Real-Time Syst (2010) 46: 160–188
DOI 10.1007/s11241-010-9104-7

Schedulability analysis in hard real-time systems under thermal constraints

Shengquan Wang · Youngwoo Ahn · Riccardo Bettati

Published online: 3 August 2010

Abstract In this paper, we study thermal-constrained hard real-time systems, where real-time guarantees must be met without exceeding safe temperature levels within the processor. Dynamic speed scaling is one of the major techniques to manage power so as to maintain safe temperature levels. As example, we adopt a reactive speed control technique in our work. We design an extended busy-period analysis methodology to perform schedulability analysis for general task arrivals under reactive speed control with First-In-First-Out (FIFO), Static-Priority (SP), and Earliest-Deadline-First (EDF) scheduling. As a special case, we obtain a closed-form formula for the worst-case response time of jobs under the leaky-bucket task arrival model. Our data show how reactive speed control can decrease the worst-case response time of tasks in comparison with any constant-speed scheme.

Keywords Thermal · Dynamic speed scaling · Real-time · Scheduling · Schedulability analysis

S. Wang (✉)
Department of Computer and Information Science, University of Michigan-Dearborn, Dearborn, MI 48128, USA
e-mail: shqwang@umd.umich.edu

Y. Ahn · R. Bettati
Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843, USA

Y. Ahn
e-mail: ayw@ece.tamu.edu

R. Bettati
e-mail: bettati@cs.tamu.edu

1 Introduction

With the rapidly increasing power density in processors the problem of thermal management in systems is becoming acute. Methods to manage heat to control its dissipation have been gaining much attention by researchers and practitioners. Techniques are being investigated for thermal control both at design time through appropriate packaging and active heat dissipation mechanisms, and at run time through various forms of dynamic thermal management (DTM) (e.g., Brooks and Martonosi 2001).

Thermal management through packaging (that improves airflow, for example) and active heat dissipation is very expensive. Tiwari et al. (1998) for example show how the incremental packaging cost per additional Watt becomes very high for processors above 35–40 W power dissipation. A recent technology roadmap of the Semiconductor Industry Association (ITRS05 2005) predicts that packaging will become increasingly challenging in the near future, due to the high levels of peak power involved and the extremely high power density in emerging systems-in-package. In addition, for many high-performance embedded systems the packaging requirements and operating environments render expensive and bulky packaging solutions inappropriate.

A number of DTM approaches to control the temperature at run time have been proposed, ranging from clock throttling to dynamic voltage scaling (DVS) to in-chip load balancing:

– Many processors use *Clock Throttling* (e.g., Rotem et al. 2004) or *Clock Gating* (Skadron et al. 2003) to stall the clock and so allow the processor to cool during thermal overload.
– *DVS* (Brooks and Martonosi 2001) is used in a variety of modern processor technologies and allows to switch between different frequency and voltage operating points at run time in response to the current thermal situation. In the Enhanced Intel SpeedStep mechanism in the Pentium M processor, for example, a low-power operating point is reached in response to a thermal trigger by first reducing the frequency (within a few microseconds) and then reducing the voltage (at a rate of one mV per microsecond; Rotem et al. 2004).
– A number of *architecture-level* mechanisms for thermal control have been proposed that turn off components inside the processor in response to thermal overload. Skadron et al. (2003) for example argue that the microarchitecture should distribute the workload in response to the thermal situation by taking advantage of instruction-level parallelism. The performance penalty caused by this "local gating" would not be excessive. On a coarser level, the Pentium Core Duo Architecture allows the OS or the BIOS to disable one of the cores by putting it into sleep mode (Gochman et al. 2006).

As high-performance embedded systems become increasingly thermally-constrained, the question of how the thermal behavior of the system and the thermal control mechanisms affect *real-time guarantees* must be addressed. In this paper we describe schedulability analysis techniques in thermally-constrained hard real-time systems, where *deadline* constraints for tasks have to be balanced against *temperature* constraints of the system.

Both in research and practice, dynamic speed control[1] is one of the major techniques to control the temperature. Dynamic speed scaling allows for a trade-off between these two performance metrics: To meet the deadline constraint, we run the processor at a higher speed; To maintain the safe temperature levels, we run the process at a lower speed.

There is some recent work on dynamic speed scaling techniques to control temperature. For example, in Srinivasan and Adve (2003), a predictive DTM algorithm was designed to improve the performance of multimedia applications. In Cohen et al. (2003), Rao et al. (2006), optimal speed profiles were derived to achieve high resource utilization. This work focuses on improving resource utilization rather than on providing real-time guarantees, which is the focus of this paper. Zhang and Chatha (2007) address the knapsack problem for a given execution sequence of jobs by assigning discrete frequency/voltage states. They prove that the problem is NP-hard and proceed to formulate a pseudo-polynomial optimal speed assignment algorithm and a polynomial time approximation algorithm.

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 release time. In contrast, in our work we assume a predictive task model (e.g., periodic tasks) and so allow for design-time schedulability analysis. In Ferreira et al. (2006), a thermal model is presented that is capable of modeling cooling faults such as CPU fan or case fan failures and load-balancing algorithms were design based on this model. This work is complementary to ours.

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 they are simple to integrate with current processor capabilities through existing power control frameworks such as the Advanced Configuration and Power Interface (ACPI) power control framework (ACPI 2010; Sanchez et al. 1997; Rotem et al. 2004).

In order to provide timing guaranteed services for real-time tasks, schedulability analysis is needed. Unfortunately, traditional schedulability analysis will not work under thermal-aware design. In traditional schedulability analysis, we could target on jobs only in a busy period (before and after that the processor is idle) because the behavior of jobs in different busy periods do not affect each other (Liu 2000). However, in thermal-aware design, it becomes difficult to separate the execution of jobs in a busy period from the interference by the execution of jobs in an earlier busy period because the speed of the processor is triggered by the thermal behavior and varies over time under the DTM scheme. To tackle this issue, we introduce an

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

extended busy-period analysis methodology. In Wang and Bettati (2008), we perform schedulability analysis under reactive speed scaling for identical-period tasks. In this paper, we extend it to general task arrivals with First-In-First-Out (FIFO), Static-Priority (SP), and Earliest-Deadline-First (EDF) scheduling.

The rest of the paper is organized as follows. In Sect. 2, we introduce the thermal model, the speed scaling schemes, and the task model and the scheduling algorithms. After discussing the thermal interference on schedulability analysis in Sect. 3, we design the extended busy-period analysis methodology to perform schedulability analysis for FIFO, SP, and EDF scheduling algorithms in Sects. 4, 5, and 6 respectively. We measure the response time performance for tasks under each scheduling algorithm in Sect. 7. Finally, we conclude our work with final remarks and give an outlook on future work in Sect. 8.

2 Models

2.1 Power model

The power consumption P is contributed by the following two main sources:

- *The dynamic power consumption* P_D mainly resulting from the charging and discharging of gates on the circuits. The dynamic power consumption could be modelled as a convex function of the processor speed such as the dynamic power consumption in CMOS processors (Rabaey and Chandrakasan 2002): $P_D = C_{ef} V_{dd}^2 s$, where $s = \kappa_v \frac{(V_{dd}-V_t)^2}{V_{dd}}$ is defined as the *processor speed*.[2] We can further simplify the formula of the dynamic power consumption as $P_D = \beta_0 s^\alpha$, where β_0 and $\alpha \leq 3$ are constants. Usually, it is assumed that $\alpha = 3$ (Bansal et al. 2004; Bansal and Pruhs 2005).
- *The leakage power consumption* P_L mainly resulting from leakage current. The leakage power consumption function of the system could be modelled as a non-negative constant when leakage power consumption is irrelevant to the temperature (Xu et al. 2005; Jian-Jia et al. 2007). When the leakage power consumption is related to the temperature, it could be approximately modelled by a linear function of the temperature (Chantem et al. 2008). Hence, the leakage power consumption is as follows: $P_L = \beta_1 T + \beta_2$, where T is the temperature and β_1 and β_2 are constants.

In this paper, we use the following formula as the overall power consumption

$$P = P_D + P_L = \beta_0 s^\alpha + \beta_1 T + \beta_2. \tag{1}$$

The power consumption $P = P(t)$, the speed $s = s(t)$ and the temperature $T = T(t)$ are all functions of time t.

[2] C_{ef}, V_t, V_{dd}, and κ_v denote the effective switch capacitance, the threshold voltage, the supply voltage, and a hardware-design-specific constant, respectively. $V_{dd} \geq V_t \geq 0$; $\kappa_v, C_{ef} > 0$.

2.2 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. 2003), 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 a very simple chip-wide thermal model 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. (2003) for example agree that it is somewhat 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, 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). 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 augmenting the input power by a stochastic component, etc.

We assume that the ambient has a fixed temperature T_a. We adopt Fourier's Law as shown in the following formula (Bansal et al. 2004; Bansal and Pruhs 2005; Cohen et al. 2006):

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

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:

$$a = \frac{\beta_0}{C_{th}}, \qquad b = \frac{1}{R_{th}C_{th}} - \frac{\beta_1}{C_{th}}, \tag{4}$$

and $T(t)$ is also scaled to be $T(t) - R_{th}\beta_2 - T_a$. Equation (3) is a classic linear differential equation. If we assume that the temperature at time t_0 is T_0, i.e., $T(t_0) = T_0$, (3) 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 can 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 many speed scaling schemes.

