Real-Time Syst (2008) 39: 73–95
DOI 10.1007/s11241-007-9046-x

Reactive speed control in temperature-constrained real-time systems

Shengquan Wang · Riccardo Bettati

Published online: 8 December 2007

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. We develop a delay computation methodology that can be used in combination with a simple reactive speed control technique, and show how this simple reactive scheme can decrease the delay of tasks compared with any constant-speed scheme.

Keywords Speed control · Temperature · Real-time

1 Introduction

In recent years, power density in processors has increased exponentially (Skadron et al. 2003b). Processors are prone to overheating caused by the large energy consumption. It is generally assumed that over 50% of electronic failures are temperature-related, as the circuit reliability is highly dependent on the operating temperature (Yeh and Chu 2002). Temperature therefore is becoming one of the major concerns in system design. A variety of techniques are being investigated for thermal control at design time, for example through appropriate packaging, and

S. Wang (✉)
The University of Michigan, Dearborn, MI 48128, USA
e-mail: shqwang@umd.umich.edu

R. Bettati
Texas A&M University, College Station, TX 77843, USA
e-mail: bettati@cs.tamu.edu

at run time, through various forms of dynamic thermal management (DTM) (e.g., Brooks and Martonosi 2001).

It is expected that packaging-based thermal control solutions (for example improvements in airflow,) or active heat dissipation will become increasingly less effective, due to high peak power of upcoming processors and the extremely high power density in emerging systems-in-package technologies (Association 2005). In addition, many high-performance embedded devices have specific packaging requirements and generally operate in environments that make the use of appropriate packaging and of active dissipation based schemes impossible.

A number of *dynamic* thermal management approaches to control the temperature at run time have been proposed, ranging from clock throttling and clock gating in the Pentium 4 Series processors (Rotem et al. 2004a; Skadron et al. 2003a) to dynamic voltage scaling in a variety of modern processors, to various forms of in-chip load balancing, such as "local gating" (Skadron et al. 2003a) or the selective idling of one of the cores in the Pentium Core Duo architecture (Gochman et al. 2006).

There is an extensive literature on power management for 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 (Pillai and Shin 2001; Aydin et al. 2001; Qadi et al. 2003; Liu and Mok 2003; Saewong and Rajkumar 2003; Quan et al. 2003; Zhang and Chanson 2005). 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 (Skadron et al. 2003a; Bansal et al. 2004; Bansal and Pruhs 2005). 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 (Bansal and Pruhs 2005).

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 (Bansal et al. 2004; Bansal and Pruhs 2005) for power management.

In this paper, we study temperature-constrained real-time systems. In this kind of systems, we have two major constraints: *delay* constraints for jobs and *temperature* constraints for the processor. Dynamic speed scaling allows for a trade-off between these two performance metrics: To meet the delay constraints (or deadlines), 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 Bansal et al. (2004) and further investigated in Bansal and Pruhs (2005). Both Bansal et al. (2004) and Bansal and Pruhs (2005) focus on online algorithms in real-time systems, where the scheduler learns about a task only at its

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

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 Sect. 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 can be easily integrated into the ACPI power control framework (Sanchez et al. 1997; Rotem et al. 2004b).

One of the keys to perform design-time schedulability 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 Sect. 4. In Sect. 5, we describe a methodology to perform schedulability analysis for an identical-periodic-task model in real-time systems with reactive-speed scaling. We measure its performance in Sect. 6. Finally, we conclude our work with final remarks and give an outlook on future work in Sect. 7.

2 Models

2.1 Thermal model

A wide range of increasingly sophisticated thermal models for integrated circuits have been proposed in the last few years. Some are comparatively simple, chip-wide models, such as developed by Dhodapkar et al. (2000) in TEMPEST. Other models, such as used in *hotspot* (Skadron et al. 2003a), describe the thermal behavior at the granularity of architecture-level blocks or below, and so more accurately capture the effects of hotspots.

In this paper we will be using the popular and very simple chip-wide RC thermal model (Sergent and Krum 1998) previously used in Bansal et al. (2004), Bansal and Pruhs (2005), Dhodapkar et al. (2000), Cohen et al. (2003). While this model does not capture fine-granularity thermal effects, the authors in Skadron et al. (2003a) for example agree that it is appropriate for the investigation of chip-level techniques, such as speed-scaling. In addition, existing processors typically have well-defined hotspots, and accurate placement of sensors alleviates the need for fine-granularity temperature modeling. The Intel Core Duo processor family, for example, has a highly accurate digital thermometer placed at the single hotspot of each die, in addition to a single legacy thermal diode for both cores (Gochman et al. 2006).

We define $s(t)$ as the *processor speed (frequency)* at time t. Then the input power $P(t)$ at time t is usually represented as

$$P(t) = \kappa s^{\alpha}(t), \tag{1}$$

for some constant κ and $\alpha > 1$. Usually, it is assumed that $\alpha = 3$ (Bansal et al. 2004; Bansal and Pruhs 2005).

We assume that the ambient has a fixed temperature, and that temperature is scaled so that the ambient temperature is zero. We define $T(t)$ as the temperature at time t. We adopt Fourier's Law as shown in the following formula (Bansal et al. 2004; Bansal and Pruhs 2005; Cohen et al. 2003):

$$T'(t) = \frac{P(t)}{C_{th}} - \frac{T(t)}{R_{th}C_{th}}, \tag{2}$$

where R_{th} is the thermal resistance and C_{th} is the thermal capacitance of the chip. Applying (1) into (2), we have

$$T'(t) = as^{\alpha}(t) - bT(t), \tag{3}$$

where a and b are positive constants and defined as follows:

$$a = \frac{\kappa}{C_{th}}, b = \frac{1}{R_{th}C_{th}}. \tag{4}$$

We assume that the temperature at time t_0 is T_0, i.e., $T(t_0) = T_0$. Equation (3) is a classic linear differential equation, which can be solved as

$$T(t) = \int_{t_0}^{t} as^{\alpha}(\tau)e^{-b(t-\tau)}d\tau + T_0e^{-b(t-t_0)}. \tag{5}$$

We observe that we can always appropriately scale the speed to control the temperature:

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

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

- 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:

$$T(t) = \frac{as_C^{\alpha}}{b} + \left(T(t_0) - \frac{as_C^{\alpha}}{b}\right)e^{-b(t-t_0)}. \tag{7}$$

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

More accurate thermal models can be derived from this simple one by more closely modeling the power dissipation such as the use of active dissipation devices, or by adding a stochastic component to the input power to model environmental influences other than CPU speed. The authors of Ferreira et al. (2007), for example, extend the thermal model described above to handle thermal effects of dish activity in addition to active dissipation through CPU and case fans. Deterministic extensions to the basic thermal model (e.g., constant dissipation or active dissipation mechanisms that are triggered by temperature thresholds can be easily integrated into the RC model, or

simple extensions of it. Stochastic influences, such as thermal interference in multi-core systems can be either approximated using the RC model, or may need stochastic service models, such as Wang et al. (2004). For sake of simplicity in this paper we focus on the simple RC model, with the understanding that more general models can be handled through appropriate extension of this work.

