RC24331 (W0708-051) August 13, 2007
Computer Science

IBM Research Report

Temperature-Aware Scheduling: When Is System-Throttling Good Enough?

Deepak Rajan, Philip S. Yu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Temperature-aware scheduling: When is system-throttling good enough?

Deepak Rajan and Philip S. Yu
IBM T.J. Watson Research Center 19 Skyline Drive, Hawthorne, NY 12524, USA
drajan,psyu@us.ibm.com

Abstract

Power-aware operating systems ensure that the system temperature does not exceed a threshold by utilizing system-throttling. In this technique, the system load (or alternatively, the clock speed) is scaled when the temperature hits this threshold. At other times, the system operates at maximum load.

In this paper, we show that such simple system-throttling rules are in fact the best one can achieve under certain assumptions. We show that maintaining a constant operating speed (and thus temperature) always does more work than operating in alternating periods of cooling and heating. As a result, for certain settings and for a reasonable temperature model, we prove that system-throttling is the most effective temperature aware-scheduling. Naturally, these assumptions do not always hold; we also discuss the scenario when some of our assumptions are relaxed, and argue why one needs more complex scheduling algorithms in this case.

1 Introduction

Energy and temperature management of processor systems is an increasingly important problem as the power consumption of these chips rises drastically with every new generation. At the same time, the rate of technological improvements in cooling systems has not been keeping pace [3]. Naturally, this has resulted in a large body of work that attempts to incorporate energy and temperature considerations into processor scheduling levels. Some of these are incorporated at the system level, where the on-chip architecture scales the speed of the processor if it is getting too hot.

With advances in processor technology, this is also possible at the operating system level since most modern day processors have interfaces that allow the user to control its speed in real-time. The current processors by Intel, AMD, and IBM now allow a mechanism called dynamic voltage scaling (DVS) to control the clock speed of the processor [14] by varying the supply voltage. Most operating systems now also include commands which allow the user to access this interface (e.g., `cpufreq` in Linux).

This allows the user to design efficient energy-aware algorithms (to schedule tasks and scale processor speeds), thus improving the performance of the system subject to energy or temperature constraints. A detailed investigation into the various trade-offs involved, the techniques utilized, and mechanisms for implementing dynamic thermal management (DTM) can be found in [7]; observe that DVS is just one of the mechanisms for implementing DTM. In Section 5, we describe some of these techniques in detail.

However, the focus of this paper is quite different; we try to analyze the settings under which system-enforced throttling is sufficient. We argue that simple system-throttling rules are the best one can do in many settings. While this may not be true in all settings, this still justifies the use of such simple rules at a system level. We believe that the theoretical justification for the effectiveness of simple system-throttling policies as a first-pass approach is the main contribution of this work.

To make our case, we argue that we would like to maximize the amount of work done over the time horizon. We believe this is a good metric to consider since it is applicable in a variety of settings. Even when the objective is more complex, this is still often a good rule of thumb: more work done is

better. We make a clear distinction between decisions involving job selection and selection of system operating load. We do not concern ourselves with scheduling decisions in the traditional task-scheduling sense (which job to run next based on a certain objective, etc.), but instead with the decision of how much processing to give to the currently active jobs. We prove that under many scenarios, simple system-throttling rules are sufficient to guarantee the maximum amount of work done.

In Section 2, we introduce the thermodynamics models for estimating the energy and temperature characteristics of the system. We also present our scheduling objectives, and introduce system-throttling, a simple scheduling policy to ensure that the temperature of the system does not exceed its threshold. The main contributions of this paper are presented in Section 3. In this section, we argue that system-throttling is the most effective scheduling policy under a variety of circumstances, and derive expressions for choosing the optimal load under this policy. In Section 4, we consider some extensions for which system-throttling is still optimal. In Section 5, we closely analyze some of our assumptions, and present scenarios where system-throttling can be improved upon by more complex scheduling policies. We also present a short literature review of some of the work in this domain, characterizing them based on the assumptions of the models considered. Finally, we conclude in Section 6 summarizing the contributions of this paper.

2 Problem setting

In this section, we introduce the relevant concepts for the rest of the paper. We first present the heat model for estimating the temperature of the system at any time. Then, we discuss the operating constraints that the system has to enforce, and introduce system-throttling, a simple scheduling policy for ensuring the operating constraints. We intend to argue that system-throttling is a very effective policy for maximizing the amount of work done subject to the operating constraints.

2.1 Heat model

We assume that the system is cooled using Newton's law; i.e. $dT = \rho(T_{amb} - T)$, where dT is the instantaneous rate of change of temperature, T_{amb} is the ambient temperature (or the temperature of the cooling medium), and ρ is a positive constant. Here, the system being modeled is the processor itself, and T_{amb} is the temperature of the cooling medium. This principle of heat-loss is not exact, but provides a good first-order approximation to the process [20].

To model the heat gain due to the load of the processors, we assume that $dT = \beta P$, where P is the power dissipated by the processors, and β is a positive constant. Again, this model provides a good first order approximation, and has been used often in literature [1]. Combining the two effects, one can estimate the instantaneous rate of change of temperature as follows:

$$dT = \beta P + \rho(T_{amb} - T). \tag{1}$$

Equation was used in [4] to develop a thermal management scheme for determining energy consumptions of processors on the fly. In the paper, the authors also measured their temperature estimates against real measurements, and empirically showed that this model is very accurate.

