Appeared in: Proceedings of the 2006 IEEE Real-Time Systems Symposium (RTSS 2006), Rio de Janeiro, BRAZIL, December 2006.

Delay Analysis in Temperature-Constrained Hard Real-Time Systems with General Task Arrivals

Shengquan Wang
The University of Michigan - Dearborn
Dearborn, MI 48128, USA
shqwang@umd.umich.edu

Riccardo Bettati
Texas A&M University
College Station, TX 77843, USA
bettati@cs.tamu.edu

Abstract

In this paper, we study temperature-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 simple reactive speed control technique in our work. We design a methodology to perform delay analysis for general task arrivals under reactive speed control with First-In-First-Out (FIFO) scheduling and Static-Priority (SP) scheduling. As a special case, we obtain a close-form delay formula for the leaky-bucket task arrival model. Our data show how simple reactive speed control can decrease the delay of tasks compared with any constant-speed scheme.

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., [1]).

Thermal management through packaging (that improves airflow, for example) and active heat dissipation 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 [2]. In addition, the packaging requirements and operating environments of many high-performance embedded devices render such approaches inappropriate.

This work was funded by NSF under Grant No. CNS-0509483, while Dr. Wang was at Texas A&M University.

A number of dynamic thermal management 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:

- The Pentium 4 Series processors uses *Clock Throttling* [3] or *Clock Gating* [4] to stall the clock and so allow the processor to cool during thermal overload.
- *Dynamic Voltage Scaling* (DVS) [1] 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) [3].
- 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. [4] 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 [5].

As high-performance embedded systems become increasingly temperature-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 delay analysis techniques in temperature-constrained hard real-time systems, where *deadline* constraints for tasks have to be balanced against *temperature* constraints of the system.

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. The work on dynamic speed scaling techniques to control temperature in real-time systems was initiated in [6] and further investigated in [7]. Both [6] and [7] 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 deterministic task model (e.g., periodic tasks) and so allows for design-time delay analysis.

We distinguish between proactive and reactive speed scaling schemes. Whenever the temperature model is known, the scheduler could in principle use a *proactive* speed-scaling approach, where – similarly to a non-work-conserving scheduler – resources are preserved for future use. In this paper, we limit ourselves to *reactive* schemes, and propose a simple reactive speed scaling technique for the processor, which will be discussed in Section 2. We focus on reactive schemes primarily because they are simple to integrate with current processor capabilities through the ACPI power control framework [8, 9]. In our previous paper [10], we motivate the reactive scheme and perform delay analysis for identical-period tasks. In this paper, we extend it to general task arrivals with First-in First-out (FIFO) scheduling and Static-Priority (SP) scheduling.

The rest of the paper is organized as follows. In Section 2, we introduce the thermal model, speed scaling schemes, and task model and scheduling algorithms. After introducing two important lemmas in Section 3, we design the methodology to perform delay analysis for FIFO and SP scheduling algorithms in Sections 4 and 5 respectively. We measure the performance in Section 6. Finally, we conclude our work with final remarks and give an outlook on future work in Section 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.* [11] in TEMPEST. Other models, such as used in HotSpot [4], 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 [6, 7, 11, 12]. While this model does not capture fine-granularity thermal effects, the authors in [4] 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 allows 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 [5]. 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 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 $\alpha = 3$ [6, 7].

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 [6, 7, 12]:

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

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:

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

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

$$T(t) = \int_{t_0}^{t} as^{\alpha}(\tau)e^{-b(t-\tau)}d\tau + T_0 e^{-b(t-t_0)}. \tag{5}$$

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

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

- 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:

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

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

2.2 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

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

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

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

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

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

For any constant processor speed not exceeding s_E, the processor does not exceed temperature T_H. 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 that (a) it must be supported by existing power control frameworks such as ACPI [8,9], and (b) it must lead to tractable design – time delay analysis. 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.

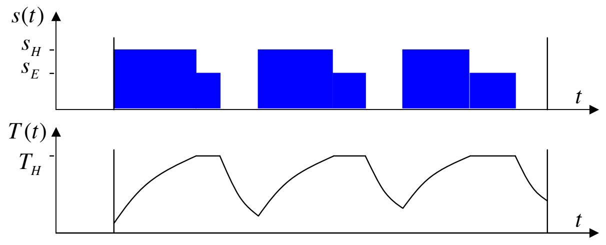

Figure 1. Illustration of reactive speed scaling.