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

$$T(t) \leq T_H. \tag{8}$$

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

$$0 \leq s(t) \leq s_H. \tag{9}$$

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 (2), 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

$$s_E = \left(\frac{b}{a}T_H\right)^{\frac{1}{\alpha}}. \tag{10}$$

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.

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 the speed s_C in the constant-speed scaling. As a result, *dynamic speed scaling* would be used to improve the overall processor utilization. Any dynamic speed scaling algorithm would have to (a) be supported by existing power control frameworks, such as ACPI and (b) lead to tractable design-time analysis. We therefore use the following *reactive* speed scaling technique:

The processor will run at maximum speed s_H when there is backlogged workload and the temperature is below the threshold T_H. Otherwise, the processor speed is scaled as follows: 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 (10).

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:

$$s(t) = \begin{cases} s_H, & (W(t) > 0) \wedge (T(t) < T_H), \\ s_E, & (W(t) > 0) \wedge (T(t) = T_H), \\ 0, & W(t) = 0. \end{cases} \tag{11}$$

It is easy to show that in any case the temperature never exceeds the threshold T_H. By running the processor at maximum speed, we aim to improve the processor utilization compared with 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 interval P_i between jobs and each job requires C_i processor cycles to complete in the worst case. Figure 1 shows an example of a single periodic task system that uses reactive-speed scaling.

In order to perform design-time schedulability tests, we need to determine the worst-case delay (also called the worst-case response time) for jobs in tasks. This requires us to first identify the worst-case arrival pattern, also called the *critical instance*. Due to the additional temperature constraint, the analysis of the critical instance is different from the traditional case. 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.

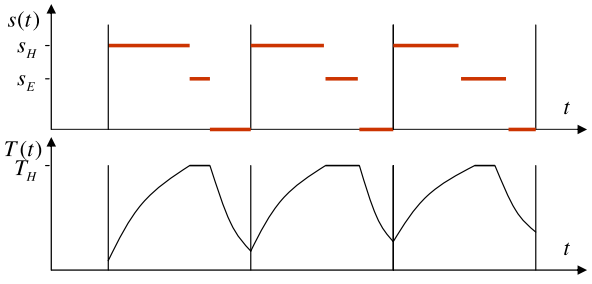


Fig. 1 Illustration of a periodic task with reactive-speed scaling

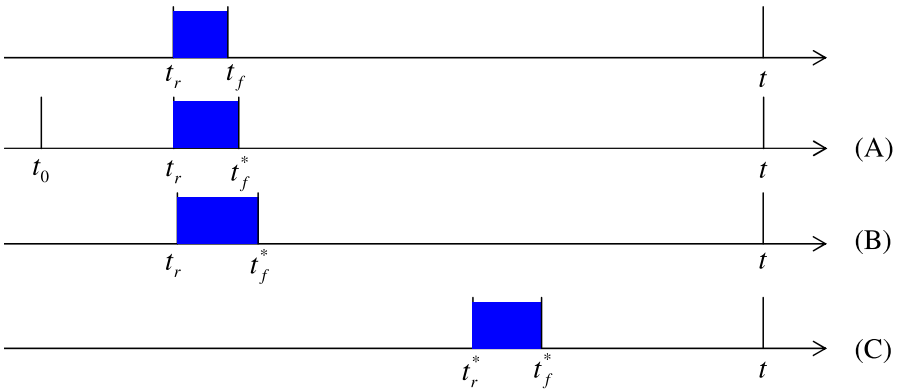


Fig. 2 Temperature effect

3 Important lemmas

The difficulty for delay analysis in a system with reactive-speed scaling lies in the speed of the processor not being constant. Moreover the changes in processing speed are triggered by the thermal behavior, which follows (7). As a result, as we will show, simple busy-period analysis does not work.

The following two lemmas show how the change of temperature, job arrival, and job execution will affect the temperature at a later time or the delay of a later job.

Lemma 1 *In a system under reactive-speed scaling, given a instance t, we consider a job with a release time t_r and a completion time t_f such that $t_r < t$ and $t_f < t$. We assume that the processor is idle during $[t_f, t]$. If we take either of the following actions as shown in Fig. 2:*

- *Action A: Increase the temperature at time t_0 ($t_0 \leq t_r$) such that the job has the same release time t_r but a new completion time t_f^* satisfying $t_f^* < t$;*
- *Action B: Increase the processor cycles for this job such that the job has the same release time t_r but a new completion time t_f^* satisfying $t_f^* < t$;*
- *Action C: Delay the job to have a new release time t_r^* and a new completion time t_r^* satisfying $t_r < t_r^* < t$ and $t_f < t_f^* < t$,*

then we have $T_t \leq T_t^*$, where T_t and T_t^* are the temperatures at time t in the original and the modified scenarios respectively.

Proof For Actions A and B, it is not hard to prove that $t_f \leq t_f^*$ and $T(t_f) \leq T^*(t_f^*)$ when we increase the temperature at time t_r from T_r to T_r^* in Action A or we increase the processor cycles of this job from C to C^* in Action B. Therefore, $T_t \leq T_t^*$. We skip the details here.

For Action C we delay the job to have release time t_r^* and completion time t_f^* such that $t_r < t_r^* < t$ and $t_f < t_f^* < t$. We assume a fixed temperature $T(t_r)$ at t_r. Considering when the temperature will hit T_H, we find four cases:

Case 1: The temperature will hit T_H neither during $[t_r^*, t_f^*]$ for the job-delayed scenario nor during $[t_r, t_f]$ for the original scenario. For the job-delayed scenario, we have[2]

$$T(t_r^*) = T(t_r)e^{-b(t_r^* - t_r)}, \tag{12}$$

$$T(t_f^*) = \frac{as_H^\alpha}{b}(1 - e^{-b(t_f^* - t_r^*)}) + T(t_r^*)e^{-b(t_f^* - t_r^*)}, \tag{13}$$

$$T_t^* = T(t_f^*)e^{-b(t - t_f^*)}. \tag{14}$$

Hence,

$$T_t^* = \frac{as_H^\alpha}{b}(1 - e^{-b(t_f^* - t_r^*)})e^{-b(t - t_f^*)} + T(t_r)e^{-b(t - t_r)}. \tag{15}$$

Similarly, for the original scenario, we have

$$T_t = \frac{as_H^\alpha}{b}(1 - e^{-b(t_f - t_r)})e^{-b(t - t_f)} + T(t_r)e^{-b(t - t_r)}. \tag{16}$$

Now we are ready to compare T_t^* and T_t. In this case, $t_f^* - t_r^* = t_f - t_r = \frac{C}{s_H}$ because the temperature will neither hit T_H during the job execution in both scenarios. Furthermore since $t_r \le t_r^*$, we have $t_f \le t_f^*$. Therefore, with all of these conditions, by (15) and (16), we conclude that $T_t \le T_t^*$ for this case.

