Minimizing the Maximum Processor Temperature by Temperature Aware Scheduling of Real-Time Tasks

Baver Ozceylan, Boudewijn R. Haverkort, Maurits de Graaf, and Marco E. T. Gerards

***Abstract*—Thermal management is gaining importance since it is a promising method for increasing the reliability and the lifespan of mobile devices. Although the temperature can be decreased by reducing processor speed, one must take care not to increase the processing times too much; violations of deadline constraints must be prevented. This article focuses on the tradeoff between performance and device temperature. We first analyze this tradeoff and show how to determine the optimal lower bound for the maximum temperature for a given set of jobs with known workloads and deadlines. To do so, we use a thermal model, which describes how future decisions impact temperature dynamics. Then, we introduce a processor scheduling algorithm that computes the resource allocation that achieves this lower bound. Consequently, our algorithm finds the optimal resource allocation for the purpose of minimizing the maximum processor temperature for a set of jobs with known workloads and deadlines. Our experimental validation shows that our thermal management algorithm can achieve a reduction of up to 15^oC (42%) of the maximum temperature when the workload is high, where a previously proposed method achieved a reduction of up to 10^oC (25%). Another advantage of our method is that it decreases the variance in the temperature profile by 16%, compared to previously proposed methods.**

***Index Terms*—Thermal management, resource allocation, leakage current, model predictive control, processor scheduling, reliability.**

I. Introduction

THE applications that are used in electronic devices have become more complex over the last decade, thus, requiring a significant amount of computational power. With the advances in electronic chip manufacturing technologies, mobile devices are able to keep up with this trend using multiprocessor System-on-Chips (SoCs), however, this has impact on high temperature related issues. Although the impact is partially alleviated by the increased efficiency of advance chip technologies, the overall heat dissipation increases due to the increase in computations [1]. High temperature values have catastrophic consequences for electronic devices. In [2], the authors emphasize that even small differences in the operating temperature can have a high impact on the lifespan of devices and underline the fact that high temperatures can also decrease the effective operating speed. Moreover, high variations in the temperature profile further decrease the reliability [3]. When dealing with high temperatures, the battery dependent nature and deployment environments of mobile devices should be taken into consideration. For example, active cooling mechanisms, such as using a fan or water cooling, consumes too much power for mobile devices due to their battery dependent nature, these are typically not possible in some environments, such as highly mobile environments, where equipment is subject to vibrations and shocks. Therefore, we focus on passive cooling mechanisms.

Passive cooling mechanisms are mostly based on throttling (slowing down) the processor so that they adjust the resources assigned to a task [4]. A commonly used mechanism is idle time scheduling, which periodically idles the processor to cool it. Another well known method is dynamic voltage and frequency scaling (DVFS), which controls the operating frequency and voltage [5]. Although these mechanisms are able to decrease the temperature effectively, they do decrease the performance as well. Therefore, every thermal management technique has to balance temperature vs. performance reduction. If the application requires a high performance, the adapted technique has to allow high temperature values to provide the required performance. However, if the application does not require a high performance, it should throttle the device. Most employed approaches assign a temperature limit and throttle the processor only if the temperature exceeds this limit, which particularly favors performance. In the Linux kernel [6], the default thermal policy, the so-called *performance* policy, implements this approach. The goal here is to prevent damage caused by high temperatures. However, this policy assigns a predetermined maximum temperature limit, which is based on the specifications of the hardware. On the other hand, there are some approaches [7]–[9] that throttle the processor *before* the temperature reaches the predetermined limit and prolong the duration until the temperature reaches the predetermined limit. To do so, they use algorithms that provide the minimum required performance during the full available processing time. For example, for a single job, these approaches assign the resources such that the job is completed precisely at the deadline, and not before. We refer to this as the *just enough* policy, as in [7]. The goal of this policy is to balance energy consumption and performance reduction. Another advantage is to reduce the exposure to high temperature values compared to the *performance* policy. As a result, the *performance* and *just enough* policies regulate the device performance while keeping the device temperature under a predetermined temperature limit, which means that they do not necessarily minimize the maximum temperature.

B. Ozceylan and M. E. T. Gerards are with the Department of the Computer Science and the Electrical Engineering, University of Twente, 7522 NB Enschede, the Netherlands (e-mail: b.ozceylan@utwente.nl; m.e.t.gerards@utwente.nl).

B. Haverkort is with the school of Humanities and Digital Sciences, Tilburg University, 5037 AB Tilburg, the Netherlands (e-mail: b.r.h.m.haverkort@tilburguniversity.edu).

M. de Graaf is with Thales Nederland B.V., 1271 ZA Huizen, the Netherlands (e-mail: maurits.degraaf@nl.thalesgroup.com).

Our aim in this article is to minimize the maximum temperature by assigning dynamic temperature limits while meeting the performance constraints. We focus on a use case with a given set of real-time jobs, which means that the workloads and deadlines are known. To achieve this, we first determine the optimal lower bound for the maximum temperature limit. Such a problem requires the use of a thermal model, as in [3], [10]–[14], to define it as a convex optimization problem. We use the thermal model from our previous work [12] for this purpose. After solving this problem analytically, we prove that this lower bound is optimal for a given set of jobs with known workloads and deadline, which means the best that the device can achieve with any thermal/power management approach based on throttling. Then, we propose a dynamic thermal management algorithm that allocates the resources and adjusts the temperature limit with respect to this lower bound. Furthermore, we observe that a side effect of our algorithm is a reduction in the variance of the temperature profile, which further improves reliability.

Consequently, our algorithm minimizes the maximum temperature for a given set of real-time jobs. Our contributions in this article are as follows:

- An analytic method that calculates the optimal lower bound for the maximum temperature limit at a given time with respect to known workloads and deadlines.
- An algorithmic approach that minimizes the maximum temperature for a set of real-time jobs by adjusting the allocated resources and the temperature limit dynamically.
- A mathematical proof that this algorithm finds the theoretical minimum value for the maximum temperature limit.
- Experimental evaluation on an Exynos 5422 MPSoC, which is a widely used commercial processor.

The remainder of this article is organized as follows. In Section II, we describe related work on thermal modeling and thermal management policies. In Section III, we briefly explain the thermal model that we use. Section IV defines and formalizes the problem. Then, in Section V, we start with the case with only one job and find an optimal solution. In Section VI, we generalize this solution to the multiple job case. In Section VII, we describe our implementation and present the experimental results. Section VIII concludes the article.

II. Related Work

The collected data from built-in temperature sensors in chips have drawn significant attention in recent research. For example, [15] proposes a method to detect malicious activities using built-in temperature sensors. On the other hand, there is a variety of proposed methods that increase the accuracy of built-in temperature sensors. While [16] uses convolution neural networks, [17] uses approximate computing to better increase the sensor accuracy. Their common motivation is to predict the temperature with the collected data using only built-in temperature sensors.

In the literature, several ways to predict the temperature have been proposed. Whereas initial work uses autoregressive moving averages (ARMA) to estimate future temperatures [10], more recent studies [3], [11]–[14] combine this statistical method with an analytical temperature model. In [3], the authors propose a thermal model and describe an experimental setup to determine the model parameters. This experimental setup requires a furnace and periodic measurements of the temperature and power using on-board sensors. The authors of [11] introduce a similar concept, however, without requiring power measurements. In order to deal with different environments, [13] uses a look-up table that contains error correction coefficients for different system states. This look-up table is updated on the fly based on the error between temperatures measured and estimated values to adapt the temperature to environmental changes. We introduced a thermal model in [12], and a method to estimate the system parameters. This method only uses built-in temperature sensors and is easy to apply, since it does not require any additional hardware, such as a furnace or power sensor. Moreover, in [14], we introduced two different extensions to this approach using a Kalman filter and a particle filter, such that the model can adapt to environmental changes. Thus, these methods are well studied to capture the dynamic thermal behavior of a system.

Although our goal in this article is to minimize the maximum temperature using the captured thermal behavior, various thermal management techniques have been proposed in the literature with different goals. In [11], a PID controller is used to reduce the temperature variance for mobile gaming applications. In [5] and its extended version [3], the authors propose an algorithm to calculate and distribute a power budget to control thermal violations. In [13], the goal is to determine the maximum operating frequency and processor utilization setting to avoid temperature violations before it reaches the assigned temperature limit. In [18], the authors focus on the temperature variance. They first introduce this as a convex optimization problem and then solve it to reduce the temperature variance. In [7] and [8], the authors propose a closed-loop control policy to achieve sustainable QoS (Quality of Service). They control the temperature using real-time QoS measurements. In [9], this policy is extended by adding application awareness.

There are several papers that address minimizing the maximum processor temperature as in this article. In [19], [20] and [21], the goal is to minimize the steady state temperatures while executing periodic tasks. The method in [20] first splits tasks and, then, schedules idle time, whereas, in [21] and [19], the authors use DVFS. All three studies use simulations to validate their methods. In [22], the task model is a set of real-time jobs. However, the authors mostly focus on leakage current, they only consider jobs with a common deadline and they validate their method using simulations.

In this article, we also address minimizing the maximum processor temperature and the task model that we use is a set of real-time jobs with different deadlines. We first introduce this as a convex optimization problem as in [18]. However, we solve it to minimize the maximum processor temperature. Our solution is a hybrid method that combines the *just enough* policy, which is proposed to achieve sustainable QoS in [7], and the *performance* policy, which is the default policy in Linux kernel.

III. Thermal Model

The heat dissipation in the system is directly related to assigned resources. There are two heat sources in a processor unit [23]. The first source, $Q_{\mathrm{D}}(t)$, reflects the switching activities in the processor, which depends on the active resources of the processor and it can be expressed as $Q_{\mathrm{D}}(t) = \alpha x(t)$, where $x(t)$ is the normalized amount of active resources, i.e., $0 \leq x(t) \leq 1$, and $\alpha > 0$ is a system dependent parameter. The second source, $Q_{\mathrm{L}}(t)$, relates to the leakage current in the processor, which depends on the temperature of the system and is given by $Q_{\mathrm{L}}(t) = C_1 T(t)^2 e^{\frac{C_2}{T(t)}}$ [23], where C_1 and C_2 are system dependent parameters and $T(t)$ is the processor temperature at time t. In our previous study [12], we derived a method to estimate the dynamic temperature behavior of CPUs and our experiments showed that the effect of $Q_{\mathrm{L}}(t)$ is very low compared to $Q_{\mathrm{D}}(t)$ when the temperature is not high and we can control $Q_{\mathrm{D}}(t)$ via adjusting the assigned resources. Since our aim is to operate in the low temperature range by adjusting the assigned resources, we focus on $Q_{\mathrm{D}}(t)$, the dominant contributor to the heat dissipation[1]. As a result, in this study, we use the following dynamic heat transfer equation:

$$\dot{T}(t) = -\frac{1}{\tau} T(t) + \frac{1}{\tau} T_{\mathrm{a}} + \frac{\alpha}{\tau} x(t), \tag{1}$$

where $\dot{T}(t)$ is the first order derivative of $T(t)$, $\tau > 0$ is the thermal system dependent time constant and T_{a} is the *ambient temperature*, which mainly depends on environmental factors. We assume T_{a} constant over time in this study. To ease the notation, we define the *output* as $y(t) = \frac{1}{\alpha} T(t) - \frac{1}{\alpha} T_{\mathrm{a}}$, which is the normalized temperature difference. With this, we can rewrite (1) as follows:

$$\dot{y}(t) = -\frac{1}{\tau} y(t) + \frac{1}{\tau} x(t), \tag{2}$$

and the solution to this first order differential equation is:

$$y(t) = \frac{1}{\tau} \int_{t_0}^{t} e^{\frac{-(t-\sigma)}{\tau}} x(\sigma) d\sigma + y(t_0) e^{\frac{-(t-t_0)}{\tau}}, \ \forall t \geq t_0. \tag{3}$$