2.3 Speed scaling

The effect of many dynamic thermal management schemes (most prominently DVS and clock throttling) can be described by the speed/temperature relation depicted in (6) and (7). The goal of dynamic thermal management is to maintain the processor temperature within a safe operating range, and not exceed what we call the *highest-temperature threshold* T_H, which in turn should be at a safe margin from the maximum junction temperature of the chip. Temperature control must ensure that

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

On the other hand, we can freely set the processor speed, up to some maximum speed s_H, i.e.,

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

In the absence of dynamic speed scaling we have to set a constant value of the processing speed so that the temperature will never exceed T_H. Assuming that the initial temperature is less than T_H, we can define *equilibrium speed* s_E as

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

For any constant processor speed not exceeding s_E, the processor does not exceed temperature T_H, which can be easily proved by (7) and (10). Note that the equilibrium speed s_E is the maximum constant speed that we can set to maintain the safe temperature level.

A dynamic speed scaling scheme would take advantage of the power dissipation during idle times. It 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_E. As a result, dynamic speed scaling would be used to improve the overall processor utilization.

In defining the dynamic speed scaling algorithm we must keep in mind the following important criteria:

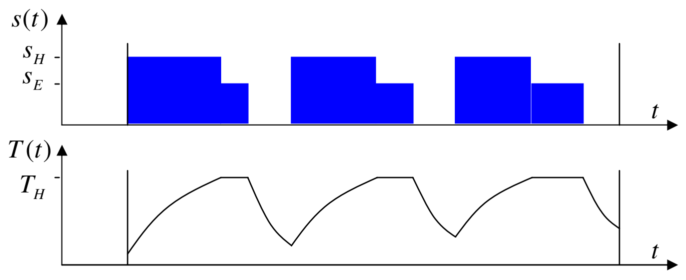


Fig. 1 Illustration of reactive speed scaling

- It must be supported by existing power control frameworks such as ACPI (ACPI 2010; Sanchez et al. 1997; Rotem et al. 2004). The ACPI specification was developed to provide a configuring the hardware, systems, and software necessary for power and thermal management within the PC. For the thermal management, the ACPI specification describes the threshold temperature spectrum and includes guidelines on how to use thermal monitor and CPU frequency(i.e., speed) scaling to make thermal management decisions.
- It must lead to tractable design–time delay analysis. We can provide a theoretical performance evaluation of the proposed scheme.

We therefore use the following very simple *reactive* speed scaling algorithm:

> The processor will run at maximum speed s_H when there is backlogged workload and the temperature is below the threshold T_H. Whenever the temperature hits T_H, the processor will run at the equilibrium speed s_E, which is defined in (10). Whenever the backlogged workload is empty, the processor idles (runs at the zero speed).

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:

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

Figure 1 shows an example of how temperature changes under reactive speed scaling.

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. We will also see in the rest of this paper that reactive speed scaling will lead to tractable design—time delay analysis.

2.4 Task model and scheduling algorithms

The workload consists of a set of tasks $\{\Gamma_i : i = 1, 2, \dots, n\}$. Each task Γ_i is composed of a sequence of jobs. For a job, the time elapsed from the *release* time t_r to the *completion* time t_f is called the *response time* of the job, and the worst-case response time of all jobs in Task Γ_i is denoted by d_i. Jobs within a task are executed in a first-in first-out order.

We characterize the workload of Task Γ_i by the *workload function* $f_i(t)$, the accumulated requested processor cycles of all the jobs from Γ_i released during $[0, t]$. Similarly, to characterize the actual executed processor cycles received by Γ_i, we define $g_i(t)$, the *service function* for Γ_i, as the total executed processor cycles rendered to jobs of Γ_i during $[0, t]$.

In reality, the time-dependent workload function is hard to obtain. Furthermore, even if it were available, it would be intractable to perform schedulability analysis. A well-known alternative to the workload function is the time-independent workload constraint function $F_i(I)$, which is defined as follows.

Definition 1 (Workload constraint function) $F_i(I)$ is a workload constraint function for the workload function $f_i(t)$, if for any $0 \le I \le t$,

$$f_i(t) - f_i(t - I) \le F_i(I). \tag{12}$$

For $I < 0$, we define $F_i(I) = 0$.

For example, if a task Γ_i is constrained by a leaky bucket with a bucket size σ_i and an average rate ρ_i, then its workload constraint function can be written as

$$F_i(I) = \sigma_i + \rho_i I. \tag{13}$$

Once tasks arrive in our system, a scheduling algorithm will be used to schedule the service order of jobs from different tasks. Both the workload and the scheduling algorithm will determine the response time experienced by jobs. In this paper, we consider three scheduling algorithms: *First-in First-out (FIFO)*, *Static Priority (SP)*, and *Earliest-Deadline-First (EDF)* scheduling.

3 Thermal interference on schedulability analysis

The interference of job execution in thermal-aware design expands to a very wide temporal domain beyond single busy period in traditional schedulability analysis. In traditional schedulability analysis, we could target on jobs only in a single busy period because the behavior of jobs in different busy periods do not affect each other (Liu 2000). However, in thermal-aware design, it becomes difficult to separate the execution of jobs in a busy period from the interference by the execution of jobs in an earlier busy period because the speed of the processor can be triggered by the thermal behavior and varies over time under the thermal management.

The following two lemmas (Wang and Bettati 2008) show how the change of temperature, job arrival, or job execution affect the temperature at a later time or the response time of a later job.

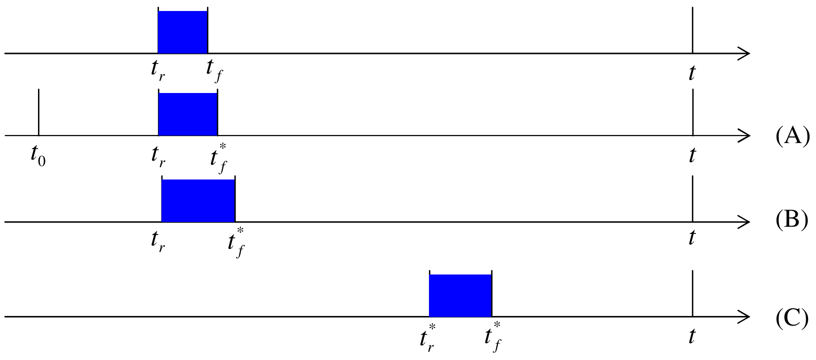


Fig. 2 Temperature effect

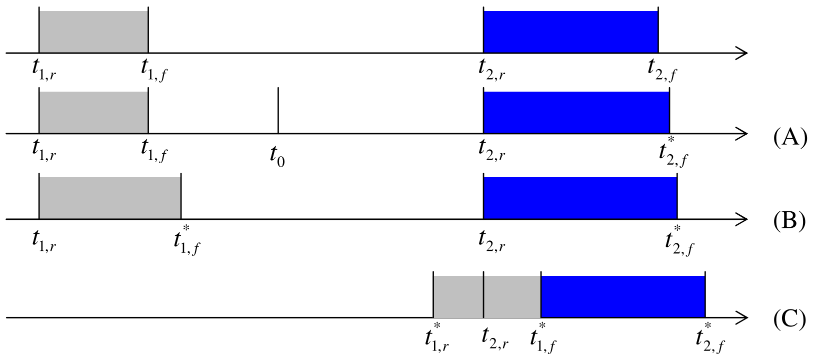


Fig. 3 Response time effect

Lemma 1 *In a system with reactive speed scaling, given a time 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: Increasing the temperature at time t_0 ($t_0 \le t_r$) such that the job has the same release time t_r but a new completion time t_f^* satisfying $t_f^* < t$;*
- *Action B: Increasing 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: Shifting the job such that the job has 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 $T_t \le T_t^*$, where T_t and T_t^* are the temperatures at time t in the original and the modified scenarios respectively.

Lemma 2 *In a system with 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:*

– *Action A*: *Increasing the temperature at t_0 ($t_0 \leq t_{2,r}$) such that Job J_2 has the same release time $t_{2,r}$ but a new completion time $t^*_{2,f}$;*
– *Action B*: *Increasing the processor cycles of Job J_1 such that Job J_k ($k = 1, 2$) has the same release time $t_{k,r}$ but a new completion time $t^*_{k,f}$;*
– *Action C*: *Shifting 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 the same release time $t_{2,r}$ and a new completion time $t^*_{2,f}$ satisfying $t_{1,r} \leq t^*_{1,r}$ and $t^*_{1,f} \leq t^*_{2,f}$,*

then $t_{2,f} \leq t^*_{2,f}$. If we define d_2 and d^*_2 as the response time of Job J_2 in the original and the modified scenarios respectively, then $d_2 \leq d^*_2$.

The proofs of Lemmas 1 and 2 can be found in Wang and Bettati (2008).

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

– Action A: Increasing the temperature at some time instances;
– Action B: Increasing the processor cycles of some jobs;
– Action C: Shifting the execution of 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 response time of some later jobs.