Case 2: The temperature will hit T_H at t_H^* during $[t_r^*, t_f^*]$ for the job-delayed scenario and also hit T_H in $[t_r, t_f]$ for the original scenario. For the job-delayed scenario, we have

$$T(t_r^*) = T(t_r)e^{-b(t_r^* - t_r)}, \tag{17}$$

$$T(t_H^*) = \frac{as_H^\alpha}{b}(1 - e^{-b(t_H^* - t_r^*)}) + T(t_r^*)e^{-b(t_H^* - t_r^*)} \tag{18}$$

$$= T_H, \tag{19}$$

$$T(t_f^*) = T_H, \tag{20}$$

$$T_t^* = T(t_f^*)e^{-b(t - t_f^*)}. \tag{21}$$

Hence,

$$t_H^* = -\frac{1}{b}\ln\frac{T_H - \frac{as_H^\alpha}{b}}{T(t_r)e^{bt_r} - \frac{as_H^\alpha}{b}e^{bt_r^*}}. \tag{22}$$

Since $C = s_H(t_H^* - t_r^*) + s_E(t_f^* - t_H^*)$ is fixed, we have

$$t_f^* = \frac{C}{s_E} + \frac{s_H}{s_E}t_r^* - \left(\frac{s_H}{s_E} - 1\right)t_H^*. \tag{23}$$

[2]In the following, the temperature calculation is based on (7).

By (21), (22), and (23), we have

$$T_t^* = T_H e^{-b(t-\frac{C}{s_E}-\frac{s_H}{s_E}t_r^*)}\left(\frac{T_H-\frac{as_H^\alpha}{b}}{T(t_r)e^{bt_r}-\frac{as_H^\alpha}{b}e^{bt_r^*}}\right)^{\frac{s_H}{s_E}-1}. \tag{24}$$

Similarly, for the original scenario, we have

$$T_t = T_H e^{-b(t-\frac{C}{s_E}-\frac{s_H}{s_E}t_r)}\left(\frac{T_H-\frac{as_H^\alpha}{b}}{T(t_r)e^{bt_r}-\frac{as_H^\alpha}{b}e^{bt_r}}\right)^{\frac{s_H}{s_E}-1}. \tag{25}$$

Since $t_r \le t_r^*$, by (24) and (25), we conclude that $T_t \le T_t^*$ for this case.

Case 3: The temperature will not hit T_H during $[t_r^*, t_f^*]$ for the job-delayed scenario but hit T_H in $[t_r, t_f]$ for the original scenario. In this case, we introduce a transition scenario that the job is executed during $[t_r^{**}, t_f^{**}]$, where $t_r \le t_r^{**} \le t_r^*$, $t_f \le t_f^{**} \le t_f^*$, and the temperature hits T_H just at t_f^{**}. The existence of this scenario is obvious. We define T_t^{**} as the temperature at t in this transition scenario.

Since the temperature will hit T_H at the boundary t_f^{**} 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 the temperature analysis in either Case 1 or Case 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
$$T_t \le T_t^{**}. \tag{26}$$
- Second, we compare the transition scenario with the job-delayed scenario. We apply the temperature analysis in Case 1 to both scenarios. Following the result of Case 1, we have
$$T_t^{**} \le T_t^*. \tag{27}$$

Therefore, by (26) and (27), we conclude that $T_t \le T_t^*$ for this case.

Case 4: The temperature will hit T_H during $[t_r^*, t_f^*]$ for the job-delayed scenario but not hit T_H during $[t_r, t_f]$ for the original scenario. Since $t_r \le t_r^*$, by (17), we have $T_r \ge T_r^*$. It is impossible to hit T_H during $[t_r^*, t_f^*]$ and not hit T_H during $[t_r, t_f]$. This case will never happen.

In all possible cases, we conclude that $T_t \le T_t^*$. □

Lemma 2 *In a system under reactive-speed scaling, we consider two jobs J_k's ($k = 1, 2$), each of which has a release time $t_{k,r}$ and the completion time $t_{k,f}$. We assume $t_{1,f} < t_{2,f}$. If we take either of the following actions as shown in Fig. 3, each of which does not delay the release time t_{2r} of Job J_2:*

- *Action A: Increase the temperature at t_0 ($t_0 \le t_{2,r}$) such that Job J_2 has a new completion time $t_{2,f}^*$;*

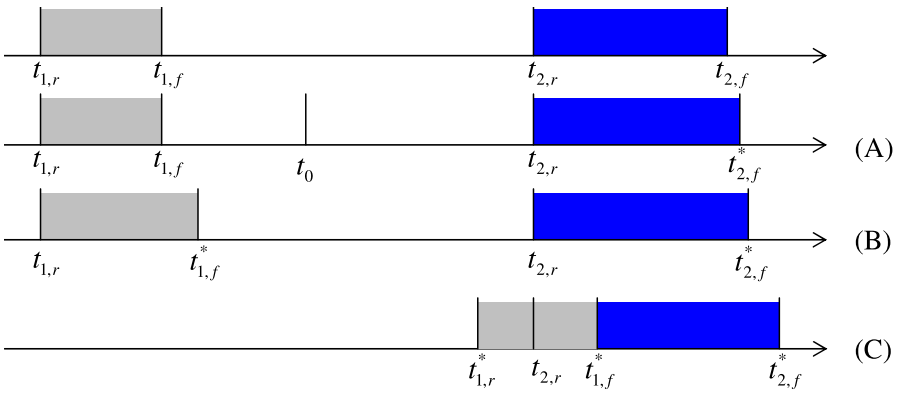
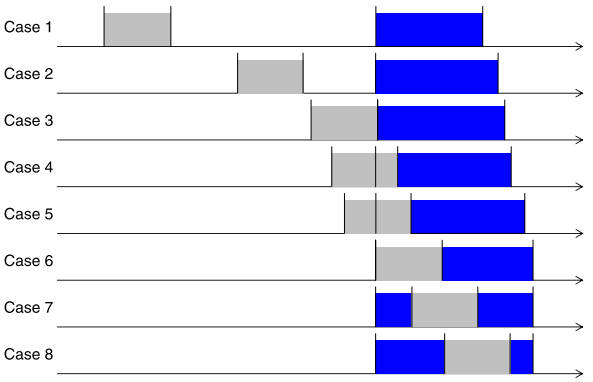