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.

2.3 Task Model and Scheduling Algorithms

The workload consists of a set of tasks $\{\Gamma_i : i = 1, 2, \ldots, n\}$. 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 *delay* of the job, and the worst-case delay 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 $[0, t]$.

A time-independent representation of $f_i(t)$ is the 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 $0 \leq I \leq t$,*

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

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

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

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 delay experienced by jobs. In this paper, we consider two scheduling algorithms: *First-in First-out (FIFO)* scheduling and *Static Priority (SP)* scheduling.

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 (11). 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, job execution will affect the temperature at a later time or the delay of a later job.

Lemma 1. *In a system under our 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 Figure 2:*

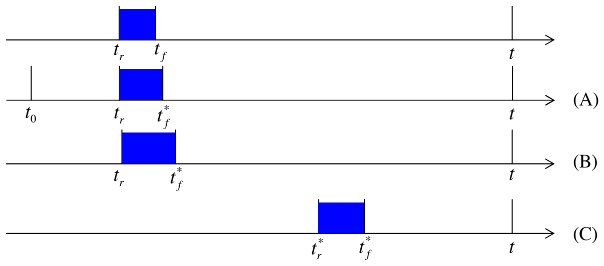

Figure 2. Temperature effect.

- *Action A: Increasing 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: 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 $T_t \leq T_t^*$, 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 under our 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 Figure 3:*

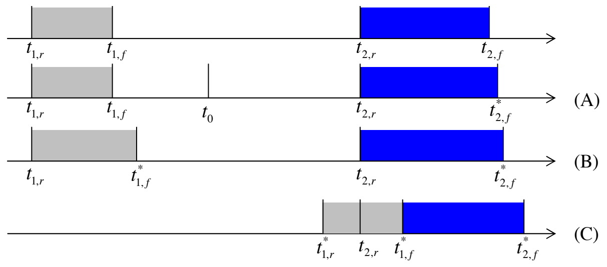

Figure 3. Delay effect.

- *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 delay 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 [13]. 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 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 two sections.

4 Delay Analysis of 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 delay 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 delay 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 delay analysis in the system with FIFO scheduling. Under FIFO scheduling, all tasks experience the same worst-case delay as the aggregated task does. Therefore, we consider the aggregated task, whose workload constraint function can be written as $F(I) = \sum_{i=1}^{n} F_i(I)$. First, we investigate the worst-case delay for the aggregated task.

Delay Constraint We consider a busy period $[t_1, t_0]$ with length δ_1 during which a job will experience the longest delay 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 Figure 4(a).

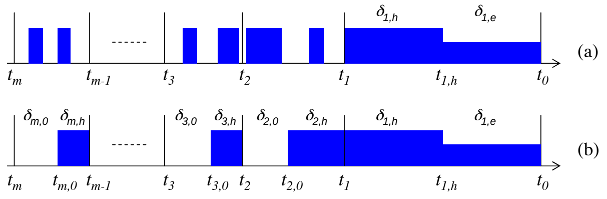

Figure 4. Job executions.

We define d as the worst-case delay experienced by a job in the busy period $[t_1, t_0]$. Then, by the definition of worst-case delay, we have

$$d = \sup_{t \geq t_1} \{\inf\{\tau : f(t) \leq g(t+\tau)\}\}, \tag{14}$$

where $f(t)$ and $g(t)$ are the workload function and the service function of the aggregated task respectively, as defined in Section 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 delay 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 (14) can be written as

$$f(t) - f(t_1) \leq g(t+\tau) - g(t_1). \tag{15}$$

First, we study the right side of (15). 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

$$g(t+\tau) - g(t_1) = G(I+\tau), \tag{16}$$

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

$$G(I) = \min\{(s_H - s_E)\delta_{1,h} + s_E I, s_H I\}. \tag{17}$$

Next, we study the left side of (15). With Action B, the job will experience a longer delay with more workload released and completed before the completion of this job. Therefore, if we set

$$f(t) - f(t_1) = F(I), \tag{18}$$

together with (16), then the worst-case delay in (15) can be written as (see Figure 5)

$$d = \sup_{I \geq 0} \{\inf\{\tau : F(I) \leq G(I+\tau)\}\}. \tag{19}$$

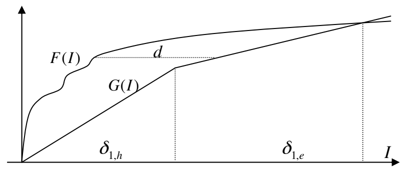

Figure 5. Delay constraint.