In this study, we take the initial value $y(t_0)$ between 0 and 1. Thus, the expression in (3) always results in $0 \leq y(t) \leq 1$ since $x(t)$ is normalized. This means that the system temperature $T(t)$ always remains between T_{a} and $T_{\mathrm{a}} + \alpha$. Now considering a time interval $[\delta_a, \delta_b]$ where $0 \leq \delta_a < \delta_b$, we make two remarks based on (2) and (3). First, if $x(t)$ is a constant in this time interval as $x(t) = x(\delta_a)$, $\forall t \in [\delta_a, \delta_b]$, we can write $y(t)$ using (3) as:

$$y(t) = x(\delta_a)(1 - e^{\frac{-(t-\delta_a)}{\tau}}) + y(\delta_a) e^{\frac{-(t-\delta_a)}{\tau}}, \ \forall t \in [\delta_a, \delta_b]. \tag{4}$$

Second, if $y(t)$ is a constant in this time interval as $y(t) = y(\delta_a)$, $\forall t \in [\delta_a, \delta_b]$, we can write $x(t)$ using (2) as:

$$x(t) = y(\delta_a), \ \forall t \in [\delta_a, \delta_b], \tag{5}$$

As a result, we use (4) to predict the temperature change in time intervals where $x(t)$ is constant and (5) to adjust $x(t)$ in time intervals where a constant temperature value is desired.

[1]We show high temperature results in Section VII and revisit the factor $Q_{\mathrm{L}}(t)$ to explain the measurement results

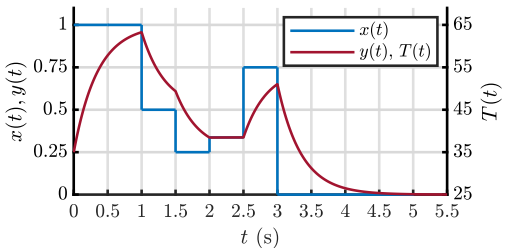


Fig. 1. Example of the dynamic thermal behavior of the system with varying input.

The example below illustrates the relation between $x(t)$, $T(t)$ and $y(t)$.

Example III.1. Figure 1 shows a sample input $x(t)$, which refers to the assigned resources, the corresponding output $y(t)$ and the temperature $T(t)$, which follows from (3) with parameters $\tau = 350ms$, $\alpha = 40^o C$, $T_{\mathrm{a}} = 25^o C$ and the initial value, $T_0 = 35^o C$.

IV. System Definition

Our target system is a processor running a set of real-time jobs, which means that the deadlines and workloads are known. Our goal is to find a resource allocation that minimizes the maximum processor temperature while considering system constraints. We model this system as a linear time-invariant system with an input $x(t)$ and output $y(t) = \frac{1}{\alpha}(T(t) - T_{\mathrm{a}})$ and $t \geq 0$. The initial temperature of the system at $t = 0$ is T_0. Since T_{a} is constant and α is positive, minimizing $\max T(t)$, our main objective, is the same as minimizing $\max y(t)$.

For each job $n \in \{1, 2, \ldots, N\}$ in a given set of N real-time jobs, where every job becomes available at $t = 0$, we denote the workload and deadline by p_n and d_n respectively, where p_n is the time that is needed to process the job at full system resources (i.e., $x(t) = 1$) and d_n is the time before which the job has to be completed. To ease the notation, we assume jobs are ordered by increasing deadlines and processed in this order. We represent the minimum cumulative required resources to meet all the deadlines coming before a given time t with the function $F_p(t)$:

$$F_p(t) = \sum_{n \in \{n | d_n \leq t\}} p_n, \forall t \geq 0. \tag{6}$$

We also represent the cumulative resources assigned up to a given time t with the function $F_x(t)$ as:

$$F_x(t) = \int_0^t x(\sigma) d\sigma. \tag{7}$$

Thus, the deadline constraints can be represented by $F_x(t) \geq F_p(t)$, $\forall t \in [0, d_N]$. Since we cannot assign more resources if there is no remaining workload, $F_x(t) \leq F_p(d_N)$, $\forall t \in [0, d_N]$, and $x(t) = 0$ when $t > d_N$. This also means that $y(t)$ is decreasing for $t > d_N$. Moreover, $0 \leq x(t) \leq 1$ leads to $0 \leq F_x(t) \leq t$, $\forall t \in [0, d_N]$. Therefore, the feasibility conditions for the input are:

$$\min\{F_p(d_N), t\} \geq F_x(t) \geq F_p(t), \ \forall t \in [0, d_N]. \tag{8}$$

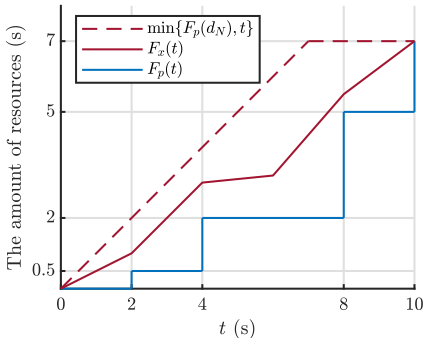


Fig. 2. Example of $F_p(t)$ and $F_x(t)$.

TABLE I
OVERVIEW OF THE USED NOTATION

$T(t)$	Processor temperature at time t
T_0	Initial temperature of the system
α, τ	System dependent parameters
T_a	Ambient temperature
$x(t)$	Normalized amount of active resources
$y(t)$	Normalized processor temperature at time t
$y(t)^c$	Complement of $y(t)$ as $y^c(t) = 1 - y(t)$
y_0	Normalized initial temperature of the system
p_n, d_n	Workload and deadline of the n^{th} job respectively
$F_p(t)$	Minimum cumulative required resources up to time t
$F_x(t)$	Cumulative resources assigned up to time t
$\lambda(t;d_n)$	Minimum cumulative required resources at time t up to d_n
$\lambda^c(t;d_n)$	Complement of $\lambda(t;d_n)$ as $\lambda^c(t;d_n) = d_n - t - \lambda(t;d_n)$
$y_{ss}(t;d_n)$	Stable state temperature corresponding to d_n and p_n
$\mu(t;d_n)$	Time point when the system reaches $y_{ss}(t;d_n)$

This also means that this system has a solution only if $F_p(t) \leq t$, $\forall t \in [0, d_N]$. From now on, we only focus on the time interval $[0, d_N]$ and assume that the above conditions are always satisfied. The example below illustrates the relation between $F_p(t)$, $F_x(t)$ and $\min\{F_p(d_N), t\}$.

Example IV.1. Figure 2 shows a sample set of real-time jobs in the form of $F_p(t)$ and a sample input $x(t)$ in the form $F_x(t)$. This set contains $N = 4$ jobs, where $p_1 = 0.5s$, $p_2 = 1.5s$, $p_3 = 3s$, $p_4 = 2s$, $d_1 = 2s$, $d_1 = 4s$, $d_1 = 8s$ and $d_4 = 10s$. Furthermore, the dashed line represents the upper boundary $\min\{F_p(d_N), t\}$ in (8).

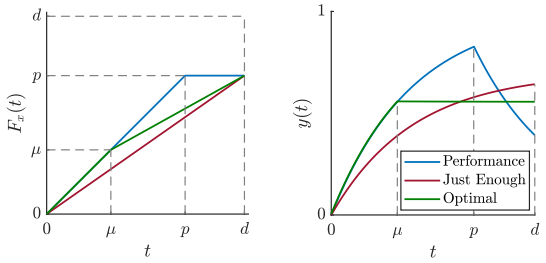
To ease the notation, at time t and for a given deadline d_n, where $t < d_n$, we define the remaining minimum required amount of resources as:

$$\lambda(t;d_n) = F_p(d_n) - F_x(t). \tag{9}$$

This gives the required amount of resources at time t to meet the deadline d_n. This function may have negative values, which implies that the corresponding deadline is already satisfied. Since $F_p(t) \leq t$, the maximum value for $\lambda(t;d_n)$ is $d_n - t$, which is the total available amount of resources. Hence, we define the complement of $\lambda(t;d_n)$ as:

$$\lambda^c(t;d_n) = d_n - t - \lambda(t;d_n). \tag{10}$$

Moreover, using $y(t) \in [0,1]$, we define the complement of the output as $y^c(t) = 1 - y(t)$. This means that $y^c(t)$ is also in the range of $[0,1]$.



(a) Input (b) Output

Fig. 3. Comparison of three different scheduling policies for the single job case with $y_0 = 0$.

In Section V, we consider the case with only one job and find an optimal solution. In Section VI, we generalize this solution to the multiple job case.

V. SINGLE JOB CASE

In this section, we consider a single real-time job that becomes available at $t = 0$ with a workload p and a deadline d hence:

$$F_p(t) = \begin{cases} p, & t \geq d, \\ 0, & t < d. \end{cases}$$

and the deadline constraint is $F_x(d) = p$. Therefore, given p, d and y_0, we aim to find an input $x(t)$ that minimizes $\max y(t)$, where $y(t)$ depends on $x(t)$ as in (3). We call this the *single job resource allocation problem* and its full description is:

$$\min_{x(t)} \max_{t \in [0,d]} y(t) \tag{11a}$$

$$\text{s.t. } y(t) = \frac{1}{\tau} \int_0^t e^{\frac{-(t-\sigma)}{\tau}} x(\sigma) d\sigma + y_0 e^{\frac{-t}{\tau}}, \tag{11b}$$

$$\int_0^d x(t) dt = p, \tag{11c}$$

$$0 \leq x(t) \leq 1 \ \forall t \in [0,d]. \tag{11d}$$

To develop an intuition, we compare the *performance* and *just enough* policies with the optimal solution. Figure 3a shows three different $F_x(t)$ samples, which all result in the same amount of workload (i.e., $F_x(d) = p$). The resulting outputs based on (11b) are shown in Figure 3b. The *performance* policy aims to complete the process as soon as possible, which means that $x(t) = 1$ $\forall t \in [0,p]$ and $x(t) = 0$ otherwise. Hence, $y(t)$ reaches its maximum value at $t = p$ and $\max y(t) = 1 + (y_0 - 1)e^{\frac{-p}{\tau}}$ due to (4). The *just enough* policy aims to provide sustainable and no more than required performance, which means that $x(t) = \frac{d}{p}$, $\forall t \in [0,d]$. Hence, $y(t)$ reaches its maximum value at $t = d$ and $\max y(t) = \frac{d}{p} + (y_0 - \frac{d}{p})e^{\frac{-d}{\tau}}$ due to (4). Although the *just enough* policy does not greedily use the resources, and consequently achieves a lower objective value $\max y(t)$ than the *performance* policy, it does not solve the resource allocation problem (11) optimally because the *just enough* policy does not utilize the resources effectively when the temperature is low. Therefore, our proposed solution (*optimal*) in Figure 3a behaves like the *performance* policy until a time point μ and then behaves like the *just enough* policy so that it

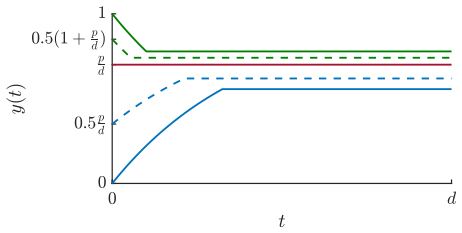


Fig. 4. Our approach to solve the single job resource allocation problem in (11) with different initial values.

minimizes the maximum temperature value as in Figure 3b. We introduce Algorithm 1 to find this critical time point μ and the corresponding input $x(t)$ and Theorem V.5 proves the optimality of this input.