Fig. 3 Delay effect

Fig. 4 8 cases for delaying job



- *Action B: Increase the processor cycles of Job J_1 such that Job J_k $(k = 1, 2)$ has a new completion time $t^*_{k,f}$;*
- *Action C: Delay Job J_1 such that Job J_1 has a new release time $t^*_{1,r}$ and a new completion time $t^*_{1,f}$, and Job J_2 has a new completion time $t^*_{2,f}$ satisfying $t_{1,r} \leq t^*_{1,r}$ and $t^*_{1,f} \leq t^*_{2,f}$,*

then $t_{2,f} \leq t^*_{2,f}$, i.e., the completion time of Job J_2 (or the delay of Job J_2) after the action does not decrease.

Proof For the first two actions (increasing the temperature at t_0 or the processor cycles of Job J_1), we can follow a similar analysis as in the proof of Lemma 1. In the following, we will focus on the last action, i.e., delaying Job J_1.

We consider 8 cases as shown in Fig. 4. We define $t^{(i)}_{k,r}$ and $t^{(i)}_{k,f}$ as the release time and completion time for Job J_k, $k = 1, 2$ in Case i, $i = 1, 2, \ldots, 8$. Define $d^{(i)}_2$ as the delay of Job J_2 in Case i, $i = 1, 2, \ldots, 8$. We find that the original case and the job-delayed case will be Cases i and Cases j in Fig. 4 respectively, where

$1 \le i < j \le 8$. If we can prove that $t_{2,f}^{(i)} \le t_{2,f}^{(i+1)}$ for $i = 1, 2, \ldots, 7$, then $t_{2,f}^{(i)} \le t_{2,f}^{(j)}$ for $1 \le i < j \le 8$.

Case 1 *vs. Case* 2: By Lemma 1, the temperature at time $t_{2,r}$ will increase. It is not hard to prove that the delay of Job J_2 will increase after the job delaying. Therefore we have $t_{2,f}^{(1)} \le t_{2,f}^{(2)}$.

Case 4 *vs. Case* 5: Define C_k as the processor cycles for Jobs J_k, $k = 1, 2$. If the processor does not hit T_H during the execution of Jobs J_1 and J_2 for Case i, $i = 4, 5$, then

$$t_{2,f}^{(i)} - t_{1,r}^{(i)} = \frac{C_1 + C_2}{s_H}. \tag{28}$$

Otherwise

$$t_{2,f}^{(i)} - t_{1,r}^{(i)} = \frac{C_1 + C_2}{s_E} - \frac{1}{b}\left(\frac{s_H}{s_E} - 1\right) \ln \frac{\left(\frac{s_H}{s_E}\right)^{\alpha} - \frac{T_{2,f}^{(i)}}{T_H}}{\left(\frac{s_H}{s_E}\right)^{\alpha} - 1}, \tag{29}$$

where $T_{2,f}^{(i)} = T(t_{2,f}^{(i)})$. We have the following combination scenarios:

- The processor does not hit T_H for both Cases 4 and 5, then by (28), we have

$$t_{2,f}^{(4)} - t_{1,r}^{(4)} = t_{2,f}^{(5)} - t_{1,r}^{(5)}. \tag{30}$$

  Since $t_{1,r}^{(4)} > t_{1,r}^{(5)}$, then $t_{2,f}^{(4)} < t_{2,f}^{(5)}$.
- The processor hits T_H for both Cases 4 and 5, then by (29), we have

$$(t_{2,f}^{(5)} - t_{2,f}^{(4)}) - (t_{1,r}^{(5)} - t_{1,r}^{(4)}) = -\frac{1}{b}\left(\frac{s_H}{s_E} - 1\right) \ln \frac{\left(\frac{s_H}{s_E}\right)^{\alpha} - \frac{T_{2,f}^{(5)}}{T_H}}{\left(\frac{s_H}{s_E}\right)^{\alpha} - \frac{T_{2,f}^{(4)}}{T_H}}. \tag{31}$$

  Define $\epsilon = t_{1,r}^{(5)} - t_{1,r}^{(4)}$ and $T_0 = T_{2,f}^{(4)}$, then $T_{2,f}^{(5)} = T_0 e^{-b\epsilon}$. Hence

$$t_{2,f}^{(5)} - t_{2,f}^{(4)} = \epsilon - \frac{1}{b}\left(\frac{s_H}{s_E} - 1\right) \ln \frac{\left(\frac{s_H}{s_E}\right)^{\alpha} - \frac{T_0}{T_H} e^{-b\epsilon}}{\left(\frac{s_H}{s_E}\right)^{\alpha} - \frac{T_0}{T_H}}, \tag{32}$$

$$\ge \epsilon - \epsilon \frac{\frac{s_H}{s_E} - 1}{\frac{T_H}{T_0}\left(\frac{s_H}{s_E}\right)^{\alpha} - 1}, \tag{33}$$

$$\ge 0. \tag{34}$$

  Therefore, $t_{2,f}^{(4)} \le t_{2,f}^{(5)}$.
- The processor hits T_H for Case 4, but not for Case 5, then we introduce a transition Case 4.5, where the processor hits T_H at the completion time of Job J_2. By the previous analysis, we have $t_{2,f}^{(4)} \le t_{2,f}^{(4.5)}$ and $t_{2,f}^{(4.5)} < t_{2,f}^{(5)}$. Therefore, $t_{2,f}^{(4)} < t_{2,f}^{(5)}$.

- The processor hits T_H for Case 5, but not for Case 4. Since the temperature at $t_{1,r}^{(4)}$ for Case 4 is higher than the one at $t_{1,r}^{(5)}$ for Case 5, It is impossible to hit T_H for Case 5 but not for Case 4. This scenario will never happen.

Case 7 *vs. Case* 8: The temperature at time $t_{2,r}$ the execution processor cycles during the execution of the two aggregated jobs will not change for both cases. Therefore, we have $t_{2,f}^{(1)} = t_{2,f}^{(2)}$.

Case 2 *vs. Case* 3: Since Case 3 is a special case of Case 1 or 2, we follow the same proof as shown in Case 1 vs. Case 2. Therefore we have $t_{2,f}^{(2)} \leq t_{2,f}^{(3)}$.