The above lemmas show the thermal interference on schedulability analysis beyond single busy period. Therefore, a new schedulability analysis approach has to be designed. In the following section, we will present our new schedulability analysis approach: *Extended busy-period analysis*.

4 Schedulability analysis for FIFO scheduling

Recall that the speed of the processor is triggered by the thermal behavior and varies over time under reactive speed scaling. Simple busy-period analysis will not work in this environment. In simple busy-period analysis, the jobs arriving before the busy period will not affect the response time of jobs arriving during the busy period. However, under reactive speed scaling, the execution of a job arriving earlier will heat up the processor and so affect the response time of a job arriving later as shown in Lemma 2. Therefore, in the busy-period analysis under reactive speed scaling, we have to take this effect into consideration. We start our schedulability analysis in the system with FIFO scheduling.

4.1 Single busy-period analysis

Under FIFO scheduling, all tasks experience the same worst-case response time as the aggregated task does. Therefore, we consider the aggregated task, whose workload constraint function can be written as

$$F(I) = \sum_{i=1}^{n} F_i(I). \tag{14}$$

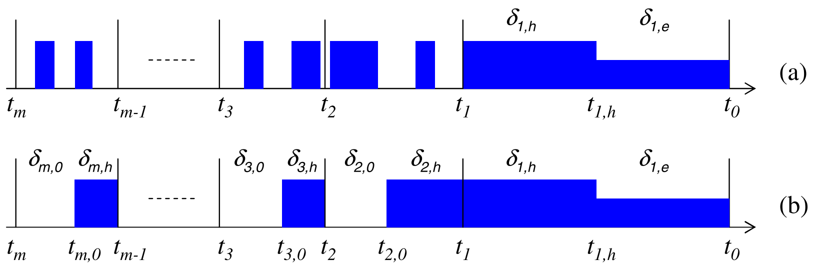


Fig. 4 Job executions

We consider a busy period $[t_1, t_0]$ with length δ_1 during which a job will experience the longest response time and immediately before which the processor is idle. The processor runs at high speed s_H in Interval $[t_1, t_{1,h}]$ with length $\delta_{1,h}$ and at equilibrium speed s_E in Interval $[t_{1,h}, t_0]$ with length $\delta_{1,e}$ as shown in the right side of Fig. 4(a).

We define d as the worst-case response time experienced by a job in the busy period $[t_1, t_0]$. Then, by the definition of worst-case response time (Wu et al. 2005), we have

$$d = \sup_{t \geq t_1}\{\inf\{\tau : f(t) \leq g(t+\tau)\}\}, \tag{15}$$

where $f(t)$ and $g(t)$ are the workload function and the service function of the aggregated task respectively, as defined in Sect. 2. In other words, if by time $t+\tau$, the service received by the task is no less than its workload function $f(t)$, then all jobs of the task arriving before time t should have been served, with a response time no more than τ.

Since the processor is idle at time t_1, we have $f(t_1) = g(t_1)$. Therefore, $f(t) \leq g(t+\tau)$ in (15) can be written as

$$f(t) - f(t_1) \leq g(t+\tau) - g(t_1). \tag{16}$$

First, we study the right side of (16). Recall that the processor runs at high speed s_H in Interval $[t_1, t_{1,h}]$ with length $\delta_{1,h}$ and at equilibrium speed s_E in Interval $[t_{1,h}, t_0]$ with length $\delta_{1,e}$. If we define $I = t - t_1$, then we have

$$g(t+\tau) - g(t_1) = G(I+\tau), \tag{17}$$

where $G(I)$, which we call *service constraint function* of $g(t)$, is defined as

$$G(I) = \min\{(s_H - s_E)\delta_{1,h} + s_E I, s_H I\}. \tag{18}$$

Next, we study the left side of (16). With Action B, a job will experience a longer response time with more workload released and completed before its completion.

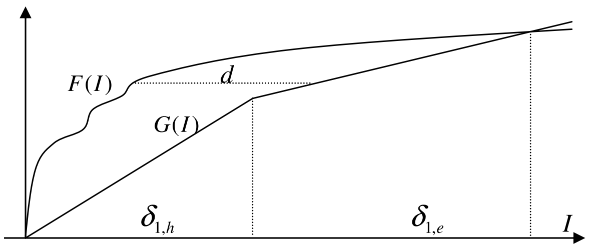


Fig. 5 Response time constraint

Therefore, if we set

$$f(t) - f(t_1) = F(t - t_1) = F(I), \tag{19}$$

then together with (17) the worst-case response time in (16) can be written as

$$d = \sup_{I \geq 0}\{\inf\{\tau : F(I) \leq G(I + \tau)\}\}. \tag{20}$$

The above formula is illustrated in Fig. 5.

4.2 Extended busy-period analysis

The above single busy-period is not enough for thermal-constrained schedulability analysis. As we can see, the undetermined service constraint function $G(I)$ is the key in the worst-case response time formula (20). As defined in (18), $G(I)$ is a function of $\delta_{1,h}$, which obviously depends on the temperature at time t_1. As we mentioned in previous section, the temperature at time t_1 will also be affected by earlier job executions. In the following, we will present our extended busy-period schedulability analysis to address this issue.

The exact temperature at t_1 is hard to obtain. Instead, we aim to obtain a tight upper-bound of the temperature at t_1, which will result in an upper-bound of the worst-case response time according to Lemma 2.

To achieve this, we introduce extra intervals $[t_{k+1}, t_k]$'s ($k = 1, \ldots, m - 1$), as shown in Fig. 4(a). By Lemma 1, we can use the three actions mentioned earlier to upper-bound the temperature at t_1. With Action A, we upper-bound the temperature at t_m by T_H. With Action C, for each Interval δ_k ($k = 2, \ldots, m$), we shift all parts of job execution to the end of this interval, such that the beginning part is idle with length $\delta_{k,0}$ and the ending part is busy with length $\delta_{k,h}$, as shown in Fig. 4(b). We assume that the temperature will not hit T_H during $[t_m, t_1]$,[3] then the processor will run at high speed s_H during each interval $[t_{k+1,0}, t_k]$.

In the following, we investigate the service and the thermal interference in the extended busy period.

[3]If there is an interval $[t_{k_0+1}, t_{k_0}]$ during which the temperature hits T_H, then the temperature at t_{k_0} is T_H. In this case, we can set $m = k_0$ and remove all intervals on the left.

4.2.1 Service in extended busy period

We consider the service received in each interval $[t_k, t_0]$, $k = 1, \dots, m$. As shown in Fig. 4(b), the executed processor cycles in $[t_k, t_0]$ can be written as

$$g(t_0) - g(t_k) = s_H \sum_{j=1}^{k} \delta_{j,h} + s_E \delta_{1,e}. \tag{21}$$

For $k = 1$, we have $g(t_0) - g(t_1) = G(t_0 - t_1) = f(t_0) - f(t_1)$. Following the schedulability analysis in the above response time constraint, we consider the worst-case workload $f(t_0) - f(t_1) = F(t_0 - t_1)$. Therefore, by (21) we have

$$s_H \delta_{1,h} + s_E \delta_{1,e} = F(\delta_{1,h} + \delta_{1,e}). \tag{22}$$

For $k = 2, \dots, m$, by the definition of the worst-case response time in (15), we have $f(t_k - d) \le g(t_k)$. Then together with $g(t_0) \le f(t_0)$ the number of processor cycles in Interval $[t_k, t_0]$ is bounded as

$$g(t_0) - g(t_k) \le f(t_0) - f(t_k - d). \tag{23}$$

By Lemma 2, the response time will become longer when $g(t_0) - g(t_k) = f(t_0) - f(t_k - d) = F(t_0 - t_k + d)$ by either shifting the job execution or increasing the processor cycles of jobs. Therefore, by (21) we have

$$s_H \sum_{j=1}^{k} \delta_{j,h} + s_E \delta_{1,e} = F\left(\sum_{j=1}^{k} \delta_{j,h} + \delta_{1,e} + d\right). \tag{24}$$

Note that the service received by jobs depends on the processing speed, which changes with the thermal behavior. Next we want to see how the temperature changes in each interval.

4.2.2 Thermal interference in extended busy period

First, we consider each interval $[t_{k+1}, t_k]$, $k = 1, \dots, m-1$, which is composed of an idle period with length $\delta_{k+1,0}$ and a busy period with length $\delta_{k+1,h}$. Define T_k as the temperature at t_k, then following the temperature formula (7), we have

$$T_k = \frac{a s_H^\alpha}{b} + \left(T_{k+1} e^{-b\delta_{k+1,0}} - \frac{a s_H^\alpha}{b}\right) e^{-b\delta_{k+1,h}}. \tag{25}$$

Together with the assumption that $T_k \le T_H$ and $T_m = T_H$, we have

$$\frac{T_k}{T_H} = \left(\frac{s_H}{s_E}\right)^\alpha \sum_{r=k+1}^{m} e^{-b\sum_{l=k+1}^{r-1} \delta_l} \left(1 - e^{-b\delta_{r,h}}\right) + e^{-b\sum_{l=k+1}^{m} \delta_l} \le 1. \tag{26}$$

Next, considering Interval $[t_1, t_{1,h}]$, we have

$$\frac{T_1}{T_H} = \left(\frac{s_H}{s_E}\right)^\alpha - \left(\left(\frac{s_H}{s_E}\right)^\alpha - 1\right) e^{b\delta_{1,h}}. \tag{27}$$

In summary, the above schedulability analysis results in several important concrete constraint conditions represented by expressions: (20), (22), (24), (26), and (27). With these constraint conditions, we are able to obtain a tight upper-bound of the worst-case response time.