At this point, we introduce the following lemma to provide an insight into the problem. In particular, it shows that $\int_0^d y(t)dt + \tau y(d)$ is constant for all possible inputs $x(t)$.

Lemma V.1. *Given τ, p, d and y_0, any $x(t)$ that follows the constraint (11c) leads to an output $y(t)$ with respect to (11b) such that:*

$$\int_0^d y(t)dt + \tau y(d) = \tau y_0 + p. \tag{12}$$

Proof. To prove this lemma, we first integrate both sides of the equality (2) from $t = 0$ to $t = d$:

$$y(d) - y_0 = -\frac{1}{\tau}\int_0^d y(t)dt + \frac{1}{\tau}\int_0^d x(t)dt.$$

Then, we rearrange this equality using (11c) to obtain (12). ■

We have developed our approach based on Lemma V.1. Intuitively, the naive approach would be to obtain a constant output $y(t)$ in the time interval $[0,d]$ since $\int_0^d y(t)dt + \tau y(d)$ is constant. This constant value is $\frac{\tau y_0 + p}{d+\tau}$ according to (12). However, constraints (11b) and (11d) only allow this solution when the initial value y_0 equals this constant output value. Thus, if the system meets this condition, the output $y(t)$ can be kept constant during the whole process as the red line in Figure 4 indicates. Otherwise, our approach is heating up (the blue line and blue dashed line in Figure 4) or cooling down (the green line and green dashed line in Figure 4) the system until the time point that the remaining workload, remaining time to the deadline and the output meet this condition. This implies that the policy behind our algorithm is keeping the temperature constant as long as possible as shown in Figure 4.

In order to allow for later generalization to a system with multiple jobs, we analyze the system with respect to a deadline d when the system is at time $t < d$, which means that we need to determine $x(t)$ for the time interval $[t,d]$. The remaining workload with respect to the deadline d is $\lambda(t;d)$ and the remaining time is $d-t$. We define three system states at time t as follows:

- The system is in a **heating state** when $y(t) < \dfrac{\lambda(t;d)}{d-t}$.
- The system is in a **stable state** when $y(t) = \dfrac{\lambda(t;d)}{d-t}$.
- The system is in a **cooling state** when $y(t) > \dfrac{\lambda(t;d)}{d-t}$.

For example, when $t = 0$, then the system is in a stable state when $y_0 = \frac{p}{d}$, in a heating state when $y_0 > \frac{p}{d}$ and in a cooling state when $y_0 < \frac{p}{d}$. At time t, if the system is in a stable state, the output $y(t)$ can be kept constant until this deadline d; if the system is in a heating state, our approach is to fully utilize ($x(t) = 1$) the system until it reaches a stable state; and if the system is in a cooling state, our approach is to idle ($x(t) = 0$) the system until it reaches a stable state. We denote the corresponding stable state temperature by $y_{ss}(t;d)$ and the time point when the system reaches this temperature value by $\mu(t;d)$. The main challenge of this approach is to determine $y_{ss}(t;d)$ and $\mu(t;d)$ with respect a deadline d when the system is at time t. For this purpose, Lemmas V.2 and V.3 use the *Lambert W function* [24], which is defined to solve $x = ze^z$ as $z = W(x)$. In this study, we only use the real values and principle branch of the W function, which is denoted by W_0. Hence, it has the following properties:

$$z = W_0(ze^z), \tag{13}$$

$$z = W_0(z)e^{W_0(z)}. \tag{14}$$

As a result, at any time t, we can calculate $y_{ss}(t;d)$ and $\mu(t;d)$ using (16), (15), (19) and (18) with respect to the remaining workload $\lambda(t;d)$, the remaining time to the deadline $d-t$ and the initial value $y(t)$.

Lemma V.2. *At time $t \in [0,d)$, if the system is in a heating state, it reaches a stable state after applying $x(t) = 1$ (fully utilized) for a time period $\mu(t;d) \in [0,d-t)$, where:*

$$\mu(t;d) = \frac{\lambda(t;d) - y_{ss}(t;d)(d-t)}{1 - y_{ss}(t;d)}. \tag{15}$$

Then, the stable state temperature $y_{ss}(t;d)$ is defined as:

$$y_{ss}(t;d) = 1 - \frac{\lambda^c(t;d)}{\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}\Big)}, \tag{16}$$

and it satisfies the following equality:

$$e^{\frac{-\mu(t;d)}{\tau}} = \frac{1 - y_{ss}(t;d)}{y^c(t)}. \tag{17}$$

Proof. We prove this lemma in Appendix A. ■

Lemma V.3. *At time $t \in [0,d)$, if the system is in a cooling state, it reaches a stable state after applying $x(t) = 0$ (idled) for a time period $\mu(t;d) \in [0,d-t)$, where:*

$$\mu(t;d) = \frac{y_{ss}(t;d)(d-t) - \lambda(t;d)}{y_{ss}(t;d)}. \tag{18}$$

Then, the stable state temperature $y_{ss}(t;d)$ is defined as:

$$y_{ss}(t;d) = \frac{\lambda(t;d)}{\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}\Big)}. \tag{19}$$

Proof. We prove this lemma in Appendix B. ■

The approach explained above is implemented in Algorithm 1 as the function $\texttt{SJ}\,(\lambda(t;d), d-t, y(t))$. We prove the optimality of this algorithm in Theorem V.5 using the upper bound in the following lemma:

Algorithm 1: The solution to the single job resource allocation problem (11)

Parameters : τ
Input: d, p, y_0
Result: $x(t)$
Function $\mathtt{SJ}(p,d,y_0)$ **:**
  if $y_0 \le \frac{p}{d}$ **then**
    $\mu \leftarrow d - \tau W_0\Big(\frac{e^{\frac{d}{\tau}}}{\tau}\frac{d-p}{1-y_0}\Big)$;
    $x(t) \leftarrow 1 \ \forall t \in [0,\mu)$;
    $p \leftarrow p - \mu$;
  else
    $\mu \leftarrow d - \tau W_0\Big(\frac{e^{\frac{d}{\tau}}}{\tau}\frac{p}{y_0}\Big)$;
    $x(t) \leftarrow 0 \ \forall t \in [0,\mu)$;
  end
  $x(t) \leftarrow \frac{p}{d-\mu} \ \forall t \in [\mu,d)$;
End Function

Lemma V.4. *At $t = 0$, for $\delta \in [0,d]$, the objective function (11a) of the resource allocation problem (11) has a lower bound in the time interval $[\delta,d]$ as:*

$$\max_{t\in[\delta,d]} y(t) \ge 1 - \frac{\lambda^c(0;d) + \tau y^c(0) e^{\frac{-\delta}{\tau}}}{d-\delta+\tau}. \tag{20}$$

Proof. We prove this lemma in Appendix C. ■

Theorem V.5. *Algorithm 1 results in an optimal solution to the single job resource allocation problem (11) with respect to the deadline d, workload p and initial value y_0.*

Proof. Given that, at $t = 0$, the workload is p, the deadline is d and the initial value $y(0)$ is y_0. This means that $\lambda(0;d) = p$, $\lambda^c(0;d) = d - p$ and $y^c(0) = 1 - y_0$. We can prove this theorem by showing that the maximum temperature value equals the lower bound (20). To prove this, we separately consider all three possible cases, which are $y_0 > \frac{p}{d}$ (cooling state), $y_0 = \frac{p}{d}$ (stable state) and $y_0 < \frac{p}{d}$ (heating state) in the time interval $[0,d]$:

Case 1. If $y_0 > \frac{p}{d}$ (cooling state), Algorithm 1 computes $\mu > 0$ according to Lemma V.3 and then idles the system, i.e., $x(t) = 0$, until it reaches a stable state. The output achieved by the algorithm is always less than y_0, which means $y(t) < y_0$, $\forall t(0,d]$ and $y(t) = y_0$ at $t = 0$. Therefore, the maximum value is y_0, which is the lower bound for the objective function. Therefore, if the system is in a cooling state, Algorithm 1 is optimal.

Case 2. If $y_0 = \frac{p}{d}$ (stable state), Algorithm 1 computes $\mu = 0$ due to (13) and then generates the input $x(t) = \frac{p}{d}$, $\forall t \in [0,d]$. This input keeps $y(t)$ constant in the time interval $[0,d]$ according to (5), which means $y(t) = y_0$, $\forall t[0,d]$. To see that this is optimal, consider the lower bound in Lemma V.4 in the time interval $[0,d]$ as:

$$\max_{t\in[0,d]} y(t) \ge 1 - \frac{\lambda^c(0;d) + \tau y^c(0)}{d+\tau} = \frac{p+\tau y_0}{d+\tau}.$$

After substituting $y_0 = \frac{p}{d}$, we have $\max_{t\in[0,d]} y(t) \ge y_0$, which means that y_0 is the lower bound for the objective function. Therefore, if the system is in a stable state, Algorithm 1 is optimal.

Case 3. If $y_0 < \frac{p}{d}$ (heating state), Algorithm 1 computes $\mu > 0$ according to Lemma V.2 and then fully utilizes the system, i.e., $x(t) = 1$, until it reaches a stable state. To see that this is optimal, consider the lower bound in Lemma V.4 in the time interval $[\mu,d]$ as:

$$\max_{t\in[\mu,d]} y(t) \ge 1 - \frac{\lambda^c(0;d) + \tau y^c(0) e^{\frac{-\mu}{\tau}}}{d-\mu+\tau}.$$

Due to (17), $y^c(0)e^{\frac{-\mu}{\tau}} = 1 - y_{\mathrm{ss}}(0;d)$. Then:

$$\max_{t\in[\mu,d]} y(t) \ge \frac{p - \mu + \tau y_{\mathrm{ss}}(0;d)}{d-\mu+\tau}.$$

Substituting (15):

$$\max_{t\in[\mu,d]} y(t) \ge \frac{p - \frac{p - d y_{\mathrm{ss}}(0;d)}{1-y_{\mathrm{ss}}(0;d)} + \tau y_{\mathrm{ss}}(0;d)}{d - \frac{p - d y_{\mathrm{ss}}(0;d)}{1-y_{\mathrm{ss}}(0;d)} + \tau} = y_{\mathrm{ss}}(0;d).$$

This means that $y_{\mathrm{ss}}(0;d)$ computed with respect to Lemma V.2 is the lower bound for the objective function. Therefore, if the system is in a heating state, Algorithm 1 is optimal. ■

We now make the following remarks regarding Algorithm 1. These help to generalize the single job case to the multiple job case in the next section. For the input obtained by $\mathtt{SJ}(\lambda(0;d),d,y_0)$ (using Algorithm 1) for the time interval $[0,d]$, we have:

Remark 1. $y_{\mathrm{ss}}(t;d) = y_{\mathrm{ss}}(0,d)$, $\forall t \in [0,d]$.