Case 3 *vs. Case* 4: Since Case 3 is a special case of Case 4 or 5, we follow the same proof as shown in Case 4 vs. Case 5. Therefore we have $t_{2,f}^{(3)} \leq t_{2,f}^{(4)}$.

Case 5 *vs. Case* 6: Since Case 6 is a special case of Case 4 or 5, we follow the same proof as shown in Case 4 vs. Case 5. Therefore we have $t_{2,f}^{(5)} \leq t_{2,f}^{(6)}$.

Case 6 *vs. Case* 7: Since Case 6 is a special case of Case 7 or 8, we follow the same proof as shown in Case 7 vs. Case 8. Therefore we have $t_{2,f}^{(6)} = t_{2,f}^{(7)}$.

Therefore, in any case, we have $t_{2,f}^{(i)} \leq t_{2,f}^{(i+1)}$. Hence $t_{2,f}^{(i)} \leq t_{2,f}^{(j)}$ for $1 \leq i < j \leq 8$, i.e., the completion time of Job J_2 (or the delay of Job J_2) after the action does not decrease.

In the proof above, although we assume there are no other jobs between J_1 and J_2, the lemma still holds for any number of jobs. In the multiple-job case, we split the interval into multiple sub-intervals such that each sub-interval includes only two jobs. For each sub-interval, by Lemma 1, any of three actions will increase the temperature at the end of the sub-interval. Iteratively, by Lemma 2, the delay of the last job will be longer. □

Here we summarize the three actions defined in the above two lemmas as follows:

- Action A: Increase the temperature at some instances;
- Action B: Increase the processor cycles of some jobs;
- Action C: Delay some jobs to a later time.

By the lemmas, with either of the above three actions, we can increase the temperature at a later time and the delay of the later job.

The above two lemmas together with the three actions are important to our delay analysis under reactive-speed scaling, which will be our focus in the next three sections.

4 Critical instance

A *critical instance* of a task Γ_i is an 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 (Liu 2000). 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 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. The higher the temperature at this instance, the longer the 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 instance. In the following, we will base on traditional critical-instance analysis to derive the critical instance in the temperature-constrained case.

In traditional critical-instance analysis in systems without blocking, the critical instance of a task Γ_i occurs when one of its job, say $J_{i,c}$, is released at the same time $r_{i,c}$ with a job of every higher-priority task. In a temperature-constrained system, we must consider not only the worst-case preemption by the high-priority tasks, but also the worst-case temperature at this instance, which is caused by the jobs of *all* tasks *before* this 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 Λ, 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 5 shows an example of a critical instance for a three-task system in both the traditional and a temperature-constrained real-time system.

Proof In the sketch of the proof, we make use of Lemmas 1 and 2. If we delay a task such that its last job during the busy interval Λ is completed a sufficiently-small 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 delayed task, more jobs should be pushed forward before the critical instance of each task. Then by Lemmas 1 and 2, 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. □

5 Identical-periodic tasks

In this paper, we consider a set of *identical-period 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).

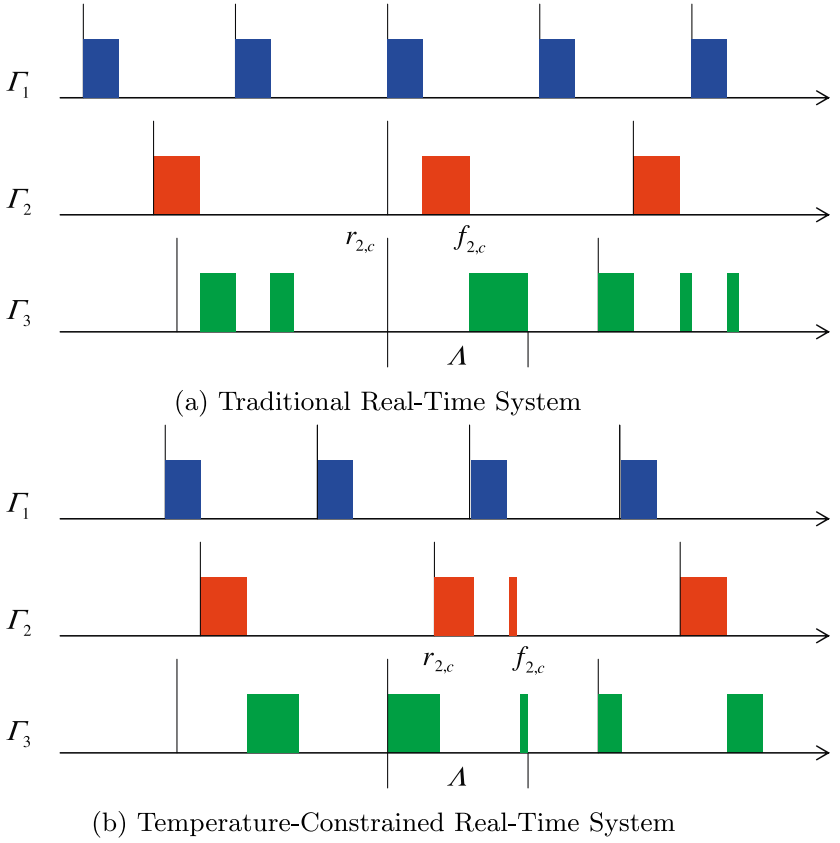


Fig. 5 Illustration of a critical instance

In order to perform the schedulability test, we first consider the critical instance for each task. Figure 6 shows an example of the critical instance for Task Γ_2 in a three-task-identical-period system. 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.

5.1 Temperature computation

First, we compute the temperature at the release time of each job of the new single-task system. We assume the temperature will hit T_H sometime (otherwise, the case at hand is not interesting).

