saving techniques do not work well in reducing peak temperature [11, 12, 13]. This is due to the fact that energy-aware techniques focus on dealing with the *average* power consumption while temperature-aware ones focus on handling *peak* power consumption [13].

Both in research and practice, dynamic speed scaling [1] is one of the major techniques for power management. By dynamically changing the speed of the processor, speed scaling can control the power in the processor to save energy or reduce temperature. Many current microprocessors for general-purpose or embedded applications allow for various forms of dynamic speed scaling [12, 13] for power management.

In this paper, we study temperature-constrained real-time systems. In this kind of systems, we have two major constraints: the *delay* constraint for jobs and the *temperature* constraint for the processor. Dynamic speed scaling allows for a trade-off between these two performance metrics: To meet the delay constraint (or deadline), we run the processor at a higher speed; To maintain the safe temperature levels, we run the process at a lower speed.

The work on dynamic speed scaling techniques to control temperature in real-time systems was initiated in [12] and further investigated in [13]. Both [12] and [13] focus on online algorithms in real-time systems, where the scheduler learns about a task only at its release time. In contrast, in our work we assume a periodic-task model. We distinguish between proactive and reactive speed scaling schemes. Whenever the temperature model is known, the scheduler could in principle use a *proactive* speed-scaling approach, where – similarly to a non-work-conserving scheduler – resources are preserved for future use. In this paper, we limit ourselves to *reactive* schemes, and propose a simple reactive speed scaling technique for the processor, which will be discussed in Section 2. We focus on reactive schemes primarily because several current Pentium and IBM Power-PC processors support Dynamic Thermal Management in form of alerts, which are triggered whenever safe processor temperatures are exceeded [14, 15].

One of the keys to perform design-time schedulability

[1] At the risk of overly generalizing, we use the term "dynamic speed scaling" to subsume dynamic voltage scaling or dynamic frequency scaling.

tests in real-time systems is critical-instance analysis. Due to the additional temperature constraint, the critical instance analysis will be different from the traditional one, and we will address this in Section 3. Then in Section 4, we design the methodology to perform schedulability analysis for identical-periodic-task model in real-time systems with reactive speed scaling. We measure its performance in Section 5. Finally, we conclude our work with final remarks and give an outlook on future work in Section 6.

2 Model

2.1 Temperature Formula

In the following, we will be using an arguably simplistic thermal model of the processor [13, 16]. More accurate models can be derived from this simple one by more closely modeling the power dissipation (such as fan's) or by augmenting the input power by a stochastic component, etc. We apply the same approach previously used in [12, 13] and adopt Fourier's Law of heat conduction. Fourier's Law of heat conduction states that the rate of cooling is proportional to the difference in temperature between the object and the environment. We assume that the environment has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. We define $T(t)$ and $P(t)$ as the temperature and the power at time t, respectively. Then we can formulate the Fourier's Law as the following formula [12, 13]:

$$T'(t) = P(t) - bT(t), \tag{1}$$

where b is a positive constant that represents the power dissipation rate.

We define $s(t)$ as the *processor speed* at time t. The power consumption of an processor is a strictly increasing convex function of the speed [12]. Most of work in the literature assumes that power and processor speed are related as follows [12]:

$$P(t) = as^{\alpha}(t), \tag{2}$$

for some constant a and $\alpha > 1$. Usually, it is assumed that $\alpha = 3$ [12, 13].

We assume that the initial temperature is T_0, i.e., $T(0) = T_0$. It is reasonable to assume that $T_0 = 0$ (The initial temperature is the ambient one). Equation (1) is a classic linear differential equation. Together with (2), the solution to (1) can be expressed as

$$T(t) = \int_{t_0}^{t} P(\tau)e^{-b(t-\tau)}d\tau + T_0e^{-b(t-t_0)} \tag{3}$$

$$= \int_{t_0}^{t} as^{\alpha}(\tau)e^{-b(t-\tau)}d\tau + T_0e^{-b(t-t_0)}. \tag{4}$$

For the change of temperature, we observe that at any time point t, there are two possibilities for the change of temperature:

- Temperature is non-decreasing: By (1), we have $as^{\alpha}(t) - bT(t) \geq 0$, i.e.,

$$s(t) \geq \left(\frac{bT(t)}{a}\right)^{\frac{1}{\alpha}}. \tag{5}$$

- Temperature is non-increasing: We have

$$s(t) \leq \left(\frac{bT(t)}{a}\right)^{\frac{1}{\alpha}}. \tag{6}$$

We observe therefore that we can always appropriately scale the speed to change the temperature in the desired direction.

We also make the following observations about keeping either the temperature or the speed constant:

- If we want to keep the temperature constant at a value T_C during time interval $[t_0, t_1]$, then for any $t \in [t_0, t_1]$, we set

$$s(t) = \left(\frac{bT_C}{a}\right)^{\frac{1}{\alpha}}. \tag{7}$$

- If, on the other hand, we keep the speed constant at $s(t) = s_C$ during the same interval, then the temperature develops as follows:

$$T(t) = \frac{as_0^{\alpha}}{b} + (T(t_0) - \frac{as_C^{\alpha}}{b})e^{-b(t-t_0)}. \tag{8}$$

This relation between processor speed and temperature are the basis for any speed scaling scheme.