Remark 2. $\mu(t;d) = \max\{0, \mu(0;d) - t\}$, $\forall t \in [0,d]$.

Remark 3. if $y_0 > \frac{\lambda(0;d)}{d}$ (cooling state):

$$F_x(t) = \begin{cases} 0, & 0 < t \le \mu(0;d), \\ (t - \mu(0;d)) y_{\mathrm{ss}}(0,d), & \mu(0;d) < t \le d, \end{cases} \tag{21}$$

$$y(t) = \begin{cases} y_0 e^{\frac{-t}{\tau}}, & 0 < t \le \mu(0;d), \\ y_{\mathrm{ss}}(0,d), & \mu(0;d) < t \le d, \end{cases} \tag{22}$$

where $y_0 > y_{\mathrm{ss}}(0,d) = y_0 e^{\frac{-\mu(0;d)}{\tau}} > \frac{\lambda(0;d)}{d}$, and $y(t) > \frac{\lambda(t;d)}{d-t}$, $\forall t \in [0,\mu(0;d))$, $y(t) = \frac{\lambda(t;d)}{d-t}$, $\forall t \in [\mu(0;d),d)$.

Remark 4. If $y_0 = \frac{\lambda(0;d)}{d}$ (stable state):

$$F_x(t) = y_{\mathrm{ss}}(0,d)t, \tag{23}$$

$$y(t) = y_{\mathrm{ss}}(0,d), \tag{24}$$

where $y_0 = y_{\mathrm{ss}}(0,d) = \frac{\lambda(0;d)}{d}$, and $y(t) = \frac{\lambda(t;d)}{d-t}$, $\forall t \in [0,d)$.

Remark 5. If $y_0 < \frac{\lambda(0;d)}{d}$ (heating state):

$$F_x(t) = \begin{cases} t, & 0 < t \le \mu(0;d), \\ \mu(0;d) + (t - \mu(0;d)) y_{\mathrm{ss}}(0,d), & \mu(0;d) < t \le d, \end{cases} \tag{25}$$

$$y(t) = \begin{cases} 1 - (1-y_0)e^{\frac{-t}{\tau}}, & 0 < t \le \mu(0;d), \\ y_{\mathrm{ss}}(0,d), & \mu(0;d) < t \le d, \end{cases} \tag{26}$$

where $y_0 < y_{ss}(0,d) = 1-(1-y_0)e^{\frac{-\mu(0;d)}{\tau}} < \frac{\lambda(0;d)}{d}$, and $y(t) < \frac{\lambda(t;d)}{d-t}$, $\forall t \in [0,\mu(0;d))$, $y(t) = \frac{\lambda(t;d)}{d-t}$, $\forall t \in [\mu(0;d),d)$.

VI. Multiple Job Case

We now generalize the single job resource allocation problem (11) to N jobs that become available at $t=0$ with a vector of workloads $P=[p_1,p_2,\ldots,p_N]$ and a vector of deadlines $D=[d_1,d_2,\ldots,d_N]$, where $d_k \geq d_j$ if $k>j$ since the jobs are ordered by increasing deadlines. We formalize these deadline constraints as $F_x(d_k) \geq F_p(d_k)$, $\forall k \in \{1,2,\ldots,N\}$ and the constraint $F_x(d_N) = F_p(d_N)$ prevents the assignment of a different amount of resources than required similar to (11c). In the remainder, we focus on instances that have a feasible solution in the time interval $[0,d_N]$. This means that $F_p(t) \leq t$, $\forall t \in [0,d_N]$ and $x(t)=0$ when $t>d_N$. Therefore, given $F_p(t)$ and y_0, we aim to find an input $x(t)$ that minimizes $\max y(t)$, where $y(t)$ depends on $x(t)$ as in (3). We call this the *multiple job resource allocation problem* and its full description is:

$$\min_{x(t)} \max_{t\in[0,d_N]} y(t) \tag{27a}$$

$$\text{s.t. } y(t) = \frac{1}{\tau}\int_0^t e^{\frac{-(t-\sigma)}{\tau}} x(\sigma)d\sigma + y_0 e^{\frac{-t}{\tau}}, \tag{27b}$$

$$F_x(d_k) \geq F_p(d_k),\ \forall k \in \{1,2,\ldots,N-1\}, \tag{27c}$$

$$F_x(d_N) = F_p(d_N), \tag{27d}$$

$$0 \leq x(t) \leq 1\ \forall t \in [0,d_N]. \tag{27e}$$

Although we can try to solve this problem using Algorithm 1 as $\texttt{SJ}(\lambda(0;d_N),d_N,y_0)$, which satisfies the constraints (27b), (27d) and (27e), this solution may miss some of the intermediate deadline constraints (27c). However, if we consider the total workload up to each deadline independently by ignoring (27c), Algorithm 1 as $\texttt{SJ}(\lambda(0;d_n),d_n,y_0)$ gives a lower limit for the time interval $[0,d_n]$, which is $y_{ss}(0,d_n)$. This means that no input exists for the time interval $[0,d_n]$ that both satisfies the deadline d_n and keeps the temperature below $y_{ss}(0,d_n)$. Although each deadline results in a different lower limit, we can focus on only the maximum of these lower limits, which means $\max\{y_{ss}(0,d_n)\}$, $\forall n \in \{1,\ldots,N\}$, since the maximum temperature cannot be lower than any of them. This highest limit creates a thermal bottleneck, which is not present in the unconstrained case (solved by Theorem V.5). As a result, this bottleneck leads to a lower limit that the temperature cannot stay under while satisfying all the deadline constraints. This means that if $\max y(t) = \max\{y_{ss}(0,d_n)\}$, $\forall n \in \{1,\ldots,N\}$, the corresponding input is optimal. Our approach is based on the intuation that we can find such an optimal input, which means that the optimal maximum temperature depends on the thermal bottleneck. We later prove this intuation analytically.

To obtain such an optimal input, we first divide the entire time interval $[0,d_N]$ into two subintervals in such a way that the first subinterval ends at the deadline that creates a thermal bottleneck. We choose the division point d_v from the vector of deadlines using the following rule:

$$d_v = \underset{n\in\{1,\ldots,N\}}{\arg\max}\ \{y_{ss}(0,d_n)\}, \tag{28}$$

Here, we assume that all the $y_{ss}(0,d_n)$ values are different without loss of generality. Thus, the first subinterval is $[0,d_v]$ and the second is $[d_v,d_N]$, where $v=\{1,2,\ldots,N\}$. During the first subinterval, we aim to process only the jobs that have their deadlines in this subinterval, i.e., if $n \leq v$ the n^{th} job is processed during the subinterval $[0,d_v]$ otherwise during the subinterval $[d_v,d_N]$. This also implies that the arrival time of the n^{th} job is allowed to be updated to d_v if $n>v$ without loosing optimality. Therefore, at $t=0$, we compute the input $x(t)$ that corresponds to the first subinterval as a single job problem with the workload $\lambda(0;d_v)=F_p(d_v)$, the deadline d_v and the initial value y_0, i.e., $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$, and then we use this input up to the division point d_v.

For notational convenience, we now define $F_x^{(d_n)}(t)$ for a deadline d_n, which denotes the cumulative assigned resources resulted from the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_n),d_n,y_0)$ for the time interval $[0,d_n]$ and we define $\lambda^{(d_n)}(t;d_k)$, $\forall t \in [0,d_n]$, which denotes the remaining minimum required amount of resources to be allocated from t to d_k when the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_n),d_n,y_0)$ is applied on the time interval $[0,d_n]$. This means that:

$$\lambda^{(d_n)}(t;d_k) = F_p(d_k) - F_x^{(d_n)}(t). \tag{29}$$

Moreover, $F_x^{(d_n)}(d_n) = \lambda(0;d_n) = F_p(d_n)$ since Algorithm 1 satisfies the corresponding deadline. We also define $y^{(d_n)}(t)$, which denotes the output that corresponds to $F_x^{(d_n)}(t)$. The following lemma uses this notation to show that the input obtained using Algorithm 1 with respect to the deadline d_v given by (28) always allocates more resources then the one with respect to any other deadline.

Lemma VI.1. *Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28)and each deadline d_k where $k \in \{1,\ldots,N\}$:*

$$F_x^{(d_v)}(t) \geq F_x^{(d_k)}(t),\ \forall t \in [0,\min\{d_v,d_k\}]. \tag{30}$$

Proof. We prove this lemma in Appendix D. ■

To prove that the rule (28) leads to optimality, we need to consider two issues. The first one is that the obtained solution by Algorithm 1 for the subinterval $[0,d_v]$ may not satisfy the constraints (27c) since we treat a set of jobs as a single job. The second issue is that these two subintervals cannot be considered as independent since the final temperature of the first one is the initial temperature of the second one.

To cope with the issues mentioned above, we introduce Lemmas VI.2 and VI.3. Lemma VI.2 proves that the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ satisfies all the deadline constraints up to this division point, i.e., $F_x(d_k) \geq \lambda(0;d_v) = F_p(d_k)$, $\forall k \in \{1,2,\ldots,v\}$. Next, Lemma VI.3 proves that, after applying this input up to $t=d_v$, there is no deadline where the corresponding state is a heating state. Theorem VI.4 uses Lemma VI.2 to prove that the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ for the time interval $[0,d_v]$ is optimal and it uses Lemma VI.3 to prove that there exist a feasible solution to the time interval $[d_v,d_N]$ such that the output does not increase after applying this input. In other words, Theorem VI.4 proves that the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ optimally solves the multiple job resource allocation problem (27).

Lemma VI.2. *Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28), the input obtained by* $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ *satisfies the constraints (27c) on the time interval* $[0,d_v]$.

Proof. According to Lemma VI.1, for each d_k where $k \in \{1,2,\ldots,v\}$, $F_x^{(d_v)}(t) \geq F_x^{(d_k)}(t)$, $\forall t \in [0,d_k]$. Since each $F_x^{(d_k)}(t)$ is obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$, each of them satisfies the corresponding deadline constraint, i.e., $F_x^{(d_k)}(d_k) = \lambda(0;d_k) = F_p(d_k)$. Hence, $F_x^{(d_v)}(d_k) \geq F_p(d_k)$, $\forall k \in \{1,2,\ldots,v\}$, which proves that the input obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ satisfies the constraints (27c) on the time interval $[0,d_v]$. ■

Lemma VI.3. *Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28), the input obtained by* $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ *leads to an output at $t = d_v$ such that the system is a cooling state for each d_k where $k \in \{1,2,\ldots,v\}$, i.e.:*

$$y^{(d_v)}(d_v) > \frac{\lambda^{(d_v)}(d_v;d_k)}{d_k - d_v}, \ \forall d_k \in (d_v,d_N]. \tag{31}$$

Proof. We prove this lemma in Appendix E. ■

Theorem VI.4. *Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28) and the input $x(t)$ for $t \in [0,d_v]$ obtained by* $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$*, there exists a remaining input $x(t)$ for $t \in [d_v,d_N]$, such that $x(t)$ results in an optimal solution to the multiple job resource allocation problem (27).*

Proof. Algorithm 1 is optimal to the single job resource allocation problem as proven in Theorem V.5. Therefore, for each deadline d_k, where $k \in \{1,2,\ldots,N\}$, the final temperature $y^{(d_k)}(d_k) = y_{\text{ss}}(0,d_k)$ or the initial temperature y_0 indicates a lower bound for the output $y(t)$ in the time interval $[0,d_k]$ as:

$$\max_{t\in[0,d_N]} y(t) \geq \max_{t\in[0,d_k]} y(t) \geq \max_{k\in\{1,2,\ldots,N\}} \{y_0, y_{\text{ss}}(0,d_k)\}.$$

Considering the deadline d_v given by (28), we can rewrite these inequalities as:

$$\max_{t\in[0,d_N]} y(t) \geq \max\{y_0, y_{\text{ss}}(0,d_v)\}. \tag{32}$$

The input obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$ leads to the output $y^{(d_v)}(t)$ hence (22), (24) or (26) is applicable for $y^{(d_v)}(t)$. Thus:

$$\max\{y_0, y_{\text{ss}}(0,d_v)\} \geq y^{(d_v)}(t), \forall t \in [0,d_v]. \tag{33}$$

Combining (32) and (33):

$$\max_{t\in[0,d_N]} y(t) \geq y^{(d_v)}(t), \forall t \in [0,d_v].$$

This means that there exist no input that results in a lower maximum temperature value than $y^{(d_v)}(t)$ in the time interval $[0,d_v]$. Additionally, Lemma VI.2 shows that $F_x^{(d_v)}(t)$ also meets all the deadlines coming before d_v. Therefore, for the time interval $[0,d_v]$, $F_x^{(d_v)}(t)$ satisfies all the deadline constraints and minimizes the maximum temperature.