As shown in Fig. 7, we consider a job J_k with a release time $t_{k,r}$ and completion time $t_{k,f}$. The temperature will hit T_H at time $t_{k,H}$. First we obtain the instance

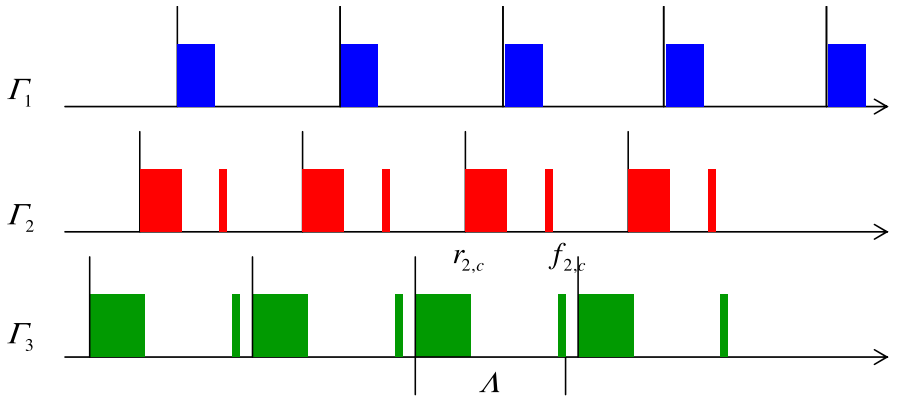


Fig. 6 Illustration of identical periodic tasks

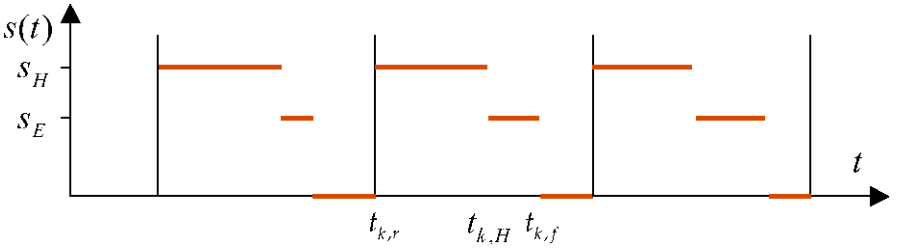


Fig. 7 Illustration of a periodic task under reactive-speed scaling

formulas for $t_{k,r}$, $t_{k,H}$, and $t_{k,f}$:[3]

$$t_{k,r} = kP, \tag{35}$$

$$t_{k,H} = \frac{1}{b} \ln \frac{\frac{as_H^\alpha}{b} - T_{k,r}}{\frac{as_H^\alpha}{b} - T_{k,H}} + t_{k,r}, \tag{36}$$

$$t_{k,f} = \frac{s_H}{s_E}\left(t_{k,r} + \frac{C}{s_H}\right) - \left(\frac{s_H}{s_E} - 1\right)t_{k,H}. \tag{37}$$

Then we obtain the temperature at each instance as follows:

$$T_{k,r} = T_{k-1,f} e^{-b(t_{k,r} - t_{k-1,f})}, \tag{38}$$

$$T_{k,H} = T_H, \tag{39}$$

$$T_{k,f} = T_H. \tag{40}$$

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

Based on the above formulas, we define four time intervals

$$\pi_{k,rH} = t_{k,H} - t_{k,r} = \frac{1}{b} \ln \frac{\frac{as_H^\alpha}{b} - T_{k,r}}{\frac{as_H^\alpha}{b} - T_H}, \tag{41}$$

$$\pi_{k,Hf} = t_{k,f} - t_{k,H} = \frac{s_H}{s_E}\left(\frac{C}{s_H} - \pi_{k,rH}\right), \tag{42}$$

$$\pi_{k,rf} = t_{k,f} - t_{k,r} = \frac{C}{s_E} + \left(1 - \frac{s_H}{s_E}\right)\pi_{k,rH}, \tag{43}$$

$$\pi_{k-1,fr} = t_{k,r} - t_{k-1,f} = \left(P - \frac{C}{s_E}\right) + \frac{1}{b}\left(\frac{s_H}{s_E} - 1\right) \ln \frac{\frac{as_H^\alpha}{b} - T_{k-1,r}}{\frac{as_H^\alpha}{b} - T_H}. \tag{44}$$

By (38), (44), and $T_H = \frac{as_E^\alpha}{b}$, we have

$$\frac{T_{k,r}}{T_H} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-1,r}}{T_H}}{(\frac{s_H}{s_E})^\alpha - 1}\right)^{1-\frac{s_H}{s_E}} \exp\left(-b\left(P - \frac{C}{s_E}\right)\right). \tag{45}$$

Then,

$$\frac{T_{k-1,r}}{T_{k,r}} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-2,r}}{T_H}}{(\frac{s_H}{s_E})^\alpha - \frac{T_{k-1,r}}{T_H}}\right)^{1-\frac{s_H}{s_E}}. \tag{46}$$

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

Next, we want to obtain $\lim_{k\to\infty} T_{k,r}$ by (45), which is the maximum of all $T_{k,r}$'s. As $k \to \infty$, we have the fix point $T^* = \lim_{k\to\infty} T_{k,r}$ by the following equation

$$\frac{T^*}{T_H} = \left(\frac{(\frac{s_H}{s_E})^\alpha - \frac{T^*}{T_H}}{(\frac{s_H}{s_E})^\alpha - 1}\right)^{1-\frac{s_H}{s_E}} \exp\left(-b\left(P - \frac{1}{s_E}\sum_{i=1}^{n} C_i\right)\right). \tag{47}$$

Figure 8 shows the relationship between $\frac{T^*}{T_H}$ and P, where T^* is obtained as the fixed point by (47): As P increases, $\frac{T^*}{T_H}$ decreases.

Now, we go back to the original identical-period task set. We want to obtain the temperature at the critical instance of each original task. Recall that T^* is the maximum temperature at the beginning of a busy period, which can be used to solve this problem as shown in the following lemma:

Lemma 3 *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*:

$$\frac{T_i^*}{T_H} = \min\left\{1, \left(\frac{s_H}{s_E}\right)^\alpha + \left(\frac{T^*}{T_H} - \left(\frac{s_H}{s_E}\right)^\alpha\right)e^{-\frac{b}{s_H}\sum_{j>i} C_j}\right\}. \tag{48}$$

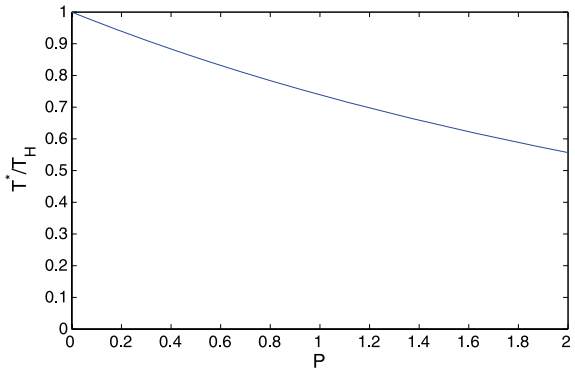


Fig. 8 $\alpha = 3$, $a = b = 1$, $\frac{s_H}{s_E} = 1.5$, $\frac{\frac{C}{s_H}}{P} = 0.5$

Proof At the critical instance $r_{i,c}$, the jobs of lower-priority task will be aligned back-to-back before $r_{i,c}$. Considering when the temperature hit T_H, we have two cases:

- If the temperature does not hit T_H at $r_{i,c}$, i.e., $T_i^* < T_H$, then by (7), we have

$$T_i^* = \frac{as_H^\alpha}{b} + \left(T^* - \frac{as_H^\alpha}{b}\right) e^{-\frac{b}{s_H}\sum_{j>i} C_j}. \tag{49}$$

  By (10), we have

$$\frac{T_i^*}{T_H} = \left(\frac{s_H}{s_E}\right)^\alpha + \left(\frac{T^*}{T_H} - \left(\frac{s_H}{s_E}\right)^\alpha\right) e^{-\frac{b}{s_H}\sum_{j>i} C_j}. \tag{50}$$

  Furthermore, together with $T_i^* < T_H$, we have

$$\left(\frac{s_H}{s_E}\right)^\alpha + \left(\frac{T^*}{T_H} - \left(\frac{s_H}{s_E}\right)^\alpha\right) e^{-\frac{b}{s_H}\sum_{j>i} C_j} < 1. \tag{51}$$

  Hence

$$\frac{T^*}{T_H} < \left(\frac{s_H}{s_E}\right)^\alpha + \left(1 - \left(\frac{s_H}{s_E}\right)^\alpha\right) e^{\frac{b}{s_H}\sum_{j>i} C_j}. \tag{52}$$

- If the temperature hit T_H before $r_{i,c}$, then we have $T_i^* = T_H$.

This proves the lemma. □

5.2 Delay computation

Now we are ready to compute the worst-case delay (or response time) for each task. We consider the response time $d_{i,c}$ for the instance $J_{i,c}$. If $T_i^* = T_H$, we have

$$d_{i,c} = \frac{1}{s_E}\sum_{j \le i} C_j. \tag{53}$$

Otherwise

$$d_{i,c} = \lim_{k\to\infty} \pi_{k,rf} - \frac{1}{s_H} \sum_{j>i} C_j. \tag{54}$$

By (38), we have

$$\lim_{k\to\infty} \pi_{k,rf} = P - \lim_{k\to\infty} (t_{k,r} - t_{k-1,f}) \tag{55}$$

$$= P + \frac{1}{b} \lim_{k\to\infty} \ln \frac{T_{k,r}}{T_H} \tag{56}$$

$$= P + \frac{1}{b} \ln \frac{T^*}{T_H}. \tag{57}$$

The above analysis gives rise to the following theorem, which 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*

$$d_i^{RSS} \le P + \frac{1}{b} \ln \frac{T^*}{T_H} - \frac{1}{s_H} \sum_{j>i} C_j; \tag{58}$$

- *Otherwise*

$$d_i^{RSS} \le \frac{1}{s_E} \sum_{j\le i} C_j. \tag{59}$$

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

$$d_i^{SSS} \le \frac{1}{s_E} \sum_{j\le i} C_j. \tag{60}$$

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*:

$$d^{RSS} \le P + \frac{1}{b} \ln \frac{T^*}{T_H}, \tag{61}$$

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

$$d^{RSS} \leq \begin{cases} P + \frac{1}{b} \ln \frac{T^*}{T_H}, & \frac{T^*}{T_H} < 1 \\ \frac{1}{s_E} \sum_{i \leq n} C_i, & \frac{T^*}{T_H} = 1 \end{cases} \tag{62}$$

By (47), $\frac{T^*}{T_H} < 1$ 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. □

5.3 Utilization computation

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 *If the deadline of task $D_i = \delta P$, where $0 < \delta \leq 1$, then the maximum schedulable utilization U^{RSS} for the tasks under reactive-speed scaling can be expressed as*

$$U^{RSS} = \frac{s_E}{s_H} \xi, \tag{63}$$

where

$$\xi = \min \left\{ 1, \delta + \left(\frac{s_H}{s_E} - 1 \right) \frac{1}{bP} \ln \left(\frac{(\frac{s_H}{s_E})^{\alpha} - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^{\alpha} - 1} \right) \right\}. \tag{64}$$

Proof By Corollary 1, we have two constraints:

- $\frac{1}{s_E} \sum_{i=1}^{n} C_i \leq P$. Hence by the definition of utilization we have

$$U \leq \frac{s_E}{s_H}, \tag{65}$$

- $d^{RSS} \leq D_i$. Hence we have $P + \frac{1}{b} \ln \frac{T^*}{T_H} \leq \delta P$, i.e.,

$$\frac{T^*}{T_H} \leq e^{-b(1-\delta)P}. \tag{66}$$

By the formula of $\frac{T^*}{T_H}$ defined in (47), the above inequality can be written as

$$\frac{1}{s_E} \sum_{i=1}^{n} C_i \leq \delta P + \frac{1}{b} \left(\frac{s_H}{s_E} - 1 \right) \ln \left(\frac{(\frac{s_H}{s_E})^{\alpha} - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^{\alpha} - 1} \right). \tag{67}$$

Therefore,

$$U \leq \frac{s_E}{s_H} \delta + \frac{s_E}{s_H} \left(\frac{s_H}{s_E} - 1 \right) \frac{1}{bP} \ln \left(\frac{(\frac{s_H}{s_E})^{\alpha} - e^{-b(1-\delta)P}}{(\frac{s_H}{s_E})^{\alpha} - 1} \right). \tag{68}$$

By (65) and (68), if we define ξ as (64), the corollary is proved. □

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

$$\frac{1}{s_E}\sum_{i=1}^{n} C_i \leq \delta P. \tag{69}$$

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

$$U^{SSS} = \frac{s_E}{s_H}\delta. \tag{70}$$

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.

6 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 Sect. 5. 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 9 shows all the performance 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

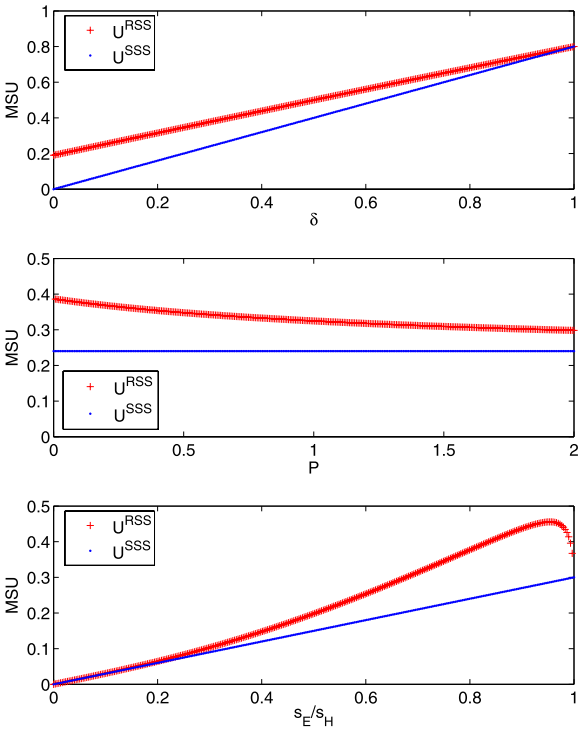


Fig. 9 Evaluation of maximum schedulable utilization

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.