2.2 Speed Scaling

The goal of temperature control is to maintain the processor temperature within a safe operating range, and not exceed what we call the *highest-temperature threshold* T_H. Temperature control must ensure that

$$T(t) \leq T_H. \tag{9}$$

Also, the processor speed is limited by some maximum speed s_H, i.e.,

$$0 \leq s(t) \leq s_H. \tag{10}$$

2.2.1 Constant-Speed Scaling

A simple speed-scaling technique would consist in having the processor run at some constant speed such that the temperature never exceeds the threshold T_H. In other words, we set $s(t) = s_C$, where s_C is constant. By Fourier's Law as expressed in (1), the temperature will first increase at the rate $T'(t) = as_C^\alpha - bT(t) > 0$ until $T'(t) = as_C^\alpha - bT(t) = 0$. At this point, some maximal temperature T_M (not necessarily T_H) is reached, and the temperature will remain constant from then on. For a given maximal temperature T_M, we have $s_C = (\frac{b}{a}T_M)^{\frac{1}{\alpha}}$. In our system, we have to ensure that the temperature stays within a safe range, that is, $T_M \leq T_H$ always holds. This bounds s_C as follows: $s_C \leq (\frac{b}{a}T_H)^{\frac{1}{\alpha}}$. If we pick $T_M = T_H$, then we define the *equilibrium speed* s_E such that

$$s_E = (\frac{b}{a}T_H)^{\frac{1}{\alpha}}. \tag{11}$$

We also assume that $s_E \leq s_H$ (otherwise the processor will never run fast enough to hit T_H, and temperature is not an issue any more). The speed s_E is the maximum speed for constant-speed scaling that maintains the temperature within the safe operating range.

2.2.2 Reactive Speed Scaling

The primary disadvantage of constant-speed scaling is that the scheme does not take advantage of the power dissipation during idle times. A better scheme would make use of periods where the processor is "cool", typically after idle periods, to dynamically scale the speed and temporarily execute tasks at speeds higher than s_C.

As a result, dynamic speed scaling would be used to improve the overall processor utilization. We adopt a fully *reactive speed-scaling* technique:

> The processor will run at full speed s_H when there is backlogged workload and the temperature is below the threshold T_H. Otherwise, we perform the following speed scaling actions: Whenever the backlogged workload is empty, the processor idles (runs at the zero speed); Whenever the temperature hits T_H, the processor will run at the equilibrium speed s_E, which is defined in (11).

If we define $W(t)$ as the backlogged workload at time t, the speed scaling scheme described before can be expressed using the following formula:

$$s(t) = \begin{cases} s_H, & (W(t) > 0) \wedge (T(t) < T_H) \\ s_E, & (W(t) > 0) \wedge (T(t) = T_H) \\ 0, & W(t) = 0 \end{cases} \tag{12}$$

It is easy to show that in any case the temperature never exceeds the threshold T_H. By using the full speed sometime, we aim to improve the processor utilization compared with the constant-speed scaling. The reactive speed scaling is very simple: whenever the temperature reaches the threshold, an event is triggered by the thermal monitor, and the system throttles the processor speed.

In this paper, we will analyze the effectiveness of reactive speed scaling and use constant-speed scaling as the baseline for performance comparison.

2.3 Task Model and Scheduler

We consider a set of *periodic tasks* $\{\Gamma_i : i = 1, 2, \ldots, n\}$, where task $\Gamma_i = (P_i, C_i)$ has a minimum time period P_i between jobs and each job requires C_i processor cycles to complete in the worst case. We also consider a fixed-priority scheduling algorithm, i.e., all jobs in a task are assigned the same fixed priority. Figure 1 shows an example of a single periodic task system that uses reactive speed scaling.

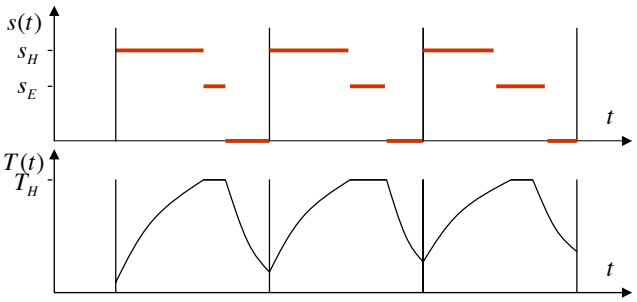


Figure 1. An Illustration of A Periodic Task with Reactive Speed Scaling

In order to determine the worst-case delay for jobs in tasks, we must first identify the worst-case arrival pattern, also called *critical instance*. Due to the additional temperature constraint, the analysis of the critical instance is different from the traditional one. In the following section, we describe how processor temperature affects the critical instance, in general, and how the critical instance can be determined for the case of reactive speed scaling.

3 Critical Instance

A *critical instance* of a task Γ_i is a time instance which is such that the job in Γ_i released at the instance has the maximum response time of all jobs in Γ_i [17]. In a fully preemptable system with reactive speed scaling, we notice that there are two factors that affect the critical instance:

- How many high-priority jobs will preempt this job? This is same as in the traditional critical-instance analysis.

- What is the temperature at this time instance? With any speed scaling, once the temperature hits the threshold, the speed will drop to no higher than the equilibrium one. Therefore, the response time of a job is affected by the temperature at its arrival. If we have a higher temperature at this time instance, we will have a longer response time.