As we can see, the undetermined service constraint function $G(I)$ is the key in the worst-case delay formula (19). Next, we will focus on obtaining $G(I)$.

Service Constraint As defined in (17), $G(I)$ is a function of $\delta_{1,h}$, which depends on the temperature at time t_1. Instead of determining the exact temperature at t_1, we aim to obtain a tight upper-bound of t_1, which will result in an upper-bound of the worst-case delay 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 Figure 4(a). By Lemma 1, we can use the three actions mentioned above to upper-bound the temperature at t_1. With Action A, we upper-bound the temperature at t_m to be 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 Figure 4(b). We assume that the temperature will not hit T_H during $[t_m, t_1]$ [1], then

[1]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.

the processor will run at high speed s_H during each interval $[t_{k+1,0}, t_k]$.

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

$$g(t_0) - g(t_k) = s_H \sum_{j=1}^{k} \delta_{j,h} + s_E \delta_{1,e}. \qquad (20)$$

For $k = 1$, we have $g(t_0) - g(t_1) = f(t_0) - f(t_1)$. Following the delay analysis in the above delay constraint, we consider the worst-case workload $f(t_0) - f(t_1) = F(t_0 - t_1)$. Therefore, by (20) we have

$$s_H \delta_{1,h} + s_E \delta_{1,e} = F(\delta_{1,h} + \delta_{1,e}). \qquad (21)$$

For $k = 2, \dots, m$, by the definition of the worst-case delay, the number of processor cycles in Interval $[t_k, t_0]$ is bounded as $g(t_0) - g(t_k) \leq f(t_0) - f(t_k - d)$. By Lemma 2, the delay becomes 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 (20) we have

$$s_H \sum_{j=1}^{k} \delta_{j,h} + s_E \delta_{1,e} = F(\sum_{j=1}^{k} \delta_{j,h} + \delta_{1,e} + d). \qquad (22)$$

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.

Temperature Constraint First, we consider each interval $[t_{k+1}, t_k]$, $k = 1, \dots, m-1$, which is composes 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

$$T_k = \frac{a s_H^\alpha}{b} + (T_{k+1} e^{-b\delta_{k+1,0}} - \frac{a s_H^\alpha}{b}) e^{-b\delta_{k+1,h}}. \qquad (23)$$

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

$$\begin{aligned} \frac{T_k}{T_H} &= (\frac{s_H}{s_E})^\alpha \sum\nolimits_{r=k+1}^{m} e^{-b\sum_{l=k+1}^{r-1} \delta_l} (1 - e^{-b\delta_{r,h}}) \\ &\quad + e^{-b\sum_{l=k+1}^{m} \delta_l} \leq 1. \end{aligned} \qquad (24)$$

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

$$\frac{T_1}{T_H} = (\frac{s_H}{s_E})^\alpha - ((\frac{s_H}{s_E})^\alpha - 1) e^{b\delta_{1,h}}. \qquad (25)$$

Therefore, for any given values of $\delta_{1,h}$, $\delta_{1,e}$, $\delta_{k,0}$, and $\delta_{k,h}$, $k = 2, \dots, m$, which are constrained by the above constraint conditions (19), (21), (22), (24), and (25), we can obtain an upper-bound of the worst-case delay, which we denote as $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \dots, \delta_{m,0}, \delta_{m,h})$. Note that $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \dots, \delta_{m,0}, \delta_{m,h})$ can always bound the worst-case delay. In order to find a tight upper-bound of the worst-case delay, we can choose a set of $\delta_{k,0}$'s and $\delta_{k,h}$'s to minimize $d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \dots, \delta_{m,0}, \delta_{m,h})$ as summarized in the following theorem:

Theorem 1. *In a system with FIFO scheduling under reactive speed scaling, the worst-case delay d can be obtained by the following formula*

$$\begin{aligned} d &= \min\{d(\delta_{1,h}, \delta_{1,e}, \delta_{2,0}, \delta_{2,h}, \dots, \delta_{m,0}, \delta_{m,h})\} \\ &\quad \text{subject to } \textit{(19), (21), (22), (24), and (25)}. \end{aligned} \qquad (26)$$