Specifically, for any given values of $\delta_{1,h}$, $\delta_{1,e}$, $\delta_{k,0}$, and $\delta_{k,h}$, $k = 2, \ldots, m$, which are constrained by (20), (22), (24), (26), and (27), we can obtain an upper-bound of the worst-case response time, which we denote as $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$. Note that, with any combination of $\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}$, and $\delta_{m,h}$, $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ can always bound the worst-case response time. In order to find a tight upper-bound of the worst-case response time, we can choose a set of $\delta_{k,0}$'s and $\delta_{k,h}$'s to minimize $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ as summarized in the following theorem:

Theorem 1 *In a system with FIFO scheduling under reactive speed scaling, a tight bound of the worst-case response time d can be obtained by the following formula*

$$\begin{aligned} d &= \min\{d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})\} \\ &\textit{subject to} \quad (20), (22), (24), (26), \textit{and } (27). \end{aligned} \tag{28}$$

As we can see, in Theorem 1, if we know the aggregate workload function $F(I)$, then we can obtain $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$. Furthermore, through optimization techniques, we can obtain the work-case response time d. As a case study, in the following, we consider a leaky-bucket task workload and have the following corollary for the worst-case response time under FIFO scheduling:

Corollary 1 *In a system with FIFO scheduling under reactive speed scaling, we consider tasks whose aggregated-task workload is $F(I) = \sigma + \rho I$. Define $\chi_1 = \frac{s_E}{s_H}$ and $\chi_2 = \frac{\rho}{s_H}$. A tight bound of the worst-case response time d is expressed as follows*:

$$d = \begin{cases} V(X - Y), & \chi_2 \le \chi_1^\alpha \\ V(X - Y - Z), & \textit{otherwise} \end{cases} \tag{29}$$

where $V = \frac{(1-\chi_1)(1-\chi_2)}{\chi_1 - \chi_2}$, $X = \frac{\chi_1}{1-\chi_1} d_E$, $Y = \frac{1}{b} \ln \frac{1-\chi_2}{1-\chi_1^\alpha}$, and $Z = \frac{1}{b} \frac{\chi_2}{1-\chi_2} \ln \frac{\chi_2}{\chi_1^\alpha}$. If we define d_H and d_E as the response time when the processor always runs at s_H and s_E respectively, i.e., $d_H = \frac{\sigma}{s_H}$ and $d_E = \frac{\sigma}{s_E}$, then the worst-case response time d is also constrained by

$$d_H \le d \le d_E. \tag{30}$$

The proof is given in Appendix A.

5 Schedulability analysis for SP scheduling

Under SP scheduling, jobs from different tasks are assigned different priorities. Low-priority jobs are preempted by high-priority jobs. We assume all jobs from Task Γ_i are assigned Priority i. A smaller index indicates a higher priority.

In order to perform schedulability analysis in the system with SP scheduling, we introduce the following lemma:

Lemma 3 *For any work-conserving scheduling algorithm in a system under reactive speed scaling as defined in* (11), *the service function $g(t)$ of the aggregated task is uniquely determined by the workload function $f(t)$ of the aggregated task, not by the scheduling algorithm.*

Proof The service function $g(t)$ can be written as

$$g(t) = \int_0^t s(\tau)\,d\tau, \tag{31}$$

where $s(\cdot)$ is the processing speed. According to (11), $s(t)$ is determined by $W(t)$ and $T(t)$ under reactive speed scaling, where $W(t)$ is the backlogged workload at time t, i.e., $W(t) = f(t) - g(t)$, and $T(t)$ is determined by $s(t)$ according to (5). Therefore, the service function $g(t)$ will be uniquely determined by the workload function $f(t)$ of the aggregated task. We have no assumption of the scheduling algorithm. Hence the lemma is proved. □

Based on Lemma 3, we are able to obtain the worst-case response time under SP scheduling as shown in the following theorem:

Theorem 2 *In a system with SP scheduling under reactive speed scaling, a tight bound of the worst-case response time d_i for Task Γ_i can be obtained by the following formula*

$$d_i = \sup_{I \ge 0}\left\{\inf\left\{\tau : \sum_{j=1}^{i-1} F_j(I+\tau) + F_i(I) \le G(I+\tau)\right\}\right\}, \tag{32}$$

where $G(I)$ is defined in (18) *and $\delta_{1,h}$ in $G(I)$ can be obtained by minimizing $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ in Theorem* 1.

Proof We consider a busy interval $[t_1, t_0]$, during which at least one job from Tasks Γ_j $(j \le i)$ is running, and immediately before which no jobs from Tasks Γ_j $(j \le i)$ are running. We know that the response time of a job J of Task Γ_i is introduced by two arrival stages of jobs in the queue: all queued jobs at J's release time and the higher-priority ones coming between J's release time and completion time. Then we have the worst-case response time for a job of Task Γ_i as follows:

$$d_i = \sup_{t \ge t_1}\left\{\inf\left\{\tau : \sum_{j=1}^{i-1} f_j(t+\tau) + f_i(t) \le \sum_{j=1}^{i} g_j(t+\tau)\right\}\right\}, \tag{33}$$

where $f_i(t)$ and $g_i(t)$ are the workload function and the service function of Task Γ_i, respectively.

By our assumption about Interval $[t_1, t_0]$, we have $f_j(t_1) = g_j(t_1)$, $j = 1, \ldots, i$, and $g_j(t) = g_j(t_1)$, $j = i + 1, \ldots, n$. Therefore, $\sum_{j=1}^{i-1} f_j(t + \tau) + f_i(t) \leq \sum_{j=1}^{i} g_j(t + \tau)$ in the above formula can be written as $\sum_{j=1}^{i-1}(f_j(t + \tau) - f_j(t_1)) + (f_i(t) - f_i(t_1)) \leq \sum_{j=1}^{n}(g_j(t + \tau) - g_j(t_1))$. With the similar analysis for FIFO scheduling, the worst-case response time happens when $\sum_{j=1}^{i-1}(f_j(t+\tau) - f_j(t_1)) + (f_i(t) - f_i(t_1)) = \sum_{j=1}^{i-1} F_j(I + \tau) + F_i(I)$, where $I = t - t_1$. Then (32) holds. In (32), $G(I)$ is defined in (18). By Lemma 3, the service function under SP scheduling is same as the one under FIFO scheduling. Then $\delta_{1,h}$ in $G(I)$ can be obtained by minimizing $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ in Theorem 1. □

Similarly, in the following we consider the leaky-bucket task workload as a case study. We have the following corollary on the worst-case response time for SP scheduling:

Corollary 2 *In a system with SP scheduling under reactive speed scaling, we assume that Task Γ_i has a workload constraint function $F_i(I) = \sigma_i + \rho_i I$. A tight bound of the worst-case response time d_i for Task Γ_i can be written as*

$$d_i = \max\{d_{E,i} - \Delta, d_{H,i}\}, \tag{34}$$

where

$$d_{E,i} = \frac{\sum_{j=1}^{i} \sigma_j}{s_E - \sum_{j=1}^{i-1} \rho_j}, \tag{35}$$

$$d_{H,i} = \frac{\sum_{j=1}^{i} \sigma_j}{s_H - \sum_{j=1}^{i-1} \rho_j}, \tag{36}$$

$$\Delta = \frac{\sum_{j=1}^{n} \sigma_j - s_E d}{s_E - \sum_{j=1}^{i-1} \rho_j}, \tag{37}$$

and d in (37) can be obtained by Corollary 1. $d_{E,i}$ and $d_{E,i}$ are the worst-case response times under constant speed scaling with speeds s_E and s_H, respectively.

The proof is given in Appendix B.

6 Schedulability analysis for EDF scheduling

Under EDF scheduling, jobs from a task are assigned priorities on the basis of their deadline. The earlier the deadline, the higher the priority. We assume any job in Task Γ_i is associated with a deadline D_i.

Similar to the schedulability analysis for SP scheduling, we rely on Lemma 3 to obtain the worst-case response time under EDF scheduling as shown in the following theorem:

Theorem 3 *In a system with EDF scheduling under reactive speed scaling, assume Task Γ_i is associated with a deadline D_i, then a tight bound of the worst-case response time d_i for Task Γ_i can be obtained by the following formula*

$$d_i = \sup_{I \geq 0} \left\{ \inf \left\{ \tau : \sum_{j=1}^{n} F_j(I - D_j + D_i) \leq G(I + \tau) \right\} \right\}, \tag{38}$$

where $G(I)$ is defined in (18) and $\delta_{1,h}$ in $G(I)$ can be obtained by minimizing $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ in Theorem 1.