Therefore, our goal in the critical-instance analysis is to obtain the worst-case preemption from the high-priority tasks *and* the maximal temperature at this time instance.

In the following, we will base on the traditional critical-instance analysis to derive the critical instance in the temperature-constrained case. First we introduce the following lemma, which states that the temperature for the critical instance always raises when the last job before the critical instance gets delayed.

Lemma 1 *Given a time instance t, we consider a successive execution part of job with a starting time t_0 and the completion time t_1, where $t_0 < t$ and $t_1 < t$. We assume the system is idle during $[t_1, t]$. Define T_t as the temperature at t. If we shift this part of job into a starting time t_0^* and a completion time t_1^* such that $t_0 < t_0^* < t$ and $t_1 < t_1^* < t$ as shown in Figure 2. Define T_t^* as the new temperature at t. Then we have $T_t \leq T_t^*$.*

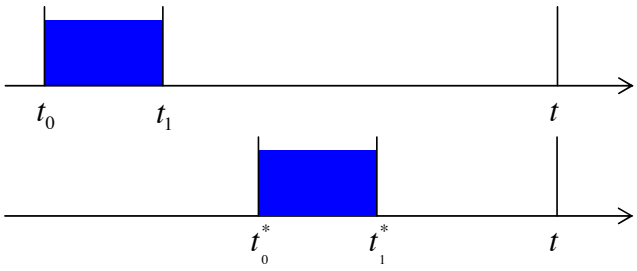


Figure 2. An example of a shifted successive execution part of job

Proof: We assume the fixed temperature $T(t_0)$ at t_0. There are four cases:

- Case 1: The temperature will neither hit T_H during $[t_0^*, t_1^*]$ for the job-shifted scenario nor during $[t_0, t_1]$ for the original scenario. Then, we have [2]

$$\begin{aligned} T(t_0^*) &= T(t_0)e^{-b(t_0^*-t_0)}, && (13)\\ T(t_1^*) &= \frac{as_H^\alpha}{b}(1-e^{-b(t_1^*-t_0^*)}) \\ &\quad +T(t_0^*)e^{-b(t_1^*-t_0^*)}, && (14)\\ T_t^* &= T(t_1^*)e^{-b(t-t_1^*)}. && (15)\end{aligned}$$

Hence,

$$\begin{aligned} T_t^* &= \frac{as_H^\alpha}{b}(1-e^{-b(t_1^*-t_0^*)})e^{-b(t-t_1^*)} \\ &\quad +T(t_0)e^{-b(t-t_0)}. && (16)\end{aligned}$$

Similarly, we have

$$\begin{aligned} T_t &= \frac{as_H^\alpha}{b}(1-e^{-b(t_1-t_0)})e^{-b(t-t_1)} \\ &\quad +T(t_0)e^{-b(t-t_0)}. && (17)\end{aligned}$$

Since $t_1^* - t_0^*$ is fixed and $t_0 \leq t_0^*$, then $t_1 \leq t_1^*$. Therefore, we have $T_t \leq T_t^*$.

- Case 2: The temperature will hit T_H at t_H^* during $[t_0^*, t_1^*]$ for the job-shifted scenario and also hit T_H in $[t_0, t_1]$ for the original scenario. Then, we have

$$\begin{aligned} T(t_0^*) &= T(t_0)e^{-b(t_0^*-t_0)}, && (18)\\ T(t_H^*) &= \frac{as_H^\alpha}{b}(1-e^{-b(t_H^*-t_0^*)}) \\ &\quad +T(t_0^*)e^{-b(t_H^*-t_0^*)} && (19)\\ &= T_H, && (20)\\ T(t_1^*) &= T_H, && (21)\\ T_t^* &= T(t_1^*)e^{-b(t-t_1^*)}. && (22)\end{aligned}$$

Hence,

$$t_H^* = -\frac{1}{b}\ln\frac{T_H - \frac{as_H^\alpha}{b}}{T(t_0)e^{bt_0} - \frac{as_H^\alpha}{b}e^{bt_0^*}}. \quad (23)$$

Define $C = s_H(t_H^* - t_0^*) + s_E(t_1^* - t_H^*)$, which is fixed. Hence,

$$t_1^* = \frac{C}{s_E} + \frac{s_H}{s_E}t_0^* - (\frac{s_H}{s_E} - 1)t_H^*. \quad (24)$$

By (22), (23), and (24), we have

$$\begin{aligned} T_t^* &= T_H e^{-b(t-\frac{C}{s_E}-\frac{s_H}{s_E}t_0^*)} \\ &\quad (\frac{T_H - \frac{as_H^\alpha}{b}}{T(t_0)e^{bt_0} - \frac{as_H^\alpha}{b}e^{bt_0^*}})^{\frac{s_H}{s_E}-1}. && (25)\end{aligned}$$

[2] In the following, the temperature calculation is based on Equation (8).

Similarly, we have

$$T_t = T_H e^{-b(t-\frac{C}{s_E}-\frac{s_H}{s_E}t_0)} \left(\frac{T_H - \frac{as_H^\alpha}{b}}{T(t_0)e^{bt_0} - \frac{as_H^\alpha}{b}e^{bt_0}}\right)^{\frac{s_H}{s_E}-1}. \quad (26)$$

Since $t_0 \leq t_0^*$, we have $T_t \leq T_t^*$.