For the time interval $[d_v,d_N]$, we here show that, after applying the input $x(t)$ obtained by $\texttt{SJ}(\lambda(0;d_v),d_v,y_0)$, there exist a feasible input that meets all the upcoming deadlines with keeping the temperature always below or equal to the temperature value at $t = d_v$, i.e., $y(t) \leq y_{\text{ss}}(0,d_v)$, $\forall t \in [d_v,d_N]$. Notice that $F_x^{(d_v)}(d_v) = \lambda(0;d_v) = F_p(d_v)$ and $y^{(d_v)}(d_v) = y_{\text{ss}}(0,d_v)$ after applying $F_x^{(d_v)}(t)$. An example input for this purpose is:

$$x^*(t) = \begin{cases} y^{(d_v)}(d_v), & d_v < t \leq d_v + \frac{\lambda^{(d_v)}(d_v;d_N)}{y^{(d_v)}(d_v)}, \\ 0, & d_v + \frac{\lambda^{(d_v)}(d_v;d_N)}{y^{(d_v)}(d_v)} \leq t, \end{cases}$$

where $y^{(d_v)}(d_v)(d_N - d_v) = \lambda^{(d_v)}(d_v;d_N)$. Due to (5), this input keeps the temperature stable after $t = d_v$ until $t = d_v + \frac{\lambda^{(d_v)}(d_v;d_N)}{y^{(d_v)}(d_v)}$. After that, the temperature decreases since it idles the system. Then, the cumulative assigned resources is $F_x^*(t) = F_x^{(d_v)}(d_v) + \int_{d_v}^t x^*(t)dt$ and can be computed as:

$$F_x^*(t) = \lambda(0;d_v) + \min\{y^{(d_v)}(d_v)(t-d_v), \lambda^{(d_v)}(d_v;d_N)\}.$$

For each deadline d_k in the time interval (d_v,d_N), $\lambda^{(d_v)}(d_v;d_N) > \lambda^{(d_v)}(d_v;d_k)$ and $y^{(d_v)}(d_v)(d_k - d_v) > \lambda^{(d_v)}(d_v;d_k)$ due to (31). Thus:

$$\begin{aligned} F_x^*(d_k) &= \lambda(0;d_v) + \min\{\lambda^{(d_v)}(d_v;d_N), y^{(d_v)}(d_v)(d_k - d_v)\} \\ &> \lambda(0;d_v) + \lambda^{(d_v)}(d_v;d_k) = \lambda(0;d_k) = F_p(d_k), \end{aligned}$$

which proves that $x^*(t)$ satisfies the deadline constraints (27c) for each d_k where $v < k < N$. Additionally, $F_x^*(d_N) = \lambda(0;d_N) = F_p(d_N)$, which proves that $x^*(t)$ satisfies the deadline constraint (27d).

As a result, Algorithm 1 as $\texttt{SJ}(F_p(d_v),d_v,y_0)$ results in an output that is optimal in the time interval $[0,d_v]$ and the remaining workload can be processed without increasing the temperature after $t = d_v$. ■

VII. Implementation

For evaluation purposes, we use the Odroid-XU4 platform, which hosts a widely used commercial processor Exynos 5422 MPSoC [3], [8], [9], [11], [13], [14]. For the implementation of our algorithm, we first calculate the division with respect to the rule (28) and then apply the input that we obtained by Algorithm 1 to the first subinterval. After reaching the division point, we repeat this approach iteratively, e.g., in an online fashion, for the remaining jobs. The overhead due to these computations are executed on the device during the experiments and included in the results. For the first iteration, we find the division point among all the deadlines, which means N computations. Then, for the second iteration, the worst case scenario results in $N-1$ computations. Therefore, considering that the computation of the Lambert function has an $O(1)$ complexity, the overall worst-case complexity of this implementation method is $O(N^2)$.

Our implementation ensures that the system is always in a cooling state with respect to all the remaining deadline constraints after each iteration and the temperature reaches its maximum in the first iteration according to Theorem VI.4. Moreover, the input obtained by Algorithm 1 satisfies the deadline constraints during each iteration while keeping the temperature below or equal to the optimal maximum temperature. This approach can be extended easily to dynamic

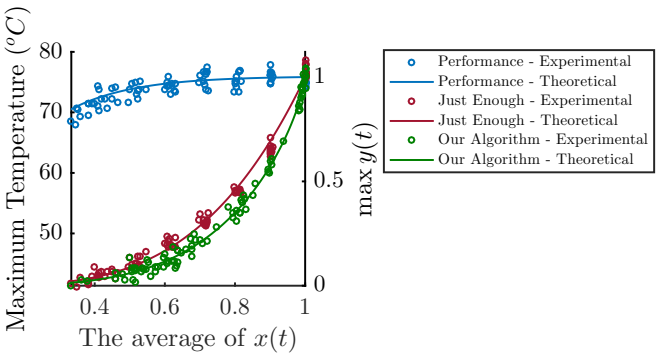


Fig. 5. Experimental and theoretical comparison of the three algorithms with varying workload, which is directly related to the average of $x(t)$.

TABLE II
MEAN AND VARIANCE OF THE DISTRIBUTIONS IN FIGURE 6

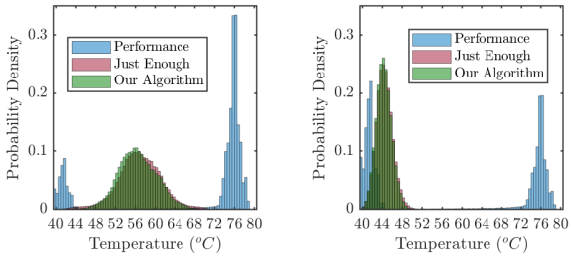
Workload intensity		Performance	Just enough	Our algorithm
High	Mean	68.4	56.8	56.4
	Variance	200	23.4	19.6
Low	Mean	57.9	44.4	44.3
	Variance	293	5.22	4.96

job arrivals by repeating this procedure whenever there is a new arrival.

Additionally, we have also tested the *performance* and the *just enough* policies explained in Section V. For the implementation of the *performance* policy, we use an algorithm that, at time t, chooses the input as $x(t) = 1$ if there is any unfinished job, otherwise, it chooses the input as $x(t) = 0$. For the implementation of the *just enough* policy, we use an algorithm that, at time t, chooses the input as $x(t) = \max\{\frac{\lambda(t;d_n)}{d_n - t}, 0\}$, $\forall n \in \{1,2,\ldots,N\}$.

To adjust the assigned resources, that is, to adapt the input $x(t)$, we use the `cpulimit` tool [25], which is available in the Linux operating system. This tool allows us to limit the CPU usage of a process by periodically idling the CPU using a closed loop control system with a high granularity. Although this may create oscillations in the temperature profile, we observed that the low pass filter characteristic of the heat transfer equation (1) smoothens the temperature profile. As a workload, we repetitively apply the square root operator to randomly chosen numbers. This guarantees that there is always a constant workload in the system. This application scenario and the `cpulimit` tool enable us to show the limitations and potential of our approach without including the effect of the workload dependencies and the discretization of the assigned resources respectively. We take the processor temperature, that is, $T(t)$, as the average of the measurements taken by the four built-in temperature sensors at time t. The conversion from $T(t)$ to $y(t)$ is explained in Section III. Each experiment starts with idling the system until the temperature converges to the ambient temperature T_a. To compute the Lambert function, we use Algorithm 743 of the TOMS database in the public domain mathematical library *Netlib* [26].

According to Theorem VI.4, in the multiple job case, the system reaches its maximum temperature always in the first



(a) High workload (b) Low workload

Fig. 6. The probability density of the measured temperature during the execution of 10 jobs for the three algorithms.

iteration when the corresponding system state is a heating state. Algorithm 1 computes $x(t)$ for the first iteration as if there is a single job. To illustrate the characteristic of the algorithm in the first iteration experimentally, we have conducted many experiments with a single job with a fixed deadline $d = 1.5s$. The workload of each job is variable and ranges from 0 to $1.5s$. Figure 5 shows the experimental results for the first iteration. The horizontal axis presents the average of $x(t)$, which refers to the workload and equals $\frac{p}{d}$. In the figure, the theoretical graphs are also shown. To compute these graphs, we also include the effect of leakage current mentioned in Section III as follows:

$$\dot{T}(t) = -\frac{1}{\tau}T(t) + \frac{1}{\tau}T_a + \frac{1}{\tau}(Q_D(t) + Q_L(t)), \qquad (34)$$

where $Q_D(t) = \alpha x(t)$ and the system parameters α and τ are found using the method that we proposed in our previous work [12]. Furthermore, $Q_L(t) = C_1 T(t)^2 e^{\frac{C_2}{T(t)}}$, as explained in Section III, and we apply a first order Taylor series approximation to the exponential term in $Q_L(t)$ as $Q_L(t) \approx CT(t)^2$.

As shown in Figure 5, the theoretical graphs fit well with the experimental results where the root mean square error (RMSE) between the theoretical graphs and the experimental results is approximately 1^oC. Clearly, the reduction that can be obtained depends on the workload. If the workload is maximum ($\frac{p}{d} = 1$), there is no flexibility and all the algorithms operate in essence the same. If the workload is low, then the difference between our algorithm and the *just enough* algorithm becomes insignificant, however, both algorithms give better results compared to the *performance* algorithm since they do not favor performance. According to these results, the *just enough* algorithm and our algorithm can process the workload with significantly lower temperature values, where the reduction in the maximum temperature is around 30^oC when the average of $x(t)$ is around 0.5. Furthermore, comparing our algorithm and the *just enough* algorithm, the difference in the maximum temperature values can reach 5^oC in favor of our algorithm. This difference is above 3^oC if the average of $x(t)$ is between 0.75 and 0.95. However, it is below 1^oC if the average of $x(t)$ is less then 0.55 or greater than 0.95. As a result, in terms of $y(t)$, the *just enough* algorithm can achieve up to 25% decrease in the maximum temperature while our algorithm can further decrease it up to 42% compared to the *performance* algorithm.