As a case study, in the following, we consider a leaky-bucket task workload and have the following theorem for the worst-case delay with FIFO scheduling:

Corollary 1. *In a system with FIFO scheduling under reactive speed scaling, we consider tasks with leaky-bucket workload and the workload constraint function of the aggregated task 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 delay d is expressed as follows:*

$$d = \begin{cases} V(X - Y), & \chi_2 \leq \chi_1^\alpha \\ V(X - Y - Z), & \textit{otherwise} \end{cases} \qquad (27)$$

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 $Z = \frac{1}{b} \frac{\chi_2}{1-\chi_2} \ln \frac{\chi_2}{\chi_1^\alpha}$. Define d_H and d_E as the delay when the processor always runs at s_H and s_E respectively, i.e., $d_H = \frac{\sigma}{s_H}$ and $d_E = \frac{\sigma}{s_E}$. The worst-case delay d is also constrained by

$$d_H \leq d \leq d_E. \qquad (28)$$

The proof is given in Appendix A.

5 Delay Analysis of SP Scheduling

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

Lemma 3. *No matter what scheduling (FIFO or SP) is used in a system under reactive speed scaling defined in (11), the service function $g(t)$ of the aggregated task will be 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

$$g(t) = \int_0^t s(\tau) d\tau. \qquad (29)$$

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 this lemma, we are able to obtain the worst-case delay under SP scheduling as shown in the following theorem [2]:

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

$$d_i = \sup_{I \geq 0}\{\inf\{\tau : \sum\nolimits_{j=1}^{i-1} F_j(I+\tau) + F_i(I) \leq G(I+\tau)\}\}, \qquad (30)$$

where $G(I)$ is defined in (17) 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 ($j \leq i$) is running, and immediately before which no jobs from Tasks Γ_j ($j \leq i$) are running. We know that the delay 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 delay for a job of Task Γ_i as follows:

$$d_i = \sup_{t \geq t_1}\{\inf\{\tau : \sum\nolimits_{j=1}^{i-1} f_j(t+\tau) + f_i(t) \leq \sum\nolimits_{j=1}^{i} g_j(t+\tau)\}\}, \qquad (31)$$

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 delay happens when $\sum_{j=1}^{i-1}(f_j(t+\tau) - f_j(t_1)) + (f_i(t) - f_i(t_1)) = \sum_{j=1}^{i-1} F_j(I+\tau) + F_i(I)$. Define $I = t - t_1$ and then (30) holds. In (30), $G(I)$ is defined in (17). 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. ■

[2]In the following, the smaller index of a task indicates a higher priority.

Similarly, in the following we consider the leaky-bucket task workload as a case study. We have the following theorem on the worst-case delay 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 $F_i(I) = \sigma_i + \rho_i I$. The worst-case delay d_i for Task i can be written as*

$$d_i = \max\{d_{E,i} - \Delta, d_{H,i}\}, \qquad (32)$$

where

$$d_{E,i} = \frac{\sum_{j=1}^{i}\sigma_j}{s_E - \sum_{j=1}^{i-1}\rho_j}, \qquad (33)$$

$$d_{H,i} = \frac{\sum_{j=1}^{i}\sigma_j}{s_H - \sum_{j=1}^{i-1}\rho_j}, \qquad (34)$$

$$\Delta = \frac{\sigma - s_E d}{s_E - \sum_{j=1}^{i-1}\rho_j}. \qquad (35)$$

and d in (35) can be obtained by Corollary 1.

The proof is given in Appendix B.

6 Performance Evaluation

In this section we evaluate the benefit of using simple reactive speed scaling scheme by comparing the worst-case delay 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 [4] for a silicon chip. The thermal conductivity of the silicon material per unit volume is $k_{th} = 100\ W/mK$ and the thermal capacitance per unit volume is $c_{th} = 1.75 \times 10^6\ J/m^3K$. The chip is $t_{th} = 0.55\ mm$ thick. Therefore, the thermal RC time constant $RC = \frac{c_{th}}{k_{th}} t_{th}^2 = 0.0044\ sec$ [4]. Hence by Equation (4) $b \approx 228.6\ sec^{-1}$. The ambient temperature is 45°C and the maximum temperature threshold is 85°C, then $T_H = 40$°C. The equilibrium speed s_E will be fixed by the system, but s_H can be freely chosen. We arbitrarily pick $s_H = \frac{10}{7}s_E$ and assume $\alpha = 3$.