- Case 3: The temperature will not hit T_H during $[t_0^*, t_1^*]$ for the job-shifted scenario but hit T_H in $[t_0, t_1]$ for the original scenario.

  In this case, we introduce a transition scenario that the job is executed during $[t_0^{**}, t_1^{**}]$, where $t_0 \leq t_0^{**} \leq t_0^*$, $t_1 \leq t_1^{**} \leq t_1^*$, and the temperature hits T_H just at t_1^{**} in this scenario. The existence of this scenario is obvious. Define T_t^{**} as the temperature at t in this scenario.

  Since the temperature will hit T_H at the boundary t_1^{**} in the transition scenario, we can treat it as the scenario that the temperature will hit T_H either during the execution time or after the execution time. Therefore, we can apply either of the temperature analysis in Cases 1 and 2 to this scenario.

  First, we compare the original scenario with the transition scenario. We apply the temperature analysis in Case 2 to both scenarios. Following the result of Case 2, we have

  $$T_t \leq T_t^{**}. \quad (27)$$

  Second, we compare the transition scenario with the job-shifted scenario. We apply the temperature analysis in Case 1 to both scenarios. Following the result of Case 1, we have

  $$T_t^{**} \leq T_t^*. \quad (28)$$

  Therefore, by (27) and (28), we have $T_t \leq T_t^*$.

- Case 4: The temperature will hit T_H during $[t_0^*, t_1^*]$ for the job-shifted scenario but not hit T_H during $[t_0, t_1]$ for the original scenario. Since $t_0 \leq t_0^*$, by (18), we have $T_0 \geq T_0^*$. It is impossible to hit T_H during $[t_0^*, t_1^*]$ and not hit T_H during $[t_0, t_1]$. This case will never happen.

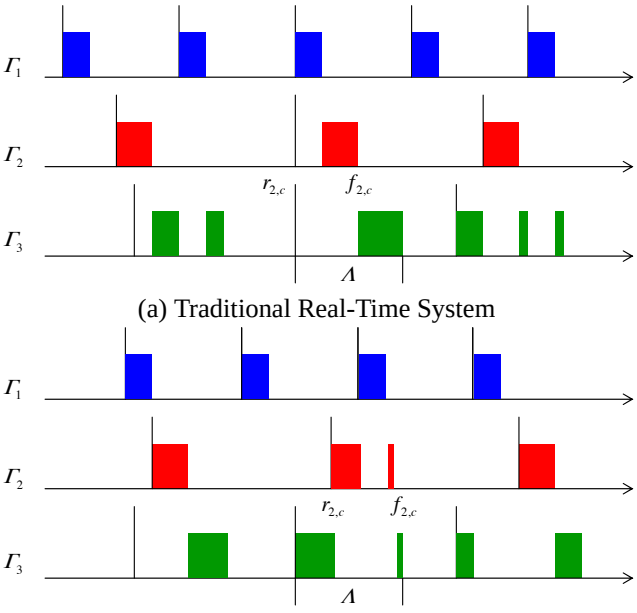
In all possible cases, we have $T_t \leq T_t^*$. ■

In traditional critical-instance analysis in systems without blocking, the critical instance of a task Γ_i occurs when one of its job $J_{i,c}$ is released at the same time $r_{i,c}$ with a job in every higher-priority task. With the additional temperature constraint, we must consider not only the worst-case preemption by the high-priority tasks, but also the worst-case temperature at this time instance, which is caused by the jobs of *all* tasks *before* this time instance.

We consider the maximum busy interval Λ beginning with the critical instance for all tasks in a traditional real-time system without blocking. Based on this maximum busy interval, we can obtain the critical instance in our temperature-constrained real-time system as described in the following theorem:

Theorem 1 *Assume that the tasks in a task system are phased so that the last job of each task during the busy interval Λ is completed a sufficiently-small time interval ϵ before the completion time of the last job of the next lower-priority task during the busy interval Λ. The release time of the first job of each task during Λ will be a critical instance when the temperature at the beginning of Λ is maximized.*

The time interval of length ϵ is used to preempt the low-priority tasks as late as possible. The execution time for this short part of the job can be neglected. Figure 3 shows an example of a critical instance for a three-task system in both the traditional and our temperature-constrained real-time system.



(a) Traditional Real-Time System

(b) Temperature-Constrained Real-Time System

Figure 3. An Illustration of A Crical Instance

Proof: In the sketch of the proof, we make use of Lemma 1. If we shift a task such that its last job during the busy interval Λ is completed ϵ time unit before the completion time of the last job of its next higher-priority during the busy interval Λ, then we will not change the worst-case preemption by the high-priority tasks. However, with the

shifted task, more jobs should be pushed forward before the critical time instance of each task. Then by Lemma 1, the initial temperature will be raised. The release time of the first job of each task during Λ will be a critical instance when the temperature at the beginning of Λ is maximized. ■

4 Identical-Periodic Tasks

We first consider a set of *identical-periodic tasks* $\{\Gamma_i: i = 1, 2, \ldots, n\}$, where $\Gamma_i = (P, C_i)$. We adopt a fixed-priority scheduling scheme, and Γ_i is assigned priority i (the smaller the index, the higher the priority).