To show the benefit of our algorithm in the multiple job case, we have conducted 500 experiments with $N = 10$ jobs for each algorithm. In each experiment, the last deadline is

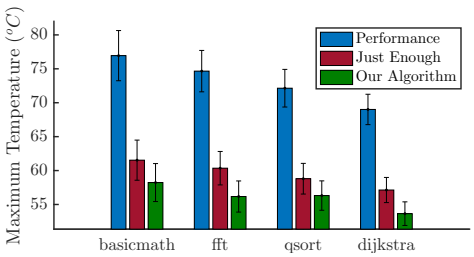


Fig. 7. Experimental Results for the three algorithms under different workload scenarios while keeping the avarage of $x(t)$ the same.

fixed to $150s$ and the other deadlines are chosen randomly in the time interval $[0, 150s]$. The experimental results show that after the first iteration, the temperature always decreases or keeps constant within a reasonable range, $\pm 0.5^oC$. Furthermore, the results give the same relation between the workload and the maximum temperature as in Figure 5, which is expected according to Theorem VI.4. To have a complete picture of the results, we show the probability density of the measurement data with histograms in Figure 6. Figure 6a shows a high workload case, where the workloads are chosen randomly such that $F_p(d_n) \in [0.65d_n, 0.95d_n]$, $\forall n \in \{1, 2, \ldots, N\}$, whereas, Figure 6b shows a low workload case, where the workloads are chosen randomly such that $F_p(d_n) \in [0.35d_n, 0.65d_n]$, $\forall n \in \{1, 2, \ldots, N\}$. Table II shows the mean and variance of the distributions in Figure 6a. Since the *performance* algorithm fully utilizes or idles the device, the temperature values are accumulated around very high and very low values. However, the *just enough* algorithm as in our algorithm do not reach very high temperature values. For this reason, the effect of the leakage current in the temperature is low in the *just enough* algorithm and our algorithm. By Lemma V.1, the mean value, which can be written as $\frac{1}{d}\int_0^d y(t)dt$, highly depends on the workload p since $\tau \ll d$. For this reason, we obtain very close mean values for the *just enough* algorithm and our algorithm. However, due to the leakage current, the mean value is slightly higher for the *performance* algorithm. On the other hand, Table II shows that we decrease the variance of the temperature with our algorithm by 16% compared to the *just enough* algorithm, which has an additional positive effect on the reliability of the device [3].

Furthermore, to illustrate the performance of our algorithm under different workload scenarios, we use MiBench [27], which provides a set of commercially representative applications. For each application, we have conducted 100 experiments and assigned the deadlines such that the ratio $\frac{p}{d}$ equals 0.85 since for this value the potential for temperature management is largest, and illustrates the differences between the evaluated approaches the best. The results in Figure 7 show the average of these experiments.

Our algorithm finds the critical time point and then switches the thermal policy at this time point to minimize maximum temperature. For example, if the initial temperature is low, it switches the policy from *performance* to *just enough* at the critical point. Additionally, periodically repeating the procedure in an online fashion can easily make it robust to unexpected events. Thus, it is easy to implement with a low overhead. On the other hand, our algorithm requires the workload information a priori and it does not create a significant advantage in low workload and very high workload situations.

VIII. Conclusion

In this article, we propose a resource allocation algorithm that minimizes the maximum processor temperature dynamically using built-in temperature sensors; we also prove the optimality of this algorithm. We focus on applications consisting of a set of jobs with known workload and deadlines. The algorithm can be implemented as an optimal thermal management policy that calculates the optimal maximum temperature and keeps the system on this temperature as long as possible. The algorithm can be can be easily adapted for use in an online fashion.

For validation purposes, we have implemented our algorithm on a commercial processor that is used in many mobile devices and compared it to two other algorithms, namely the *performance* and *just enough* algorithms. We experimentally show that our theoretical analysis matches the experimental results; and consequently, our approach can be used in real-life scenarios. Although the *performance* algorithm relies on a policy that aims to maximize the performance, the *just enough* algorithm relies on a policy that aims to provide sustainable and no more than required performance. Our experimental results show that the *just enough* algorithm decreases the maximum temperature by 25% on average, and our new algorithm further decreases it by 15% compared to the *performance* algorithm. Our experimental results also show that our algorithm decreases the variations in the temperature profile by 16% compared to the *just enough* algorithm, which further increases system reliability.

Acknowledgment

This work is funded by NWO (Dutch Research Council) through TTW Perspectief program Zero-Energy Internet-of-Things (P15-06) within Project P5.

References

[1] K. Sekar, "Power and thermal challenges in mobile devices," in *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, ser. MobiCom '13. ACM, 2013, p. 363–368.

[2] A. K. Coskun, T. S. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *2007 Design, Automation Test in Europe Conference Exhibition*, 2007, pp. 1–6.

[3] G. Bhat, G. Singla, A. K. Unver, and U. Y. Ogras, "Algorithmic optimization of thermal and power management for heterogeneous mobile platforms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 544–557, 2018.

[4] A. K. Singh, S. Dey, K. McDonald-Maier, K. R. Basireddy, G. V. Merrett, and B. M. Al-Hashimi, "Dynamic energy and thermal management of multi-core mobile platforms: A survey," *IEEE Design & Test*, vol. 37, no. 5, pp. 25–33, 2020.

[5] G. Singla, G. Kaur, A. K. Unver, and U. Y. Ogras, "Predictive dynamic thermal and power management for heterogeneous mobile platforms," in *2015 Design, Automation & Test in Europe Conference & Exhibition*, ser. DATE '15. IEEE, 2015, pp. 960–965.

[6] R. J. Wysocki. (2017) CPU performance scaling. [Online]. Available: https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html

[7] O. Sahin, P. T. Varghese, and A. K. Coskun, "Just enough is more: Achieving sustainable performance in mobile devices under thermal limitations," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, ser. ICCAD '15. IEEE, 2015, pp. 839–846.

[8] O. Sahin and A. K. Coskun, "Providing sustainable performance in thermally constrained mobile devices," in *Proceedings of the 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia*, ser. ESTIMedia'16. ACM, 2016, pp. 72–77.
[9] O. Sahin, L. Thiele, and A. K. Coskun, "Maestro: Autonomous QoS management for mobile applications under thermal constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1557–1570, 2018.
[10] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1503–1516, 2009.
[11] A. Prakash, H. Amrouch, M. Shafique, T. Mitra, and J. Henkel, "Improving mobile gaming performance through cooperative CPU-GPU thermal management," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '16. IEEE, 2016, pp. 1–6.
[12] B. Ozceylan, B. R. Haverkort, M. de Graaf, and M. E. T. Gerards, "A generic processor temperature estimation method," in *2019 25nd International Workshop on Thermal Investigations of ICs and Systems*, ser. THERMINIC '19. IEEE, 2019, pp. 1–6.
[13] E. W. Wächter, C. De Bellefroid, K. R. Basireddy, A. K. Singh, B. M. Al-Hashimi, and G. Merrett, "Predictive thermal management for energy-efficient execution of concurrent applications on heterogeneous multicores," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1404–1415, 2019.
[14] B. Ozceylan, B. R. Haverkort, M. de Graaf, and M. E. T. Gerards, "Improving temperature prediction accuracy using Kalman and particle filtering methods," in *2020 26nd International Workshop on Thermal Investigations of ICs and Systems*, ser. THERMINIC '20. IEEE, 2020.
[15] M. Yan, H. Wei, and M. Onabajo, "On-chip thermal profiling to detect malicious activity: System-level concepts and design of key building blocks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 3, pp. 530–543, 2021.
[16] X. Li, Z. Li, W. Zhou, and Z. Duan, "Accurate on-chip temperature sensing for multicore processors using embedded thermal sensors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2328–2341, 2020.
[17] S. Rahimipour, W. N. Flayyih, N. A. Kamsani, S. J. Hashim, M. R. Stan, and F. Z. B. Rokhani, "Low-power, highly reliable dynamic thermal management by exploiting approximate computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 10, pp. 2210–2222, 2020.
[18] A. Iranfar, M. Kamal, A. Afzali-Kusha, M. Pedram, and D. Atienza, "Thespot: Thermal stress-aware power and temperature management for multiprocessor systems-on-chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1532–1545, 2017.
[19] V. Chaturvedi, H. Huang, S. Ren, and G. Quan, "On the fundamentals of leakage aware real-time DVS scheduling for peak temperature minimization," *Journal of Systems Architecture*, vol. 58, no. 10, pp. 387–397, 2012.
[20] J. Zhou, J. Yan, J. Chen, and T. Wei, "Peak temperature minimization via task allocation and splitting for heterogeneous MPSoC real-time systems," *Journal of Signal Processing Systems*, vol. 84, no. 1, pp. 111–121, 2016.
[21] S. Sha, A. S. Bankar, X. Yang, W. Wen, and G. Quan, "On fundamental principles for thermal-aware design on periodic real-time multi-core systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 2, Feb. 2020. [Online]. Available: https://doi.org/10.1145/3378063
[22] J. Yue, T. Zhang, L. Yu, and T. Chen, "Leakage aware scheduling for maximum temperature minimization," in *2011 12th International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2011, pp. 245–250.
[23] D. Brooks, R. P. Dick, R. Joseph, and L. Shang, "Power, thermal, and reliability modeling in nanometer-scale microprocessors," *IEEE Micro*, vol. 27, no. 3, pp. 49–62, 2007.
[24] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the LambertW function," *Advances in Computational Mathematics*, vol. 5, no. 1, pp. 329–359, 1996.
[25] G. Herrmann, "cpulimit – limits the CPU usage of a process," 2015. [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/man1/cpulimit.1.html
[26] D. A. Barry, P. J. Culligan-Hensley, and S. J. Barry, "Real values of the w-function," *ACM Trans. Math. Softw.*, vol. 21, no. 2, p. 161–171, Jun. 1995. [Online]. Available: https://doi.org/10.1145/203082.203084
[27] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, 2001, pp. 3–14.

Baver Ozceylan is a graduate of Middle East Technical University (METU), Turkey (2014, BS and 2017, MS). From 2014 to 2018, he was with the Department of Electrical and Electronics Engineering at METU. Since 2018 he has been a PhD candidate at the Design and Analysis of Communication Systems Group of the University of Twente, The Netherlands. His research interests include mathematical modeling and analyzing, wireless communication systems, energy-efficient and energy-aware algorithms and scheduling, thermal modeling and temperature-aware scheduling.

Boudewijn R. Haverkort (Master and PhD, University of Twente, 1986 and 1991, respectively) is full professor and Dean of the Tilburg School of Humanities and Digital Sciences at Tilburg University, Netherlands. Since 2019. Before moving to Tilburg University, he was a full professor at the University of Twente since 2003, and from 1995 to 2002 he was a professor at RWTH Aachen, Germany. From 2009 to 2013 he was scientific director of the public-private Embedded Systems Institute, an applied research institute focusing on high-tech systems design. His field of interest is very wide, encompassing internet technology, cyber-physical systems, smart energy systems, energy management in data centers, computer performance and reliability evaluation, stochastic model checking, as well as data science. He is a Fellow of the IEEE since 2007, and has published around 200 papers about his scientific work in the above fields, and has chaired a large number of international conferences. Since 2016 he is chairman of the Dutch national research program on big data and applications.