Proof We consider a busy interval $[t_1, t_0]$, during which at least one job is running, and immediately before which no jobs are running. We know that the response time of a job J of Task Γ_i is introduced by two kinds of jobs in the queue: all queued jobs from Γ_i at J's release time and all jobs from Γ_j $(j \neq i)$ with earlier deadline already queued or arriving between J's release time and its completion time. Then we have the worst-case response time for a job of Task Γ_i as follows:

$$d_i = \sup_{t \geq t_1} \left\{ \inf \left\{ \tau : \sum_{j=1}^{n} f_j(t - D_j + D_i) \leq \sum_{j=1}^{n} g_j(t + \tau) \right\} \right\}, \tag{39}$$

where $f_i(t)$ and $g_i(t)$ are the workload function and the service function of Task Γ_i, respectively.

By our assumption about Interval $[t_1, t_0]$, we have $f_j(t_1) = g_j(t_1)$, $j = 1, \ldots, n$. Therefore, $\sum_{j=1}^{n} f_j(t - D_j + D_i) \leq \sum_{j=1}^{n} g_j(t + \tau)$ in the above formula can be written as $\sum_{j=1}^{n}(f_j(t - D_j + D_i) - f_j(t_1)) \leq \sum_{j=1}^{n}(g_j(t + \tau) - g_j(t_1))$. With the similar analysis for SP scheduling, the worst-case response time happens when $\sum_{j=1}^{n}(f_j(t - D_j + D_i) - f_j(t_1)) = \sum_{j=1}^{n} F_j(I - D_j + D_i)$, where $I = t - t_1$. Then (38) holds. In (32), $G(I)$ is defined in (18). By Lemma 3, the service function under EDF scheduling is same as the one under FIFO scheduling. As a result $\delta_{1,h}$ in $G(I)$ can be obtained by minimizing $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \ldots, \delta_{m,0}, \delta_{m,h})$ in Theorem 1. □

For the case of leaky-bucket task workload, we have the following corollary on the worst-case response time for EDF scheduling:

Corollary 3 *In a system with EDF scheduling under reactive speed scaling, we assume that Task Γ_i has a workload constraint function $F_i(I) = \sigma_i + \rho_i I$ and is associated with a deadline D_i. A tight bound of the worst-case response time d_i for Task Γ_i can be written as*

$$d_i = \max\{d_{E,i} - \Delta, d_{H,i}\}, \tag{40}$$

where

$$d_{E,i} = \frac{\sum_{j=1}^{n}(\sigma_j + \rho_j(D_i - D_j))}{s_E}, \tag{41}$$

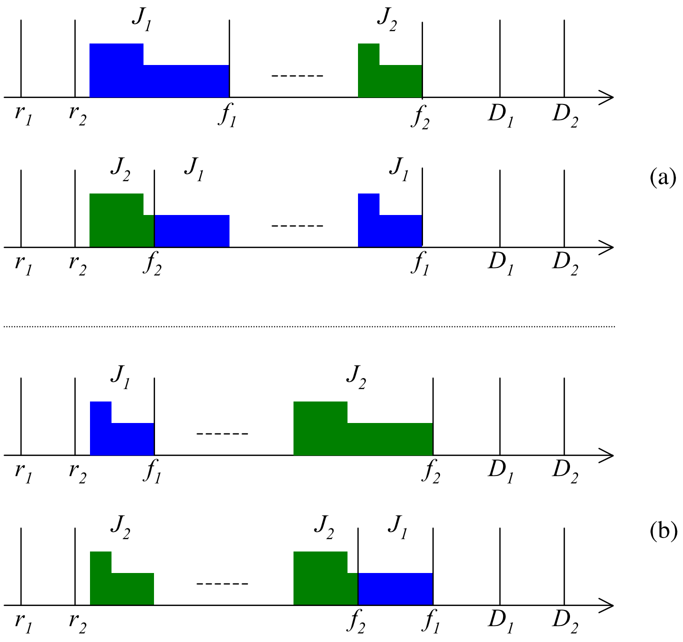


Fig. 6 Transformation of a non-EDF schedule into an EDF schedule

$$d_{H,i} = \frac{\sum_{j=1}^{n}(\sigma_j + \rho_j(D_i - D_j))}{s_H}, \tag{42}$$

$$\Delta = \frac{\sum_{j=1}^{n}\sigma_j}{s_E} - d, \tag{43}$$

and d in (43) can be obtained by Corollary 1. The expressions for $d_{E,i}$ and $d_{H,i}$ are the worst-case response time under constant speed scaling with speeds s_E and s_H, respectively.

The proof is given in Appendix C.

As we know, EDF scheduling is optimal under constant speed scaling (Liu 2000). Fortunately, we find out that EDF remains optimal under reactive speed scaling as shown in the following theorem:

Theorem 4 *EDF scheduling is optimal under reactive speed scaling.*

Proof We use the same approach used in the proof of the optimality of EDF under constant speed scaling (Liu 2000): Any feasible schedule under reactive speed scaling can be systematically transformed into an EDF schedule.

Suppose that in a schedule, we consider two jobs J_1 and J_2 as shown in the top figures in Fig. 6 (a) and (b). We assume each job is scheduled as a whole. Otherwise, we will only consider a successive part of each job. We assume that the deadlines D_2 of J_2 is later than the deadline D_1 of J_1. If the release time r_2 of J_2 is later than the completion time f_1 of J_1 (r_2 is not shown in the figures), J_2 cannot be scheduled before f_1, and the two jobs are already scheduled on the EDF basis. Therefore, in the following, we assume that r_2 is no later than f_1.

Without loss of generality, we assume that r_2 is no later than the beginning execution time of J_1. To transform the given schedule, we swap J_1 and J_2 as shown in the figure. Specifically, if the number of processor cycles of J_1 is smaller than the one of J_2 as shown in Fig. 6(a), we move the portion of J_2 that fits in the original J_1 and move the entire portion of J_1 backward to J_2 and place it after J_2. The swap is always possible because the following fact: By Lemma 3, the service function is uniquely determined by the workload function and not by the scheduling algorithm such as FIFO, SP, or EDF. And so are the temperature and speed profiles. If the number of processor cycles of J_1 is larger than the one of J_2 as shown in Fig. 6(b), we can do a similar swap. We repeat this transformation for every pair of jobs that are not scheduled on the EDF basis until no such pair exists.

Since both reactive speed scaling and the considered scheduling algorithms are work-conserving, there will not exist a case that some interval is left idle while there are jobs ready for execution but scheduled in a later interval.

The above analysis shows that we can always transform a feasible schedule under reactive speed scaling into an EDF schedule. Therefore, EDF scheduling is optimal under reactive speed scaling. □

7 Performance evaluation

In this section we quantify the benefit of using simple reactive speed scaling by comparing the worst-case response time with that of a system without speed scaling. We adopt as the baseline a constant-speed processor that runs at equilibrium speed s_E.

We choose the same setting as Skadron et al. (2003) for a silicon chip. The thermal conductivity of the silicon material per unit volume is $k_{th} = 100$ W/m K and the thermal capacitance per unit volume is $c_{th} = 1.75e+6$ J/m^3 K. The chip is $t_{th} = 0.55$ mm thick. Therefore, the thermal RC time constant $RC = \frac{c_{th}}{k_{th}} t_{th}^2 = 5.3$ ms (Skadron et al. 2003). We choose $\beta_0 = 1$, $\beta_1 = 0.001$, and $\beta_2 = 0.1$. Hence by (4) $a = 5.7e{-7}$ and $b \approx 188.9$ s^{-1}. The ambient temperature is 45°C and the maximum temperature threshold is 85°C, hence $T_H = 40$°C. The equilibrium speed s_E is fixed by the system with (10) and we choose $s_H = \frac{10}{7} s_E$ and assume $\alpha = 3$.

We consider three tasks Γ_i's ($i = 1, 2, 3$). Each task Γ_i has a leaky bucket arrival with $F_i(I) = \sigma_i + \rho_i I$. The aggregate task has an arrival with $F(I) = \sigma + \rho I$, where $\sigma = \sum_{i=1}^{3} \sigma_i$ and $\rho = \sum_{i=1}^{3} \rho_i$. In our evaluation, we vary σ/s_E and ρ/s_E in the ranges of $[0, 0.005]$ and $[0, 0.5]$ respectively. We compare the worst-case response time of jobs in the system under reactive speed scaling and the baseline one in the systems the processor always run at the equilibrium speed.

First we consider FIFO scheduling. We evaluate the worst-case response time decrease ratio $(d_E - d)/d_E$ for the aggregated task.[4] Figure 7 shows a contour plot of

[4]The alert reader has noticed that we did not define a value for parameter a. This is because a appears only in the computation of s_E, which cancels out in the response time decrease ratio.