In order to perform the schedulability test, we first consider the critical instance for each task. Figure 4 shows an example of the critical instance for Task Γ_2 in a three-task-identical-period system.

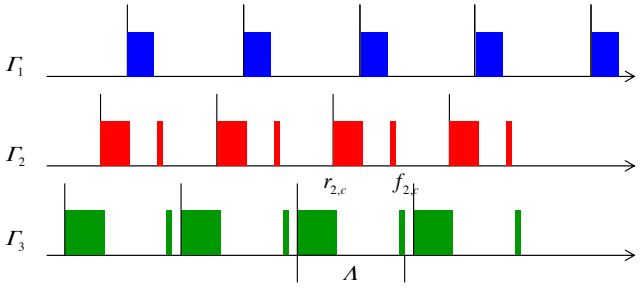


Figure 4. An Illustration of Identical Periodic Tasks

By Theorem 1, the jobs of lower-priority tasks arrive first during a busy interval. We can form a single task system, which has a periodic task with period P and processor cycles $C = \sum_{i=1}^{n} C_i$ for each job. First, we compute the temperature at the release time of each task job. We assume the temperature will hit T_H sometime (otherwise, the case at hand is not interesting).

As shown in Figure 5, we consider a job J_k with a release time $t_{k,0}$ and completion time $t_{k,1}$. The temperature will hit T_H at time $t_{k,H}$. First we obtain the time-instance formulas

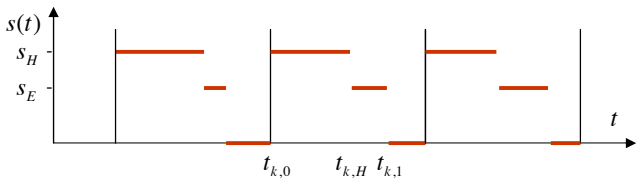


Figure 5. A Illustration of a Periodic Task under Reactive Speed Scaling

for $t_{k,0}$, $t_{k,H}$, and $t_{k,1}$ [3]:

$$t_{k,0} = kP, \tag{29}$$

$$t_{k,H} = \frac{1}{b}\ln\frac{\frac{as_H^\alpha}{b} - T_{k,0}}{\frac{as_H^\alpha}{b} - T_{k,H}} + t_{k,0}, \tag{30}$$

$$t_{k,1} = \frac{s_H}{s_E}(t_{k,0} + \frac{C}{s_H}) - (\frac{s_H}{s_E} - 1)t_{k,H}. \tag{31}$$

Then we obtain the temperature at each time instance as follows:

$$T_{k,0} = T_{k-1,1}e^{-b(t_{k,0}-t_{k-1,1})}, \tag{32}$$

$$T_{k,H} = T_H, \tag{33}$$

$$T_{k,1} = T_H. \tag{34}$$

Based on the above formulas, we define

$$\pi_{k,0H} = t_{k,H} - t_{k,0} = \frac{1}{b}\ln\frac{\frac{as_H^\alpha}{b} - T_{k,0}}{\frac{as_H^\alpha}{b} - T_H}, \tag{35}$$

$$\pi_{k,H1} = t_{k,1} - t_{k,H} = \frac{s_H}{s_E}(\frac{C}{s_H} - \pi_{k,0H}), \tag{36}$$

$$\pi_{k,01} = t_{k,1} - t_{k,0} = \frac{C}{s_E} + (1 - \frac{s_H}{s_E})\pi_{k,0H}, \tag{37}$$

and

$$\pi_{k-1,10} = t_{k,0} - t_{k-1,1} \tag{38}$$

$$= (P - \frac{C}{s_E}) + \frac{1}{b}(\frac{s_H}{s_E} - 1) \ln\frac{\frac{as_H^\alpha}{b} - T_{k-1,0}}{\frac{as_H^\alpha}{b} - T_H}. \tag{39}$$

By (32), (39), and $T_H = \frac{as_E^\alpha}{b}$, we have

$$\frac{T_{k,0}}{T_H} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-1,0}}{T_H}}{(\frac{s_H}{s_E})^\alpha - 1}\right)^{1-\frac{s_H}{s_E}} \exp\left(-b(P - \frac{C}{s_E})\right). \tag{40}$$

Then,

$$\frac{T_{k-1,0}}{T_{k,0}} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-2,0}}{T_H}}{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-1,0}}{T_H}}\right)^{1-\frac{s_H}{s_E}}. \tag{41}$$

It is easy to show that if $T_{k-2,0} \le T_{k-1,0}$, then $T_{k-1,0} \le T_{k,0}$. Also we know $T(2,0) \ge T(1,0) \ge T(0,0) = 0$. Therefore, by induction, $T_{k,0}$ is increasing in terms of k.

[3] In the following equation, for simplicity, we denote $T_{x,y} = T(t_{x,y})$.