7 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. What is 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. In Wang and Bettati (2006), for example, we approximate task arrivals using so called arrival-bounding

function and apply techniques borrowed from network calculus (Boudec and Thiran 2001) to upper-bound the thermal effect of workload before the critical instant.

Proactive speed scaling schemes come in a variety of ways. For example, the maximum speed can be defined per task, with low-priority tasks being assigned low maximum speeds. At this point we need to better understand the criteria needed to allocate speeds to tasks.

Acknowledgements This work is being funded by the University of Michigan under a Reckham Faculty Research Grant and the National Science Foundation under Grant CNS-0509483.

References

Association SI (2005) International technology roadmap for semiconductors. http://public.itrs.net

Aydin H, Melhem R, Mossé D, Alvarez PM (2001) Dynamic and aggressive scheduling techniques for power-aware real-time systems. In: Proceedings of IEEE real-time systems symposium, London, UK, December 2001

Bansal N, Pruhs K (2005) Speed scaling to manage temperature. In: Proceedings of symposium on theoretical aspects of computer science, Stuttgart, Germany, February 2005

Bansal N, Kimbrel T, Pruhs K (2004) Dynamic speed scaling to manage energy and temperature. In: Proceedings of IEEE symposium on foundations of computer science, Rome, Italy, October 2004

Boudec JYL, Thiran P (2001) Network calculus: A theory of deterministic queuing systems for the Internet. Springer, New York

Brooks D, Martonosi M (2001) Dynamic thermal management for high-performance microprocessors. In: Proceedings of international symposium on high-performance computer architecture, Nuevo Leone, Mexico, January 2001

Cohen A, Finkelstein L, Mendelson A, Ronen R, Rudoy D (2003) On estimating optimal performance of cpu dynamic thermal management. Comput Architecture Lett 2(1):6

Dhodapkar A, Lim C, Cai G, Daasch W (2000) Tempest: A thermal enabled multi-model power/performance estimator. In: Proceedings of workshop on power-aware computer systems, Cambridge, MA, November 2000

Ferreira A, Mosse D, Oh J (2007) Thermal faults modeling using a rc model with an application to web farms. In: Proceedings of the 19th euromicro conference on real-time systems (ECRTS 07), Pisa, Italy

Gochman S, Mendelson A, Naveh A, Rotem E (2006) Introduction to intel core duo processor architecture. Intel Technol J 10(2):89–97

Liu J (2000) Real-time systems. Prentice Hall, New Jersey

Liu Y, Mok AK (2003) An integrated approach for applying dynamic voltage scaling to hard real-time systems. In: Proceedings of IEEE real-time and embedded technology and applications symposium, Washington, DC, May 2003

Pillai P, Shin KG (2001) Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of ACM symposium on operating systems principles, Banff, Alberta, Canada, October 2001

Qadi A, Goddard S, Farritor S (2003) A dynamic voltage scaling algorithm for sporadic tasks. In: Proceedings of IEEE real-time systems symposium, Cancun, Mexico, December 2003

Quan G, Niu L, Hu XS, Mochocki B (2003) Fixed priority scheduling for reducing overall energy on variable voltage processors. In: Proceedings of IEEE real-time systems symposium, Cancun, Mexico, December 2003

Rotem E, Naveh A, Moffie M, Mendelson A (2004a) Analysis of thermal monitor features of the intel pentium m processor. In: Workshop on temperature-aware computer systems, Munich, Germany, June 2004

Rotem E, Naveh A, Moffie M, Mendelson A (2004b) Analysis of thermal monitor features of the intel pentium m processor. In: Workshop on temperature-aware computer systems, Munich, Germany, June 2004

Saewong S, Rajkumar R (2003) Practical voltage-scaling for fixed-priority rt-systems. In: Proceedings of IEEE real-time and embedded technology and applications symposium, Washington, DC, May 2003

Sanchez H, Kuttanna B, Olson T, Alexander M, Gerosa G, Philip R, Alvarez J (1997) Thermal management system for high performance powerpc microprocessors. In: Proceedings of IEEE international computer conference, San Jose, CA, February 1997

Sergent JE, Krum A (1998) Thermal management handbook. McGraw-Hill, New York

Skadron K, Stan M, Huang W, Velusamy S, Sankaranarayanan K, Tarjan D (2003a) Temperature-aware microarchitecture: Extended discussion and results. Technical Report CS-2003-08, Dept. of Computer Science, University of Virginia, April 2003

Skadron K, Stan MR, Huang W, Velusamy S, Tarjan D (2003b) Temperature-aware computer systems: Opportunities and challenges. IEEE Micro 23(6):52–61

Wang S, Bettati R (2006) Delay analysis in temperature-constrained hard real-time systems with general task arrivals. In: IEEE real-time systems symposium, Rio de Janeiro, Brazil, December 2006

Wang S, Nathuji R, Bettati R, Zhao W (2004) Providing statistical delay guarantees in wireless networks. In: IEEE international conference on distributed computing systems

Yeh LT, Chu RC (2002) Thermal management of microelectronic equipment: Heat transfer theory, analysis methods, and design practices. ASME Press, New York

Zhang F, Chanson ST (2005) Power-aware processor scheduling under average delay constraints. In: Proceedings of IEEE real-time and embedded technology and applications symposium, San Francisco, CA, March 2005

Shengquan Wang received his B.S. degree in Mathematics from Anhui Normal University, China, in 1995, and his M.S. degree in Applied Mathematics from the Shanghai Jiao Tong University, China. He also received M.S. degree in Mathematics in 2000 and Ph.D. in Computer Science from Texas A&M University. He is currently Assistant Professor in the Department of Computer and Information Science at the University of Michigan-Dearborn. His research interests are in real-time computing and communication, security in networks and distributed systems, and wireless networks.

Riccardo Bettati received his Diploma in Informatics from the Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, in 1988 and his Ph.D. from the University of Illinois at Urbana-Champaign in 1994. From 1993 to 1995, he held a postdoctoral position at the International Computer Science Institute in Berkeley and at the University of California at Berkeley. He is currently Professor in the Department of Computer Science at Texas A&M University. His research interests are in traffic analysis and privacy, real-time distributed systems, real-time communication, and network support for resilient distributed applications.