We consider three tasks Γ_i's ($i = 1, 2, 3$). As a case study, we consider a leaky-bucket workload and assume 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 delay of jobs in the system under reactive speed scaling and the baseline one in the systems the processor always run at the equilibrium speed.

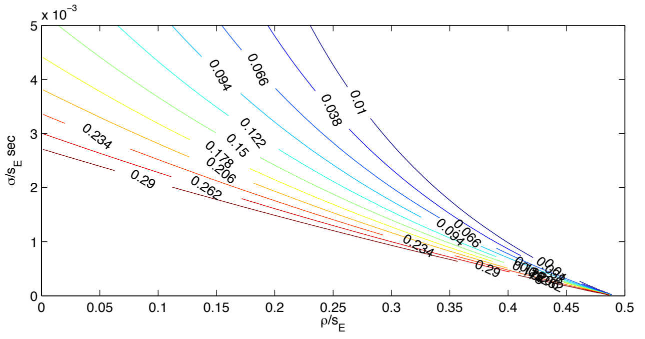

Figure 6. A contour plot of delay decrease ratio $|d - d_E|/d_E$ for the aggregated task under reactive speed scaling for FIFO scheduling.

First we consider FIFO scheduling. We evaluate the worst-case delay decrease ratio $|d - d_E|/d_E$ for the aggregated task. [3] Figure 6 shows a contour plot of $|d - d_E|/d_E$ in terms of σ/s_E and ρ/s_E. We observe that the delay 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 delay 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 delay decrease ratio $|d_i - d_{E,i}|/d_{E,i}$ for Task Γ_i. Each individual picture in Figure 7 shows contour plots of $|d_i - d_{E,i}|/d_{E,i}$ in terms of σ/s_E and ρ/s_E, for the three tasks separately. We observe that the delay 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 if the delay decrease ratio is not larger than 0.3, the ratio will decrease as either σ or ρ increases for any task. As if it becomes larger than 0.3, we have different observation results for the lower-priority tasks. In particular, considering the lower-priority task, for small σ and ρ, the delay decrease ratio can be written as $(1 - \frac{s_E}{s_H})/(1 - \frac{1}{s_H}\sum_{j=1}^{i}\rho_j)$. Therefore, as shown at the left-bottom corner of the last two contour plots in Figure 7, the delay decrease ratio will keep constant as σ increases and ρ keeps constant, but increase as ρ increases and σ keeps constant.

7 Conclusion and Future Work

Delay analysis in systems with temperature-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 delay for tasks with arbitrary job arrivals for both FIFO and SP scheduling algorithms in a system with a very simple 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 delays.

In order to further improve the performance of speed scaling, one would have to find ways to partially isolate jobs

[3] 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 delay decrease ratio.