Next, we want to obtain $\lim_{k\to\infty} T_{k,0}$ by (40), which is the maximum of all $T_{k,0}$'s. As $k \to \infty$, we have the fixed point $T^* = \lim_{k\to\infty} T_{k,0}$ by the following equation

$$\frac{T^*}{T_H} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T^*}{T_H}}{(\frac{s_H}{s_E})^\alpha - 1}\right)^{1-\frac{s_H}{s_E}} \exp\left(-b(P - \frac{1}{s_E}\sum_{i=1}^{n} C_i)\right). \quad (42)$$

Figure 6 shows the relationship between $\frac{T^*}{T_H}$ and P, where T^* is obtained as the fixed point by (42):

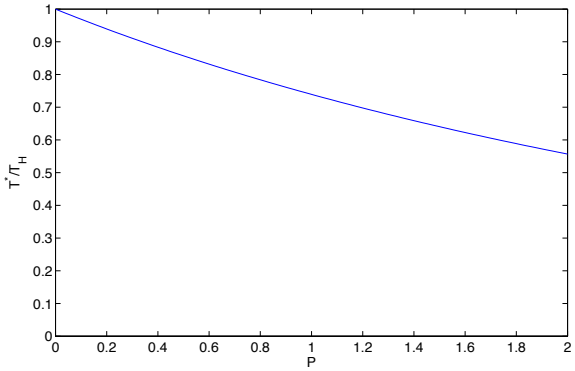


Figure 6. $\alpha = 3, a = b = 1, \frac{s_H}{s_E} = 1.5, \frac{\frac{C}{s_H}}{P} = 0.5$

Now, we go back to original identical-periodic-task set. T^* is the maximum temperature at the beginning of a busy period. Based on this, in the following, we want to obtain the temperature at the critical instance of each task. We have the following lemma:

Lemma 2 *Let T_i^* denote the temperature at the critical instance of task Γ_i, then $\frac{T_i^*}{T_H}$ can be expressed by the following formula:*

$$\frac{T_i^*}{T_H} = \min\{1, (\frac{s_H}{s_E})^\alpha + (\frac{T^*}{T_H} - (\frac{s_H}{s_E})^\alpha) e^{-\frac{b}{s_H}\sum_{j>i} C_j}\}. \quad (43)$$

Proof: At the critical instance $r_{i,c}$, the jobs of lower-priority task will be aligned back-to-back before $r_{i,c}$.

If the temperature does not hit T_H at $r_{i,c}$, i.e., $T_i^* < T_H$, then by (8), we have

$$T_i^* = \frac{as_H^\alpha}{b} + (T^* - \frac{as_H^\alpha}{b}) e^{-\frac{b}{s_H}\sum_{j>i} C_j}. \quad (44)$$

By (11), we have

$$\frac{T_i^*}{T_H} = (\frac{s_H}{s_E})^\alpha + (\frac{T^*}{T_H} - (\frac{s_H}{s_E})^\alpha) e^{-\frac{b}{s_H}\sum_{j>i} C_j}. \quad (45)$$

Furthermore, by $T_i^* < T_H$, we have

$$\frac{T^*}{T_H} < (\frac{s_H}{s_E})^\alpha + (1 - (\frac{s_H}{s_E})^\alpha) e^{\frac{b}{s_H}\sum_{j>i} C_j}. \quad (46)$$

If the temperature hit T_H before $r_{i,c}$, then we have $T_i^* = T_H$, and the lemma is proved. ■

Now we consider the response time $d_{i,c}$ for the instance $J_{i,c}$. If $T_i^* = T_H$, we have

$$d_{i,c} = \frac{1}{s_E}\sum_{j\le i} C_j. \quad (47)$$

Otherwise

$$d_{i,c} = \lim_{k\to\infty} \pi_{k,01} - \frac{1}{s_H}\sum_{j>i} C_j. \quad (48)$$

By (32), we have

$$\lim_{k\to\infty} \pi_{k,01} = P - \lim_{k\to\infty}(t_{k,0} - t_{k-1,1}) \quad (49)$$

$$= P + \frac{1}{b}\lim_{k\to\infty} \ln\frac{T_{k,0}}{T_H} \quad (50)$$

$$= P + \frac{1}{b}\ln\frac{T^*}{T_H}. \quad (51)$$

This gives rise to the following theorem that bounds the worst-case delay:

Theorem 2 *The worst-case delay d_i^{RSS} experienced by a job in task Γ_i under reactive speed scaling can be bounded as follows:*

- *If $\frac{T^*}{T_H} < (\frac{s_H}{s_E})^\alpha + (1 - (\frac{s_H}{s_E})^\alpha) e^{\frac{b}{s_H}\sum_{j>i} C_j}$, then*

$$d_i^{RSS} \le P + \frac{1}{b}\ln\frac{T^*}{T_H} - \frac{1}{s_H}\sum_{j>i} C_j; \quad (52)$$

- *Otherwise*

$$d_i^{RSS} \le \frac{1}{s_E}\sum_{j\le i} C_j. \quad (53)$$

In comparison, for the constant-speed scaling technique, it is easy to show that the worst-case delay d_i^{SSS} experienced by a job in task Γ_i can be bounded as

$$d_i^{SSS} \le \frac{1}{s_E}\sum_{j\le i} C_j. \quad (54)$$

Corollary 1 *If the deadline of task $D_i = \delta P$, where $0 < \delta \le 1$, then under reactive speed scaling we have the worst-case delay d for any job in all n tasks bounded as follows:*

$$d^{RSS} \le P + \frac{1}{b}\ln\frac{T^*}{T_H}, \quad (55)$$

when $\frac{1}{s_E}\sum_{i=1}^{n} C_i \le P$.