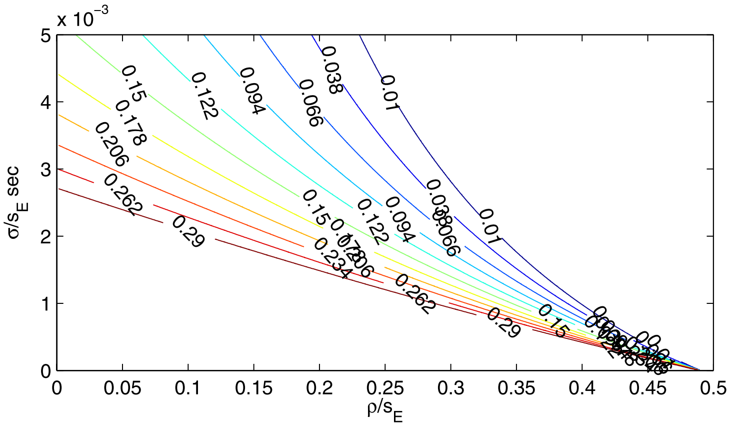


Fig. 7 A contour plot of response time decrease ratio $(d_E - d)/d_E$ for the aggregated task under reactive speed scaling for FIFO scheduling

$(d_E - d)/d_E$ in terms of σ/s_E and ρ/s_E. Based on (29) and Fig. 7, we observe that the response time decrease ratio changes from a minimum 0 (as $d = d_E$) to a maximum of $1 - \frac{s_E}{s_H} = 0.300$ (as $d = d_H$). The response time decrease ratio will decrease as either σ or ρ increases.

Next we consider SP scheduling. We assume that $\sigma_1 : \sigma_2 : \sigma_3 = \rho_1 : \rho_2 : \rho_3 = 1 : 2 : 3$. We evaluate the worst-case response time decrease ratio $(d_{E,i} - d_i)/d_{E,i}$ for Task Γ_i. Each individual picture in Fig. 8 shows contour plots of $(d_{E,i} - d_i)/d_{E,i}$ in terms of σ/s_E and ρ/s_E, for the three tasks separately. Based on (34) and Fig. 8, we observe that the response time decrease ratio changes from a minimum of 0 (as $d_i = d_{E,i}$) to a maximum of $1 - \frac{s_E}{s_H} = 0.300$ for Task Γ_1, to a maximum of $(1 - \frac{s_E}{s_H})/(1 - \frac{1}{6}\frac{s_E}{s_H}) = 0.316$ for Task Γ_2, and to a maximum of $(1 - \frac{s_E}{s_H})/(1 - \frac{1}{2}\frac{s_E}{s_H}) = 0.353$ for Task Γ_3 (as $d_i = d_{E,i}$).

As long as the response time decrease ratio is not larger than 0.300, the ratio will decrease as either σ or ρ increases for any task. Whenever it exceeds 0.300, we have different observation results for the lower-priority tasks. In particular, considering the lower-priority task Γ_3, once d_i reaches $d_{H,i}$, the response time decrease ratio can be written as $(1 - \frac{s_E}{s_H})/(1 - \frac{1}{s_H}\sum_{j=1}^{i-1}\rho_j) = 0.300/(1 - \frac{1}{s_H}\sum_{j=1}^{i-1}\rho_j)$. Therefore, as shown at the left-bottom corner of the last two contour plots in Fig. 8, the response time decrease ratio will keep constant as σ increases and ρ keeps constant, but increase beyond 0.300 as ρ increases and σ keeps constant.

Finally, we consider EDF scheduling. We consider the same assumption that $\sigma_1 : \sigma_2 : \sigma_3 = \rho_1 : \rho_2 : \rho_3 = 1 : 2 : 3$ and the deadlines are $D_1 = 0.004$, $D_2 = 0.006$, $D_3 = 0.008$. We evaluate the worst-case response time decrease ratio $(d_{E,i} - d_i)/d_{E,i}$ for Task Γ_i. Each individual picture in Fig. 9 shows contour plots of $(d_{E,i} - d_i)/d_{E,i}$ in terms of σ/s_E and ρ/s_E, for the three tasks separately. Based on (40) and Fig. 9, we observe that the response time decrease ratio changes from a minimum of 0 (as $d_i = d_{E,i}$) to a maximum of $1 - \frac{s_E}{s_H} = 0.300$ for all tasks. And the response time decrease ratio decreases with the increase of either σ or ρ, except for the case of

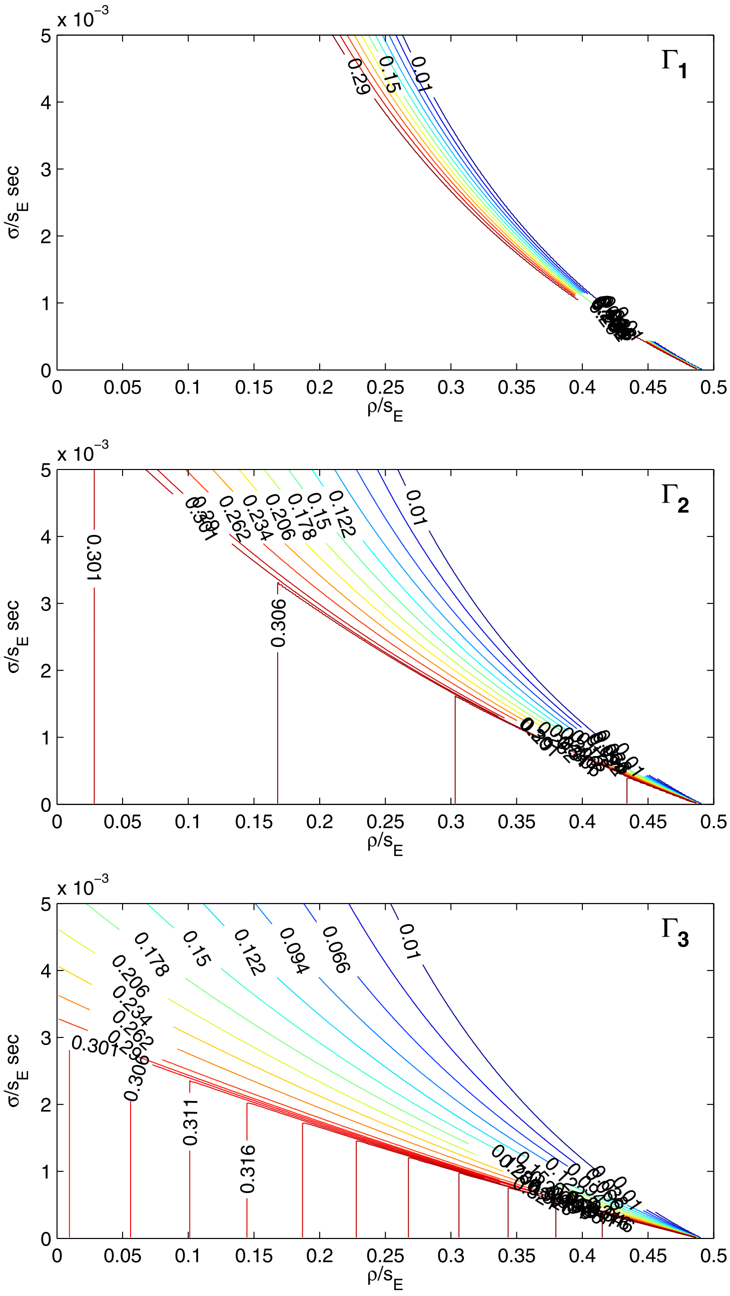


Fig. 8 Contour plots of response time decrease ratio $(d_{E,i} - d_i)/d_{E,i}$ for Task Γ_i under reactive speed scaling for SP scheduling