Maurits de Graaf is an experienced Innovation Program and Project manager with a thorough scientific background. He has guided many research projects from first concepts to final implementation. He received his PhD in 1994 at the University of Amsterdam for the thesis 'Graphs and Curves on Surfaces'. After a period with the telecommunications research institute KPN Research, he started working with Thales Netherlands B.V. in 1999, mainly in the innovation department. Since 2010 he combines this with a part-time position at the University of Twente as associate professor at the department Mathematics of Operations Research (MOR). He co-authored over 30 publications.

Marco E. T. Gerards received the M.Sc. degrees in computer science and in applied mathematics from the University of Twente, Enschede, the Netherlands, in 2008 and 2011 respectively. He finished his Ph.D. thesis titled "Algorithmic power management: energy minimization under real-time constraints" in 2014. Then he worked as a postdoc, until 2016 when he became an assistant professor. His research interests are energy management for smart grids and sustainable computing.

APPENDIX A
PROOF OF LEMMA V.2

Lemma V.2. At time $t \in [0,d)$, if the system is in a heating state, it reaches a stable state after applying $x(t) = 1$ (fully utilized) for a time period $\mu(t;d) \in [0,d-t)$, where:

$$\mu(t;d) = \frac{\lambda(t;d) - y_{\text{ss}}(t;d)(d-t)}{1 - y_{\text{ss}}(t;d)}. \tag{15}$$

Then, the stable state temperature $y_{\text{ss}}(t;d)$ is defined as:

$$y_{\text{ss}}(t;d) = 1 - \frac{\lambda^c(t;d)}{\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}\Big)}, \tag{16}$$

and it satisfies the following equality:

$$e^{\frac{-\mu(t;d)}{\tau}} = \frac{1 - y_{\text{ss}}(t;d)}{y^c(t)}. \tag{17}$$

Proof. We first prove that the system is able to reach the temperature value expressed in (16) by applying full utilization, i.e., $x(t) = 1$. To do so, we show that this stable state temperature value, $y_{\text{ss}}(t;d)$, is always between $y(t)$ and 1. At t, the system is in a heating state, which means $y(t) < \frac{\lambda(t;d)}{d-t}$, which can be rearranged as $\lambda^c(t;d) < y^c(t)(d-t)$. Since $y^c(t)$ and $\lambda^c(t;d)$ are always positive:

$$0 \le \frac{\lambda^c(t;d)}{y^c(t)} < d - t. \tag{35}$$

Then, we can write the following inequality for the argument of the Lambert function in (16):

$$\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)} < \frac{d-t}{\tau}e^{\frac{d-t}{\tau}}. \tag{36}$$

Moreover, the inequality (35) can be rearranged to have the following inequality for the argument of the Lambert function in (16):

$$0 \le \frac{1}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}e^{\frac{1}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}} < \frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}. \tag{37}$$

The principal branch of the Lambert function, which means that the domain is positive, is always positive and increasing [24]. Therefore, first combining (36) and (37) and then using (13), we write the following inequalities for the output of the Lambert function in (16):

$$0 \le \frac{1}{\tau}\frac{\lambda^c(t;d)}{y^c(t)} < W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}\Big) < \frac{d-t}{\tau}. \tag{38}$$

This proves that $y_{\text{ss}}(t;d)$ defined in (16) is always between $y(t)$ and $\frac{\lambda(t;d)}{d-t}$, which means the system can reach this temperature value by fully utilizing, i.e., $x(t) = 1$.

We now prove that the system reaches this $y_{\text{ss}}(t;d)$ value at $t + \mu(t;d)$, where $\mu(t;d) \in [0,d-t)$, and show that this time point can be computed by (15). Since the input is constant in the time interval $[t, t+\mu(t;d)]$, we can use (4) to compute the output $y(t+\mu(t;d))$, which also equals $y_{\text{ss}}(t;d)$, as follows:

$$y(t+\mu(t;d)) = y_{\text{ss}}(t;d) = 1 - y^c(t)e^{\frac{-\mu(t;d)}{\tau}}.$$

This proves the equality (17). After substituting (16) using the identity (14), we can write:

$$\mu(t;d) = d - t - \tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}\Big). \tag{39}$$

This proves that $\mu(t;d)$ is always in the time interval $[0,d-t)$ using (38). To obtain (15), we rearrange (16) as:

$$\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda^c(t;d)}{y^c(t)}\Big) = \frac{\lambda^c(t;d)}{1 - y_{\text{ss}}(t;d)},$$

then combine with (39).

Lastly, we shows that, at $t+\mu(t;d)$, the system is in stable state, i.e., $y(t+\mu(t;d)) = \frac{\lambda(t+\mu(t;d);d)}{d-t-\mu(t;d)}$. Since $x(t) = 1$ in the time interval $[t, t+\mu(t;d)]$, $\lambda(t+\mu(t;d);d) = \lambda(t;d) - \mu(t;d)$. After substituting (15), we have:

$$\lambda(t+\mu(t;d);d) = y_{\text{ss}}(t;d)\frac{\lambda^c(t;d)}{1 - y_{\text{ss}}(t;d)}.$$

On the other hand, we compute the remaining time $d-t-\mu(t;d)$ using (15) as

$$d - t - \mu(t;d) = \frac{\lambda^c(t;d)}{1 - y_{\text{ss}}(t;d)}.$$

Therefore:

$$\frac{\lambda(t+\mu(t;d);d)}{d-t-\mu(t;d)} = y_{\text{ss}}(t;d).$$

This means that the system is in a stable state at $t+\mu(t;d)$ since $y_{\text{ss}}(t;d) = y(t+\mu(t;d))$. ■

APPENDIX B
PROOF OF LEMMA V.3

Lemma V.3. At time $t \in [0,d)$, if the system is in a cooling state, it reaches a stable state after applying $x(t) = 0$ (idled) for a time period $\mu(t;d) \in [0,d-t)$, where:

$$\mu(t;d) = \frac{y_{\text{ss}}(t;d)(d-t) - \lambda(t;d)}{y_{\text{ss}}(t;d)}. \tag{18}$$

Then, the stable state temperature $y_{\text{ss}}(t;d)$ is defined as:

$$y_{\text{ss}}(t;d) = \frac{\lambda(t;d)}{\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}\Big)}. \tag{19}$$

Proof. We first prove that the system is able to reach the temperature value expressed in (19) by idling, i.e., $x(t) = 0$. To do so, we show that this stable state temperature value, $y_{\text{ss}}(t;d)$, is always between 0 and $y(t)$. At t, the system is in a cooling state, which means $y(t) > \frac{\lambda(t;d)}{d-t}$, and $d - t \ge \lambda(t;d) > 0$ according to the system definition, hence:

$$d - t > \frac{\lambda(t;d)}{y(t)} > 0, \tag{40}$$

Thus, we can write the following inequality for the argument of the Lambert function in (19):

$$\frac{d-t}{\tau}e^{\frac{d-t}{\tau}} > \frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}. \tag{41}$$

Moreover, the inequality (40) can be rearranged to have the following inequality for the argument of the Lambert function in (19):

$$\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)} > \frac{1}{\tau}\frac{\lambda(t;d)}{y(t)}e^{\frac{1}{\tau}\frac{\lambda(t;d)}{y(t)}} > 0 \tag{42}$$

The principal branch of the Lambert function, which means that the domain is positive, is always positive and increasing [24]. Therefore, first combining (41) and (42) and then using (13), we write the following inequalities for the output of the Lambert function in (19):

$$\frac{d-t}{\tau} > W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}\Big) > \frac{1}{\tau}\frac{\lambda(t;d)}{y(t)} > 0. \tag{43}$$

This proves that $y_{\text{ss}}(t;d)$ defined in (19) is always between $\frac{\lambda(t;d)}{d-t}$ and $y(t)$, which means the system can reach this temperature value by idling, i.e., $x(t)=0$.

We now prove that the system reaches this $y_{\text{ss}}(t;d)$ value at $\mu(t;d)$, where $\mu(t;d)\in[0,t-d)$, and show that this time point can be computed by (18). Since the input is constant in the time interval $[0,\mu(t;d)]$, we can use (4) to compute the output $y(t+\mu(t;d))$, which also equals $y_{\text{ss}}(t;d)$, as follows:

$$y(t+\mu(t;d)) = y_{\text{ss}}(t;d) = y(t)e^{\frac{-\mu(t;d)}{\tau}}.$$

After substituting (19) and using the identity (14), we can write:

$$\mu(t;d) = t+d-\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}\Big). \tag{44}$$

This proves that $\mu(t;d)$ is always in the time interval $[0,d-t]$ using (43). To obtain (18), we rearrange (19) as:

$$\tau W_0\Big(\frac{e^{\frac{d-t}{\tau}}}{\tau}\frac{\lambda(t;d)}{y(t)}\Big) = \frac{\lambda(t;d)}{y_{\text{ss}}(t;d)},$$

then combine with (44).

Lastly, we shows that, at $t+\mu(t;d)$, the system is in a stable state, i.e., $y(t+\mu(t;d)) = \frac{\lambda(t+\mu(t;d);d)}{d-t-\mu(t;d)}$. Since $x(t)=0$ in the time interval $[t,t+\mu(t;d)]$, $\lambda(t+\mu(t;d)=\lambda(t;d)$. On the other hand, we compute the remaining time $d-t-\mu(t;d)$ using (15) as

$$d-t-\mu(t;d) = \frac{\lambda(t;d)}{y_{\text{ss}}(t;d)}.$$

Therefore:

$$\frac{\lambda(t+\mu(t;d);d)}{d-t-\mu(t;d)} = y_{\text{ss}}(t;d).$$

This means that the system is in a stable state at $t+\mu(t;d)$ since $y_{\text{ss}}(t;d) = y(t+\mu(t;d))$. ■

APPENDIX C
PROOF OF LEMMA V.4

Lemma V.4. At $t=0$, for $\delta\in[0,d]$, the objective function (11a) of the resource allocation problem (11) has a lower bound in the time interval $[\delta,d]$ as:

$$\max_{t\in[\delta,d]} y(t) \geq 1 - \frac{\lambda^c(0;d)+\tau y^c(0)e^{\frac{-\delta}{\tau}}}{d-\delta+\tau}. \tag{20}$$

Proof. We integrate both sides of the differential equation (2) from δ to d:

$$(y(d)-y(\delta)) = -\frac{1}{\tau}\int_\delta^d y(\sigma)d\sigma + \frac{1}{\tau}\int_\delta^d x(\sigma)d\sigma,$$

which can be rewritten as follows:

$$\int_\delta^d y(\sigma)d\sigma + \tau y(d) = \int_\delta^d x(\sigma)d\sigma + \tau y(\delta).$$

Since any feasible $x(t)$ must satisfy the deadline constraint, i.e., $\int_0^d x(\sigma)d\sigma \geq \lambda(0;d)$, we can write:

$$\int_\delta^d x(\sigma)d\sigma \geq \lambda(0;d) - \int_0^\delta x(\sigma)d\sigma.$$

Hence:

$$\int_\delta^d y(\sigma)d\sigma + \tau y(d) \geq \lambda(0;d) - \int_0^\delta x(\sigma)d\sigma + \tau y(\delta).$$

After using (3) to compute $y(\delta)$:

$$\int_\delta^d y(\sigma)d\sigma + \tau y(d) \geq \lambda(0;d) - \int_0^\delta (1-e^{\frac{-(\delta-\sigma)}{\tau}})x(\sigma)d\sigma + \tau y(0)e^{\frac{-\delta}{\tau}}. \tag{45}$$