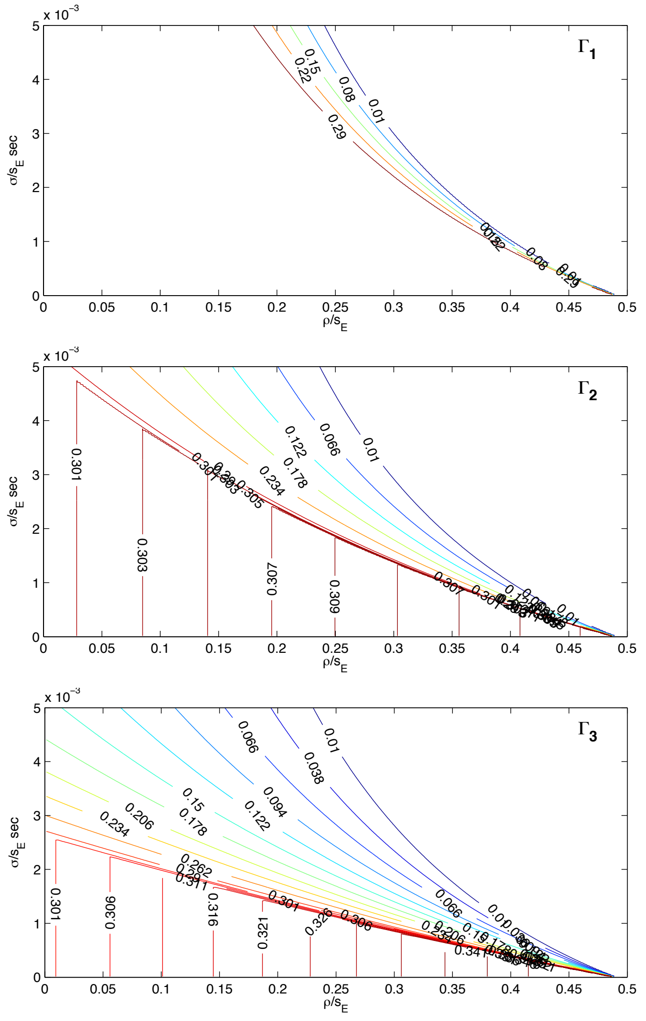

Figure 7. Contour plots of delay decrease ratio $|d_i - d_{E,i}|/d_{E,i}$ for Task Γ_i ($i = 1, 2, 3$) under reactive speed scaling for SP scheduling.

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. At this point we don't know how to perform delay analysis for non-trivial speed scaling algorithms, however.

References

[1] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 2001.

[2] Semiconductor Industry Association, "2005 international technology roadmap for semiconductors," `http://public.itrs.net`.

[3] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, "Analysis of thermal monitor features of the Intel Pentium M processor," in *Proceedings of the First Workshop on Temperature-Aware Computer Systems*, 2004.

[4] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-aware microarchitecture: Extended discussion and results," Tech. Rep. CS-2003-08, Department of Computer Science, University of Virginia, 2003.

[5] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to Intel Core Duo processor architecture," *Intel Technology Journal*, vol. 10, no. 2, pp. 89 – 97, 2006.

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

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

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

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

[10] S. Wang and R. Bettati, "Reactive speed control in temperature-constrained real-time systems," in *Euromicro Conference on Real-Time Systems*, 2006.

[11] A. Dhodapkar, C.H. Lim, G. Cai, and W.R. Daasch, "TEMPEST: A thermal enabled multi-model power/performance estimator," in *Workshop on Power-Aware Computer Systems, ASPLOS-IX*, 2000.

[12] A. Cohen, L. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy, "On estimating optimal performance of CPU dynamic thermal management," in *Computer Architecture Letters*, 2003.

[13] S. Wang and R. Bettati, "Delay analysis in temperature-constrained hard real-time systems with general task arrivals," Tech. Rep. tamu-cs-tr-2006-5-3, Department of Computer Science, Texas A&M University, 2006, `http://www.cs.tamu.edu/academics/tr/tamu-cs-tr-2006-5-3`.

A Proof of Corollary 1

We follow the analysis in Section 4, we consider the three constraints as follows:

Delay Constraint Since $F(I) = \sigma + \rho I$, by (19) we can obtain the delay d as

$$d = \max\{\frac{\sigma}{s_H}, \frac{\sigma}{s_E} - (\frac{s_H}{s_E} - 1)\delta_{1,h}\}. \tag{36}$$

Therefore, we have

$$\delta_{1,h} = \frac{\frac{\sigma}{s_E} - d}{\frac{s_H}{s_E} - 1}, \tag{37}$$

as

$$d \geq \frac{\sigma}{s_H}. \tag{38}$$

Service Constraint 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 (21) we have

$$s_H\delta_{1,h} + s_E\delta_{1,e} = \sigma + \rho(\delta_{1,h} + \delta_{1,e}). \tag{39}$$

Hence,

$$(s_H - \rho)\delta_{1,h} + (s_E - \rho)\delta_{1,e} = \sigma. \tag{40}$$

As $k = 2, \ldots, m$, by (22), we have

$$\begin{aligned} s_H \sum\nolimits_{j=1}^{k} \delta_{j,h} + s_E\delta_{1,e} \\ = \sigma + \rho((k-2)\delta + (\delta_2 + \delta_1) + d). \end{aligned} \tag{41}$$

If $k = 2$, together with (40), we have

$$\delta_{2,h} = \frac{\delta_{2,0} + d}{\frac{s_H}{\rho} - 1}. \quad (42)$$

If $k \geq 3$, we have

$$\delta_{k,h} = \frac{\rho}{s_H}\delta. \quad (43)$$

Temperature Constraint By (37) and (43), we can rewrite the temperature constraint condition (24).