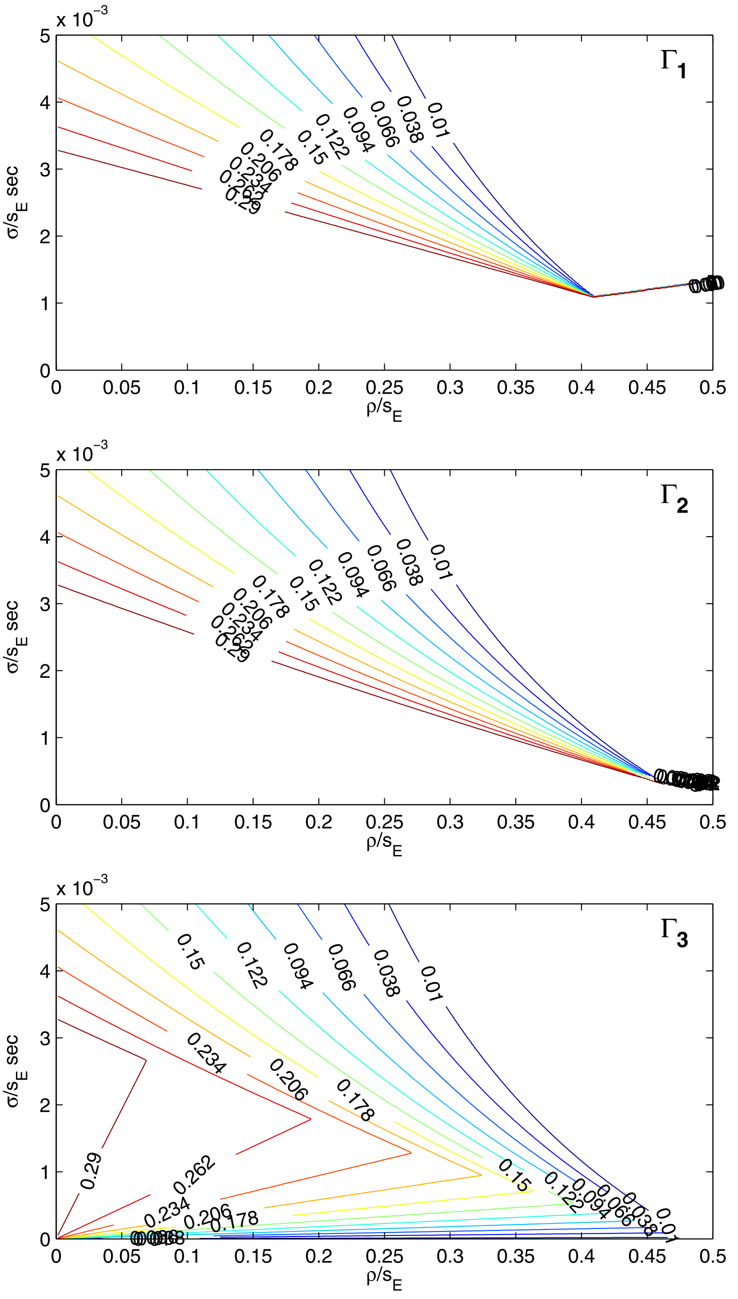


Fig. 9 Contour plots of response time decrease ratio $(d_{E,i} - d_i)/d_{E,i}$ for Task Γ_i under reactive speed scaling for EDF scheduling

Γ_3. In this exception, once $d_3 < d_{E,3}$ (i.e., the response time decrease ratio is below $1 - \frac{s_E}{s_H} = 0.300$), by (40), we have $(d_{E,3} - d_3)/d_{E,3} = \frac{1}{1+\zeta_3}(d_E - d)/d_E$, where $\zeta_3 = \frac{1}{\sigma}\sum_{j=1}^{3}\rho_j(D_3 - D_j) \geq 0$ and $(d_E - d)/d_E$ is the response time decrease ratio for FIFO as shown in Fig. 7. We consider the following two cases:

– Case $(d_E - d)/d_E < 0.300$: In this case, $\zeta_3 \approx 0$, then we have $(d_{E,3} - d_3)/d_{E,3} \approx (d_E - d)/d_E$ and the curve will be similar to Fig. 7.
– Case $(d_E - d)/d_E = 0.300$: In this case, then we have $(d_{E,3} - d_3)/d_{E,3} = \frac{0.300}{1+\zeta_3}$: as σ increases, ζ_3 decreases, then $(d_{E,3} - d_3)/d_{E,3}$ increases; as ρ increases, ζ_3 increases, then $(d_{E,3} - d_3)/d_{E,3}$ decreases.

8 Conclusion and future work

Schedulability analysis in systems with thermal-constrained speed scaling is difficult, as the traditional definition of "busy period" does not apply, and it becomes difficult to separate the execution of jobs from the interference by ones arriving earlier or having low priorities because of dynamic speed scaling triggered by the thermal behavior. In this paper we have shown how to compute bounds on the worst-case response time for tasks with arbitrary job arrivals for FIFO, SP, and EDF scheduling algorithms in a system with reactive speed scaling algorithm, which simply runs at maximum speed until the CPU becomes idle or reaches a critical temperature. In the latter case the processing speed is reduced (through DVS or appropriate clock throttling) to an equilibrium speed that keeps the temperature constant. We have shown that such a scheme reduces worst-case response time.

In order to further improve the performance of speed scaling, one would have to find ways to partially isolate jobs from the thermal effects of ones arriving earlier or having low priorities. One weakness of the proposed speed-scaling algorithm is its inability to pro-actively process low-priority tasks at lower-than-equilibrium speeds.

Acknowledgements An earlier version of this work was published in the *Proceedings of IEEE Real-Time Systems Symposium*, December 2006 (Wang and Bettati 2006). This work was funded by NSF Grant No. CNS-0509483 and Rackham Faculty Research Grant at the University of Michigan and NSF CAREER Grant No. CNS-0746906.

Appendix A: Proof of Corollary 1

We follow the analysis in Sect. 4 with the leaky bucket workload.

A.1 Single busy-period analysis

Since $F(I) = \sigma + \rho I$, by (20) we can obtain the response time d as

$$d = \max\left\{\frac{\sigma}{s_H}, \frac{\sigma}{s_E} - \left(\frac{s_H}{s_E} - 1\right)\delta_{1,h}\right\}. \tag{44}$$

Therefore, we have

$$\delta_{1,h} = \frac{\frac{\sigma}{s_E} - d}{\frac{s_H}{s_E} - 1}, \tag{45}$$

as

$$d \geq \frac{\sigma}{s_H}. \tag{46}$$

A.2 Service in extended busy period

To simplify the service analysis, we consider equal intervals and assume $\delta_k = \delta$ for $k = 3, \ldots, m$. We investigate the service received in each interval $[t_k, t_0]$, $k = 1, \ldots, m$.

As $k = 1$, by (22) we have

$$s_H \delta_{1,h} + s_E \delta_{1,e} = \sigma + \rho(\delta_{1,h} + \delta_{1,e}). \tag{47}$$

Hence,

$$(s_H - \rho)\delta_{1,h} + (s_E - \rho)\delta_{1,e} = \sigma. \tag{48}$$

As $k = 2, \ldots, m$, by (24), we have

$$s_H \sum_{j=1}^{k} \delta_{j,h} + s_E \delta_{1,e} = \sigma + \rho((k-2)\delta + (\delta_2 + \delta_1) + d). \tag{49}$$

If $k = 2$, together with (48), we have

$$\delta_{2,h} = \frac{\delta_{2,0} + d}{\frac{s_H}{\rho} - 1}. \tag{50}$$

If $k \geq 3$, we have

$$\delta_{k,h} = \frac{\rho}{s_H}\delta. \tag{51}$$

A.3 Thermal interference in extended busy period

By (45) and (51), we can rewrite the thermal constraint condition (26).

As $k = 2, \ldots, m-1$,

$$\frac{T_k}{T_H} = e^{-b(m-k)\delta}(1 - \xi) + \xi, \tag{52}$$

where

$$\xi = \left(\frac{s_H}{s_E}\right)^{\alpha} \frac{1 - e^{-b\frac{\rho}{s_H}\delta}}{1 - e^{-b\delta}}. \tag{53}$$

By (52), T_2 is a function of δ and m. It is easy to show that the smaller T_2 is, the shorter the response time d. Therefore, we want to find δ and m to minimize T_2 so that we have a tight upper-bound d of the original worst-case response time.

If $\xi \leq 1$, then $T_k/T_H \leq 1$. By (52), T_2 is a decreasing function in terms of $(m-2)\delta$, then $T_2/T_H \geq \lim_{(m-2)\delta\to\infty} T_2/T_H = \xi$.[5] Furthermore, ξ is an increasing function of δ, then $T_2/T_H \geq \lim_{\delta\to 0}\xi = (\frac{s_H}{s_E})^\alpha \frac{\rho}{s_H}$. Therefore, we choose the minimum and set $T_2/T_H = (\frac{s_H}{s_E})^\alpha \frac{\rho}{s_H}$ as $(\frac{s_H}{s_E})^\alpha \frac{\rho}{s_H} \leq 1$.

If $\xi > 1$, then T_2/T_H is the maximum among all T_k/T_H's. Therefore, we only need to consider bound $T_2/T_H \leq 1$. By (52), T_2/T_H is an increasing function in terms of m, then T_2/T_H will be minimized at $m = 2$. Hence, we set $T_2/T_H = 1$ in this case.

Therefore, with the analysis above, we can set

$$\frac{T_2}{T_H} = \min\left\{\left(\frac{s_H}{s_E}\right)^\alpha \frac{\rho}{s_H}, 1\right\}. \tag{54}$$

At the same time, by (25) and (27), we have

$$\delta_{1,h} + \delta_{2,h} = \frac{1}{b}\ln\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_2}{T_H}e^{-b\delta_{2,0}}}{(\frac{s_H}{s_E})^\alpha - 1}. \tag{55}$$

Therefore, by (45), (50), (54), and (55), we can obtain the worst-case response time d as follows:

$$d = \frac{(1-\chi_1)(1-\chi_2)}{\chi_1-\chi_2}\left(\frac{\chi_1}{1-\chi_1}\frac{\sigma}{s_E} + \frac{\chi_2}{1-\chi_2}\delta_{2,0} - \frac{1}{b}\ln\frac{1-\min\{\chi_2,\chi_1^\alpha\}e^{-b\delta_{2,0}}}{1-\chi_1^\alpha}\right), \tag{56}$$

where $\chi_1 = \frac{s_E}{s_H}$, and $\chi_2 = \frac{\rho}{s_H}$.

Equation (56) shows that d is a function of $\delta_{2,0}$. Since the above analysis works for any chosen $\delta_{2,0}$, we want to obtain a minimum d in terms of $\delta_{2,0}$. There are two cases:

- $\chi_2 \leq \chi_1^\alpha$: d will be minimized at $\delta_{2,0} = 0$, therefore

$$d = V(X - Y), \tag{57}$$

- $\chi_2 > \chi_1^\alpha$: d will be minimized at $\delta_{2,0} = \frac{1}{b}\ln\frac{\chi_2}{\chi_1^\alpha}$, therefore

$$d = V(X - Y - Z), \tag{58}$$

where V, X, Y, Z are defined in Corollary 1.