Since $x(t)\leq 1$ according to the system definition and $1-e^{\frac{-(\delta-t)}{\tau}} \geq 0$, $\forall t\in[0,\delta]$, we can write:

$$\int_0^\delta (1-e^{\frac{-(\delta-\sigma)}{\tau}})x(\sigma)d\sigma \leq \delta - \tau + \tau e^{\frac{-(\delta-0)}{\tau}}.$$

Thus, (45) can be rewritten as follows:

$$\int_\delta^d y(\sigma)d\sigma + \tau y(d) \geq d-\delta+\tau-\lambda^c(0;d) - \tau y^c(0)e^{\frac{-\delta}{\tau}}. \tag{46}$$

The maximum value of a function is always larger than or equal to its mean hence we have $\max y(t) \geq \frac{1}{d-\delta}\int_\delta^d y(\sigma)d\sigma$ in the time interval $[\delta,d]$. Additionally, $\max y(t) \geq y(d)$. Therefore, we can write:

$$(d-\delta+\tau)\max_{t\in[\delta,d]} y(t) \geq \int_\delta^d y(\sigma)d\sigma + \tau y(d). \tag{47}$$

Combining (46) and (47):

$$(d-\delta+\tau)\max_{t\in[\delta,d]} y(t) \geq d-\delta+\tau-\lambda^c(0;d) - \tau y^c(0)e^{\frac{-\delta}{\tau}}.$$

Dividing by $(d-\delta+\tau)$ leads to the lower bound (20). ■

APPENDIX D
PROOF OF LEMMA VI.1

Lemma VI.1. Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28):

$$F_x^{(d_v)}(t) \geq F_x^{(d_k)}(t),\ \forall t\in[0,\min\{d_v,d_k\}],\ \forall k\in\{1,\ldots,N\}. \tag{30}$$

Proof. To prove this lemma, we separately consider all three possible cases, which are $y_0 > \frac{\lambda(0;d_v)}{d_v}$ (cooling state), $y_0 = \frac{\lambda(0;d_v)}{d_v}$ (stable state) and $y_0 < \frac{\lambda(0;d_v)}{d_v}$ (heating state) in the time interval $[0,\min\{d_v,d_k\}]$:

Case 1. If $y_0 > \frac{\lambda(0;d_v)}{d_v}$ (cooling state), $y_0 > y_{ss}(0,d_v)$ due to (22). Hence, $\forall k \in \{1,\dots,N\}$, $y_0 > y_{ss}(0,d_v) > y_{ss}(0,d_k)$. This means that $y_0 > \frac{\lambda(0;d_k)}{d_k}$ (cooling state) for each d_k due to (22) and this also means that (21) is applicable for both $F_x^{(d_v)}(t)$ and $F_x^{(d_k)}(t)$. Thus, $\forall t \in [0, \min\{d_v,d_k\}]$:

$$y_0 > y_{ss}(0,d_v) > y_{ss}(0,d_k) \Rightarrow\ y_0 e^{\frac{-\mu(0;d_v)}{\tau}} > y_0 e^{\frac{-\mu(0;d_k)}{\tau}}$$
$$\Rightarrow \mu(0;d_k) > \mu(0;d_v) \Rightarrow\ F_x^{(d_v)}(t) > F_x^{(d_k)}(t).$$

Case 2. If $y_0 = \frac{\lambda(0;d_v)}{d_v}$ (stable state), $y_0 = y_{ss}(0,d_v)$ due to (24). Hence, $\forall k \in \{1,\dots,N\}$, $y_0 = y_{ss}(0,d_v) > y_{ss}(0,d_k)$. This means that $y_0 > \frac{\lambda(0;d_k)}{d_k}$ (cooling state) for each d_k due to (22) and this also means that (23) is applicable for $F_x^{(d_v)}(t)$, while (21) is for $F_x^{(d_k)}(t)$. Thus, $\forall t \in [0, \min\{d_v,d_k\}]$:

$$y_0 = y_{ss}(0,d_v) > y_{ss}(0,d_k) \Rightarrow\ \frac{dF_x^{(d_v)}(t)}{dt} > \frac{dF_x^{(d_k)}(t)}{dt}$$
$$\Rightarrow F_x^{(d_v)}(t) > F_x^{(d_k)}(t).$$

Case 3. If $y_0 < \frac{\lambda(0;d_v)}{d_v}$ (heating state), $y_0 < y_{ss}(0,d_v)$ due to (26). Hence, there are three possible cases, which are $y_0 > y_{ss}(0,d_k)$ (cooling state), $y_0 = y_{ss}(0,d_k)$ (stable state) and $y_0 < y_{ss}(0,d_k)$ (heating state):

- If $y_0 > y_{ss}(0,d_k)$, then $y_{ss}(0,d_v) > y_0 > y_{ss}(0,d_k)$, $\forall k \in \{1,\dots,N\}$. This means that $y_0 > \frac{\lambda(0;d_k)}{d_k}$ (cooling state) for each d_k due to (22) and this also means that (25) is applicable for $F_x^{(d_v)}(t)$, while (21) is for $F_x^{(d_k)}(t)$. Thus, $\forall t \in [0, \min\{d_v,d_k\}]$:

$$y_{ss}(0,d_v) > y_0 > y_{ss}(0,d_k) \Rightarrow\ \frac{dF_x^{(d_v)}(t)}{dt} > \frac{dF_x^{(d_k)}(t)}{dt}$$
$$\Rightarrow F_x^{(d_v)}(t) > F_x^{(d_k)}(t).$$

- If $y_0 = y_{ss}(0,d_k)$, then $y_{ss}(0,d_v) > y_0 = y_{ss}(0,d_k)$, $\forall k \in \{1,\dots,N\}$. This means that $y_0 = \frac{\lambda(0;d_k)}{d_k}$ (stable state) for each d_k due to (24) and this also means that (25) is applicable for $F_x^{(d_v)}(t)$, while (23) is for $F_x^{(d_k)}(t)$. Thus, $\forall t \in [0, \min\{d_v,d_k\}]$:

$$y_{ss}(0,d_v) > y_0 = y_{ss}(0,d_k) \Rightarrow\ \frac{dF_x^{(d_v)}(t)}{dt} > \frac{dF_x^{(d_k)}(t)}{dt}$$
$$\Rightarrow F_x^{(d_v)}(t) > F_x^{(d_k)}(t).$$

- If $y_0 < y_{ss}(0,d_k)$, then $y_{ss}(0,d_v) > y_{ss}(0,d_k) > y_0$, $\forall k \in \{1,\dots,N\}$. This means that $y_0 < \frac{\lambda(0;d_k)}{d_k}$ (heating state) for each d_k due to (26) and this also means that (25) is applicable for both $F_x^{(d_v)}(t)$ and $F_x^{(d_k)}(t)$. Thus, $\forall t \in [0, \min\{d_v,d_k\}]$:

$$y_{ss}(0,d_v) > y_{ss}(0,d_k) > y_0$$
$$\Rightarrow 1-(1-y_0)e^{\frac{-\mu(0;d_v)}{\tau}} > 1-(1-y_0)e^{\frac{-\mu(0;d_k)}{\tau}}$$
$$\Rightarrow \mu(0;d_k) < \mu(0;d_v)$$
$$\Rightarrow F_x^{(d_v)}(t) > F_x^{(d_k)}(t),\ \forall t \in [0, \min\{d_v,d_k\}].$$

As a result, if d_v is chosen according to (28), (30) holds for each d_k where $k \in \{1,2\dots,N\}$ in the time interval $[0, \min\{d_v,d_k\}]$. ■

APPENDIX E
PROOF OF LEMMA VI.3

Lemma VI.3. Given a vector of workloads P, a vector of deadlines D and the initial value y_0. For the deadline d_v given by (28), the input obtained by $\mathtt{SJ}(\lambda(0;d_v),d_v,y_0)$ leads to an output at $t = d_v$ such that the system is a cooling state for each d_k where $k \in \{1,2,\dots,v\}$, i.e.:

$$y^{(d_v)}(d_v) > \frac{\lambda^{(d_v)}(d_v;d_k)}{d_k - d_v},\ \forall d_k \in (d_v, d_N]. \tag{31}$$

Proof. According to Lemma VI.1, $F_x^{(d_v)}(t) \geq F_x^{(d_k)}(t)$, $\forall t \in [0,d_v]$ and $\forall k \in \{v+1, v+2, \dots, N\}$. This also means that $\lambda^{(d_k)}(d_v;d_k) > \lambda^{(d_v)}(d_v;d_k)$ due to (29). We first assume that:

$$y^{(d_k)}(d_v) \geq \frac{\lambda^{(d_k)}(d_v;d_k)}{d_k - d_v},\ \forall d_k \in (d_v, d_N], \tag{48}$$

which is proven later. If the output value $y^{(d_k)}(t)$ is greater than or equal to the ratio between the remaining workload $\lambda^{(d_k)}(t;d_k)$ and the remaining time $d_k - t$ at any time point $t < d_k$ as in (48), (22) or (24) is applicable for $y^{(d_k)}(t)$ in the time interval $[0,d_k]$ (see also Figure 4). Therefore, the stable state temperature $y_{ss}(0,d_k)$ is also greater than or equal to this ratio at any time $t < d_k$, which means:

$$y_{ss}(0,d_k) \geq \frac{\lambda^{(d_k)}(d_v;d_k)}{d_k - d_v},\ \forall d_k \in (d_v, d_N].$$

This proves the inequalities (31) since $y^{(d_v)}(d_v) = y_{ss}(0,d_v) > y_{ss}(0,d_k)$ due to (28) and $\lambda^{(d_k)}(d_v;d_k) > \lambda^{(d_v)}(d_v;d_k)$.

We now prove the inequalities (48) by contradiction. Let $d_k^* > d_v$ be a deadline that does not follow the condition in (48), which means:

$$y^{(d_k^*)}(d_v) < \frac{\lambda^{(d_k^*)}(d_v;d_k^*)}{d_k^* - d_v}. \tag{49}$$

If the output value $y^{(d_k^*)}(t)$ is less than the ratio between the remaining workload $\lambda^{(d_k^*)}(t;d_k^*)$ and the remaining time $d_k^* - t$ at any time $t < d_k^*$ as in (49), (26) is applicable for $y^{(d_k^*)}(t)$. According to (26), $y(t) < \frac{\lambda(t;d)}{d-t}$ when $t < \mu(0;d)$, hence, (49) means that $d_v < \mu(0;d_k^*)$. Therefore, in the time interval $[0,d_v]$, $F_x^{(d_k^*)}(t) = t$ due to (25). This means that $x(t) = 1$, $\forall t \in [0,d_v]$, hence, there is no input that the output value at $t = d_v$ is greater than $y^{(d_k^*)}(d_v)$, which leads to:

$$y^{(d_k^*)}(d_v) \geq y^{(d_v)}(d_v) = y_{ss}(0,d_v). \tag{50}$$

Moreover, $y^{(d_k^*)}(t)$ is an increasing function at $t = d_v$ due to (26), which leads to:

$$y^{(d_k^*)}(d_v) < y^{(d_k^*)}(d_k^*) = y_{ss}(0,d_k^*). \tag{51}$$

Combining (50) and (51), we have $y_{ss}(0,d_k^*) > y^{(d_k^*)}(d_v) \geq y_{ss}(0,d_v)$. However, our assumption that (48) is false leads to a contradiction since the selection of d_v follows (28). Therefore, there cannot exist a deadline that does not follow the condition in (48).

■