Proof: Since task Γ_n will experience the maximum delay, we have $d^{RSS} = d_n^{RSS}$. Then by Theorem 2, we have

$$d^{RSS} \leq \begin{cases} P + \frac{1}{b} \ln \frac{T^*}{T_H}, & \frac{T^*}{T_H} < 1 \\ \frac{1}{s_E} \sum_{i \leq n} C_i, & \frac{T^*}{T_H} = 1 \end{cases} \quad (56)$$

By (42), $\frac{T^*}{T_H} < 1$ means $\frac{1}{s_E} \sum_{i \leq n} C_i < P$ and $\frac{T^*}{T_H} = 1$ means $\frac{1}{s_E} \sum_{i \leq n} C_i = P$. Then the corollary is proved. ■

Furthermore, if we define the processor utilization as $U = \sum_{i=1}^{n} \frac{\frac{C_i}{s_H}}{P}$, then we have the following corollary:

Corollary 2 *The maximum schedulable utilization U^{RSS} for the tasks under reactive speed scaling can be expressed as*

$$U^{RSS} = \frac{s_E}{s_H} \xi, \quad (57)$$

where

$$\xi = \min\{1, \delta + (\frac{s_H}{s_E} - 1)\frac{1}{bP} \ln \left(\frac{(\frac{s_H}{s_E})^\alpha - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^\alpha - 1} \right)\}. \quad (58)$$

Proof: By Corollary 1, when $\frac{1}{s_E} \sum_{i=1}^{n} C_i \leq P$, i.e.,

$$U \leq \frac{s_E}{s_H}, \quad (59)$$

and $P + \frac{1}{b} \ln \frac{T^*}{T_H} \leq \delta P$, i.e.,

$$\frac{T^*}{T_H} \leq e^{-b(1-\delta)P}. \quad (60)$$

Then, by (42), we have

$$\frac{1}{s_E} \sum_{i=1}^{n} C_i \leq \delta P + \frac{1}{b}(\frac{s_H}{s_E} - 1) \ln \left(\frac{(\frac{s_H}{s_E})^\alpha - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^\alpha - 1} \right). \quad (61)$$

Therefore,

$$U \leq \frac{s_E}{s_H}\delta + \frac{s_E}{s_H}(\frac{s_H}{s_E} - 1)\frac{1}{bP} \ln \left(\frac{(\frac{s_H}{s_E})^\alpha - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^\alpha - 1} \right). \quad (62)$$

By (59) and (62), if we define ξ as (58), the corollary is proved. ■

It is easy to show that under constant-speed scaling we have

$$\frac{1}{s_E} \sum_{i=1}^{n} C_i \leq \delta P. \quad (63)$$

The maximum schedulable utilization U^{SSS} for the tasks under constant speed scaling can be expressed as

$$U^{SSS} = \frac{s_E}{s_H} \delta. \quad (64)$$

Therefore, as $\delta < 1$ and $T^* < T_H$, the maximum schedulable utilization for the tasks under reactive speed scaling is larger than the one under constant-speed scaling.

5 Performance Evaluation

In this section, we compare the performance of reactive speed scaling with that of constant-speed scaling based on the theoretical results from Section 4. We use the *maximum schedulable utilization* (MSU) as the performance metric.

In our evaluation, we set $\alpha = 3$ and $a = b = 1$. By Corollary 2, we know that the MSU depends on δ, P, and $\frac{s_E}{s_H}$. In the following, we fix two of these parameters and measure how the MSU is affected by the other parameters. In each setting, we measure both constant-speed scaling and reactive speed scaling. Figure 7 shows all the performance

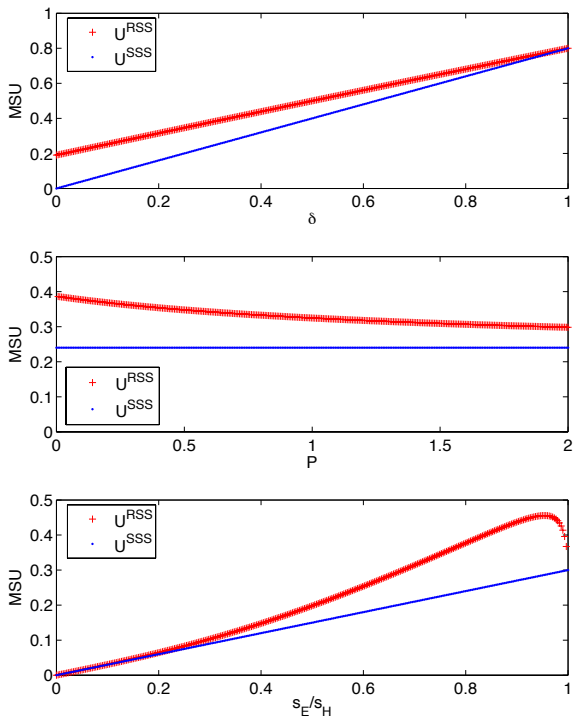


Figure 7. The Evaluation of the Maximum Schedulable Utilization

results. We notice that, in all cases, reactive speed scaling

achieves a higher MSU than constant-speed scaling. In the following, we explain the details of our results for each setting.