On the other hand, in the thermal constraint, as $k = 1$, by (27) and the constraint that $T_1/T_H \leq 1$, we have

$$\delta_{1,h} \geq 0. \tag{59}$$

[5] We assume that at time zero the system is at lowest temperature. Therefore, we can pick the intervals with overall length up to infinity.

Therefore, by (45), we have

$$d \leq \frac{\sigma}{s_E}. \tag{60}$$

Recall that $d_H = \frac{\sigma}{s_H}$ and $d_E = \frac{\sigma}{s_E}$, then by (46) and (60), the worst-case response time is also constrained by

$$d_H \leq d \leq d_E. \tag{61}$$

Appendix B: Proof of Corollary 2

By Theorem 2, $G(I) = \min\{(s_H - s_E)\delta_{1,h} + s_E(I + d_i), s_H(I + d_i)\}$ depends on $\delta_{1,h}$. For the leaky bucket task workload, by (45) we have $\delta_{1,h} = (\frac{\sigma}{s_E} - d)/(\frac{s_H}{s_E} - 1)$, where d can be obtained by Corollary 1.

By (32), the response time formula can be written as

$$\begin{aligned} d_i = \sup_{I\geq 0}\Bigg\{\inf\Bigg\{\tau : &\sum_{j=1}^{i-1}(\sigma_j + \rho_j(I+\tau)) + \sigma_i + \rho_i I \\ &\leq \min\{(s_H - s_E)\delta_{1,h} + s_E(I+\tau), s_H(I+\tau)\}\Bigg\}\Bigg\}. \end{aligned} \tag{62}$$

Then, as $d_i > \delta_{1,h}$, with $\delta_{1,h} = (\frac{\sigma}{s_E} - d)/(\frac{s_H}{s_E} - 1)$, we have

$$\begin{aligned} d_i &= \frac{\sum_{j=1}^{i-1}(\sigma_j + \rho_j d_i) + \sigma_i}{s_E} - \left(\frac{s_H}{s_E} - 1\right)\delta_{1,h} \\ &= \frac{\sum_{j=1}^{i-1}(\sigma_j + \rho_j d_i) + \sigma_i}{s_E} - \left(\frac{\sigma}{s_E} - d\right) \end{aligned} \tag{63}$$

otherwise

$$d_i = \frac{\sum_{j=1}^{i-1}(\sigma_j + \rho_j d_i) + \sigma_i}{s_H}. \tag{64}$$

Therefore, by (63) and (64), we have

$$d_i = \max\{d_{E,i} - \Delta, d_{H,i}\}, \tag{65}$$

where $d_{E,i}$, $d_{H,i}$, and Δ are defined in Corollary 2, and d can be obtained by Corollary 1. The worst-case response time d_i is constrained by

$$d_{H,i} \leq d_i \leq d_{E,i}. \tag{66}$$

Appendix C: Proof of Corollary 3

By Theorem 3, $G(I)=\min\{(s_H-s_E)\delta_{1,h}+s_E(I+d_i), s_H(I+d_i)\}$ depends on $\delta_{1,h}$. For the leaky bucket task workload, by (45) we have $\delta_{1,h}=(\frac{\sigma}{s_E}-d)/(\frac{s_H}{s_E}-1)$, where d can be obtained by Corollary 1.

By (38), the response time formula can be written as

$$d_i=\sup_{I\geq 0}\left\{\inf\left\{\tau:\sum_{j=1}^{n}(\sigma_j+\rho_j(I-D_j+D_i))\right.\right.$$
$$\left.\left.\leq \min\{(s_H-s_E)\delta_{1,h}+s_E(I+\tau), s_H(I+\tau)\}\right\}\right\}. \tag{67}$$

Then, as $d_i>\delta_{1,h}$, with $\delta_{1,h}=(\frac{\sigma}{s_E}-d)/(\frac{s_H}{s_E}-1)$, we have

$$\begin{aligned} d_i &= \frac{\sum_{j=1}^{n}(\sigma_j+\rho_j(D_i-D_j))}{s_E}-\left(\frac{s_H}{s_E}-1\right)\delta_{1,h} \\ &= \frac{\sum_{j=1}^{n}\rho_j(D_i-D_j)}{s_E}+d, \end{aligned} \tag{68}$$

otherwise

$$d_i=\frac{\sum_{j=1}^{n}(\sigma_j+\rho_j(D_i-D_j))}{s_H}. \tag{69}$$

Therefore, by (68) and (69), we have

$$d_i=\max\{d_{E,i}-\Delta, d_{H,i}\}, \tag{70}$$

where $d_{E,i}$, $d_{H,i}$, and Δ are defined in Corollary 3, and d can be obtained by Corollary 1. The worst-case response time d_i is constrained by

$$d_{H,i}\leq d_i\leq d_{E,i}. \tag{71}$$

References

Advanced configuration and power interface specification (2010) http://www.acpi.info/spec.htm. The last access time is July 2010

Semiconductor Industry Association (2005) 2005 international technology roadmap for semiconductors. http://public.itrs.net. The last access time is July 2010

Bansal N, Kimbrel T, Pruhs K (2005) Dynamic speed scaling to manage energy and temperature. In: IEEE symposium on foundations of computer science

Bansal N, Pruhs K (2005) Speed scaling to manage temperature. In: Symposium on theoretical aspects of computer science

Brooks D, Martonosi M (2001) Dynamic thermal management for high-performance microprocessors. In: The 7th international symposium on high-performance computer architecture, pp 171–182

Chantem T, Dick RP, Hu XS (2008) Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. In: Design, automation and test in Europe

Chen J-J, Hung C-M, Kuo T-W (2007) On the minimization of the instantaneous temperature for periodic real-time tasks. In: IEEE real-time and embedded technology and applications symposium

Cohen A, Finkelstein L, Mendelson A, Ronen R, Rudoy D (2003) On estimating optimal performance of CPU dynamic thermal management. In: Computer architecture letters

Cohen A, Finkelstein L, Mendelson A, Ronen R, Rudoy D (2006) On estimating optimal performance of CPU dynamic thermal management. In: Computer architecture letters

Dhodapkar A, Lim CH, Cai G, Daasch WR (2000) TEMPEST: a thermal enabled multi-model power/performance estimator. In: Workshop on power-aware computer systems, ASPLOS-IX

Ferreira AP, Oh J, Moss D (2006) Toward thermal-aware load-distribution for real-time server. In: IEEE real-time systems symposium work-in-progress session

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 York

Rabaey JM, Chandrakasan A, Nikolic B (2002) Digital integrated circuits, 2nd edn. Prentice Hall, New York

Rao R, Vrudhula S, Chakrabarti C, Chang N (2006) An optimal analytical solution for processor speed control with thermal constraints. In: International symposium on low power electronics and design. ACM Press, New York

Rotem E, Naveh A, Moffie M, Mendelson A (2004) Analysis of thermal monitor features of the Intel Pentium M processor. In: Workshop on temperature-aware computer systems

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

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

Srinivasan J, Adve SV (2003) Predictive dynamic thermal management for multimedia applications. In: International conference on supercomputing

Tiwari V, Singh D, Rajgopal S, Mehta G, Patel R, Baez F (1998) Reducing power in high-performance microprocessors. In: Design automation conference, pp 732–737

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

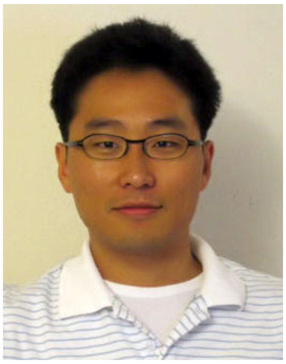
Wang S, Bettati R (2008) Reactive speed control in temperature-constrained real-time systems. Real-Time Syst J 39(1–3), 658–671

Wu J, Liu J, Zhao W (2005) On schedulability bounds of static priority schedulers. In: IEEE real-time and embedded technology and applications symposium

Xu R, Zhu D, Rusu C, Melhem R, Moss D (2005) Energy efficient policies for embedded clusters. In: ACM SIGPLAN/SIGBED conference on languages, compilers, and tools for embedded systems

Zhang S, Chatha KS (2007) Approximation algorithm for the temperature-aware scheduling problem. In: IEEE/ACM international conference on computer-aided design

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. He is a recipient of the US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award. His research interests are in real-time systems, networking and distributed systems, and security and privacy.



Youngwoo Ahn received his B.S. degree in electrical engineering from Seoul National University, Korea, in 1997 and his M.S. degree from Seoul National University, Korea, in 1999. During 1999–2004, he worked as a research engineer at LG Electronics in Korea. Also he worked at electronics and Telecommunication Research Institute in Korea from 2004 to 2005 as a researcher. He graduated with the Ph.D. in electrical and computer engineering from Texas A& M University in August 2010. His research interests lie primarily in real-time operating systems, especially in designing and analyzing task scheduling under resource constraints.

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