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

$$\frac{T_k}{T_H} = e^{-b(m-k)\delta}(1-\xi) + \xi, \quad (44)$$

where

$$\xi = (\frac{s_H}{s_E})^\alpha \frac{1 - e^{-b\frac{\rho}{s_H}\delta}}{1 - e^{-b\delta}}. \quad (45)$$

By (44), T_2 is a function of δ and m. It is easy to know that the smaller T_2 is, the short delay d is. Therefore, we want to find δ and m to minimize T_2 so that we a tight upper-bound d of the original worst-case delay.

If $\xi \leq 1$, then $T_k/T_H \leq 1$. By (44), T_2 is a decreasing function in terms of $(m-2)\delta$, then $T_2/T_H \geq \lim_{(m-2)\delta\to\infty} T_2/T_H = \xi$. [4] 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 (44), 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

$$\frac{T_2}{T_H} = \min\{(\frac{s_H}{s_E})^\alpha\frac{\rho}{s_H}, 1\}. \quad (46)$$

At the same time, by (23) and (25), we have

$$\delta_{1,h} + \delta_{2,h} = \frac{1}{b}\ln\frac{(\frac{s_H}{s_E})^\alpha - \frac{T_2}{T_H}e^{-b\delta_{2,0}}}{(\frac{s_H}{s_E})^\alpha - 1}. \quad (47)$$

Therefore, by (37), (42), (46), and (47), we can obtain the worst-case delay d as follows:

$$\begin{aligned} d &= \frac{(1-\chi_1)(1-\chi_2)}{\chi_1 - \chi_2}(\frac{\chi_1}{1-\chi_1}\frac{\sigma}{s_E} + \frac{\chi_2}{1-\chi_2}\delta_{2,0} \\ &\quad -\frac{1}{b}\ln\frac{1 - \min\{\chi_2, \chi_1^\alpha\}e^{-b\delta_{2,0}}}{1-\chi_1^\alpha}), \quad (48)\end{aligned}$$

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

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

Equation (48) 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
$$d = V(X - Y), \quad (49)$$
- $\chi_2 > \chi_1^\alpha$: d will be minimized at $\delta_{2,0} = \frac{1}{b}\ln\frac{\chi_2}{\chi_1^\alpha}$, therefore
$$d = V(X - Y - Z), \quad (50)$$

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

On the other hand, in the temperature constraint, as $k = 1$, by (25) and the constraint that $T_1/T_H \leq 1$, we have

$$\delta_{1,h} \geq 0. \quad (51)$$

Therefore, by (37), we have

$$d \leq \frac{\sigma}{s_E}. \quad (52)$$

Recall that $d_H = \frac{\sigma}{s_H}$ and $d_E = \frac{\sigma}{s_E}$, then by (38) and (52), the worst-case delay is also constrained by

$$d_H \leq d \leq d_E. \quad (53)$$

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 (37) 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 (30), the delay formula can written as

$$\begin{aligned}&\sum_{j=1}^{i-1}\sigma_j + \rho_j(I + d_i) + \sigma_i + \rho_i I = \\ &\quad \min\{(s_H - s_E)\delta_{1,h} + s_E(I + d_i), \\ &\quad s_H(I + d_i)\} \quad (54)\end{aligned}$$

Then, as $d_i > \delta_{1,h}$, we have

$$d_i = \frac{\sum_{j=1}^{i-1}(\sigma_j + \rho_j d_i) + \sigma_i}{s_E} - (\frac{s_H}{s_E} - 1)\delta_{1,h}, \quad (55)$$

otherwise

$$d_i = \frac{\sum_{j=1}^{i-1}(\sigma_j + \rho_j d_i) + \sigma_i}{s_H}. \quad (56)$$

Therefore, by (37), (55), and (56), we have

$$d_i = \max\{d_{E,i} - \Delta, d_{H,i}\}, \quad (57)$$

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

$$d_{H,i} \leq d_i \leq d_{E,i}. \quad (58)$$