MSU vs. Deadline Ratio δ: We measure how the deadline constraint affects the MSU. We set $P = 0.1$ and $\frac{s_E}{s_H} = 0.8$. We vary δ from 0 to 1. When δ is smaller, U^{RSS} is much higher than U^{SSS}. This is because smaller δ's give rise to more idle time, which results in lower temperature at the beginning of the next busy period. Therefore the processor can run at the full speed for a longer duration under reactive speed scaling, i.e., the processor can achieve higher MSU. When $\delta = 1$, reactive speed scaling is no better than constant-speed scaling.

MSU vs. Period P: We measure how the value of period affects the MSU. We set $\delta = 0.3$ and $\frac{s_E}{s_H} = 0.8$. We vary P from 0 to 2. When P is smaller, U^{RSS} is higher than U^{SSS}. With smaller P, the workload will be smooth (not bursty), and the processor will tend to not heat up as much as for more bursty workload.

MSU vs. Speed Ratio $\frac{s_E}{s_H}$: We measure how the ratio between the equilibrium and maximum speed affects the MSU. We set $P = 0.1$ and $\delta = 0.3$. We vary $\frac{s_E}{s_H}$ from 0 to 1. The MSU is not monotonically changing with the speed ratio $\frac{s_E}{s_H}$ under reactive speed scaling. For low speed ratios the maximum speed causes the processor to heat up very quickly, and so has little benefit. On the other hand, when $s_H = s_E$, there is no benefit in using dynamic speed scaling.

6 Conclusion and Future Work

It is a well-known problem that increased per-node computation capabilities come at the cost of power, and power dissipation is particularly critical in dense packaging environments encountered in many critical systems. In this paper we describe a model for temperature-aware computation in real-time systems that is based on speed scaling. We describe a simple reactive speed-scaling scheme, which could easily be implemented using the thermal management facilities on many currently available microprocessors. We show how schedulability analysis schemes for traditional environments can be extended to account for speed scaling.

The current work needs to be extended in a number of directions, primarily schedulability analysis for more general task sets and proactive speed scaling schemes. We are currently working on formulating the critical-instance test for arbitrary-period task sets. Since the critical instance depends both on the interference by higher-priority tasks and the temperature at the critical instant, we must describe the location of busy intervals *before* the critical instant so as to maximize the response time.

Proactive speed scaling schemes come in a variety of ways. For example, the maximum speed can be defined per-task, with low-priority task being assigned low maximum speeds. At this point we don't yet understand the criteria needed to allocate speeds to tasks.

Acknowledgment

This work is being funded by the National Science Foundation under Grant CNS-0509483.

References

[1] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware computer systems: Opportunities and challenges," *IEEE Micro*, 2003.

[2] L.-T. Yeh and R. C. Chu, *Thermal Management of Microelectronic Equipment: Heat Transfer Theory, Analysis Methods, and Design Practices*, ASME Press, New York, NY, 2002.

[3] Intel, "Intel Pentium D Processor and Intel Pentium Processor Extreme Edition 840^{Δ} thermal and mechanical design guidelines," May 2005, http://download.intel.com/design/PentiumXE/designex/30683002.pdf.

[4] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM Symposium on Operating Systems Principles*, 2001.

[5] H. Aydin, R. Melhem, D. Moss, and P. M. Alvarez, "Dynamic and aggressive scheduling techniques for power-aware real-time systems," in *IEEE Real-Time Systems Symposium*, 2001.

[6] A. Qadi, S. Goddard, and S. Farritor, "A dynamic voltage scaling algorithm for sporadic tasks," in *IEEE Real-Time Systems Symposium*, 2003.

[7] Y. Liu and A. K. Mok, "An integrated approach for applying dynamic voltage scaling to hard real-time systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[8] S. Saewong and R. Rajkumar, "Practical voltage-scaling for fixed-priority rt-systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2003.

[9] G. Quan, L. Niu, X. S. Hu, and B. Mochocki, "Fixed priority scheduling for reducing overall energy on variable voltage processors," in *IEEE Real-Time Systems Symposium*, 2003.

[10] F. Zhang and S. T. Chanson, "Power-aware processor scheduling under average delay constraints," in *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2005.

[11] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture," in *International Symposium on Computer Architecture*, 2003.

[12] N. Bansal, T. Kimbrel, and K. Pruhs, "Dynamic speed scaling to manage energy and temperature," in *IEEE Syposium on Foundations of Computer Science*, 2004.

[13] N. Bansal and K. Pruhs, "Speed scaling to manage temperature," in *Symposium on Theoretical Aspects of Computer Science*, 2005.

[14] H. Sanchez, B. Kuttanna, T. Olson, M. Alexander, G. Gerosa, R. Philip, and J. Alvarez, "Thermal management system for high performance powerpc microprocessors," in *IEEE International Computer Conference*, 1997.

[15] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, "Analysis of thermal monitor features of the intel pentium m processor," in *Workshop on Temperature-aware Computer Systems*, 2004.

[16] J. E. Sergent and A. Krum, *Thermal Management Handbook*, McGraw-Hill, 1998.

[17] J. Liu, *Real-Time Systems*, Prentice Hall, New Jersey, 2000.