We model the power dissipated by the processor as proposed by [22]. Thus, $P = \ell^\alpha$, where α is a positive constant strictly larger than 1, and ℓ is the speed of the processor. We use processor speed and operating load synonymously in this paper. For CMOS-based processors, it is empirically believed that $\alpha = 3$ is a good approximation [8]. For the rest of this paper, we assume *w.l.o.g* that $T_{amb} = 0$ (this can be done by reducing all temperatures by T_{amb}).

To summarize, the instantaneous change of temperature can be modeled as

$$dT = \beta \ell^\alpha + \rho(T_{amb} - T). \tag{2}$$

This is an ordinary differential equation that can be easily solved [17] for constant load. Let us define $\tau(\ell) = \beta\ell^\alpha/\rho$. If the load in time interval $[t_1, t_2]$ is ℓ, then we get

$$T(t_2) = \tau(\ell) + (T(t_1) - \tau(\ell))e^{-\rho(t_2-t_1)}, \tag{3}$$

where $T(t)$ denotes the temperature at time t. We also assume that at time 0, the system temperature is T_{amb}. For any given load, we can calculate the temperature at any time using equation 3. Conversely, given an initial temperature T_1 and load ℓ, we can calculate the time taken to reach temperature T_2 using the following equation.

$$t = -\frac{1}{\rho}\log\frac{\tau(\ell) - T_2}{\tau(\ell) - T_1} \tag{4}$$

Similar to $\tau(\ell)$, we also define the load that stabilizes the system at temperature T by $\ell(T)$. This quantity can be easily calculated as

$$\ell(T) = (T\rho/\beta)^{\frac{1}{\alpha}} \tag{5}$$

2.2 Constraints and objectives

We consider a single-processor system. In section 4, we discuss the multi-processor case. We assume, by scaling, *w.l.o.g.* that the maximum load of the machine is 1. Thus, at full load, the processor can completes x units of work in x time units. Furthermore, at load ℓ, machine can complete x units of work in time $= x/\ell$. Conversely, if the machine runs at load ℓ for time t, then the amount of work done (which we denote by W) is equal to ℓt. Let the maximum system design temperature be T_{max}. We can assume that $T_{max} < \tau(1)$; else the maximum system temperature will not be reached.

To meet this constraint, the system scheduler must modify the system load ℓ so that the temperature stays below T_{max}. Let ℓ_0 be the maximum load at which the system never reaches the maximum temperature. This can be evaluated easily:

$$\ell_0 = (T_{max}\rho/\beta)^{1/\alpha} \tag{6}$$

2.3 System throttling

One way to ensure that the system never exceeds T_{max} is to always operate each machine at load not larger than ℓ_0, and go full-blast while the temperature is less than T_{max}. In fact, this is the most commonly used rule for system-throttling.

1. If the temperature reaches T_{max}, then the system will enforce throttling (load is set to ℓ_0).
2. If the temperature is below T_{max} (due to idling periods), the system increases the load to $1 \geq \ell^* > \ell_0$, until T_{max} is reached again.

In Figure 1, we plot the temperature and load of the system as a function of time for such a system under normal operation. From time 0, the system operates at load ℓ^*, until temperature T_{max} is reached at time t_1. At this point, the system is throttled to run at load ℓ_0. Thus, the system operates in two modes, either at load ℓ^*, or at load ℓ_0, while ensuring that the temperature does not go above T_{max}. In Section 3, we show that such a simple throttling rule is the most effective, and show how the best value of ℓ^* can be calculated. We prove that going full-blast is sub-optimal (ℓ^* may be less than 1, depending on the system). Nevertheless, we show that it is optimal to operate at load ℓ^* until temperature T_{max} is reached, and then to operate at load ℓ_0.

In general, the scheduler can decide to go at any intermediate load ($\ell_0 \leq \ell \leq 1$) depending on the state of the system. This can potentially depend on various factors, including information about the tasks at hand and in the queue. Furthermore, the scheduler may choose to operate at any load

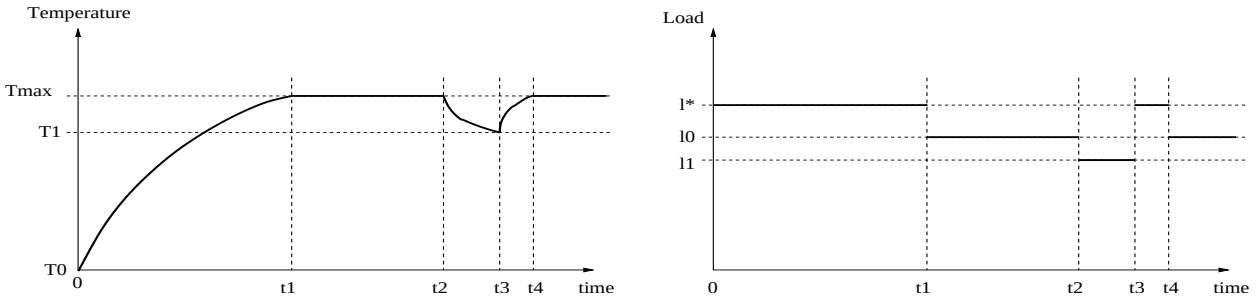

Figure 1: System-throttling

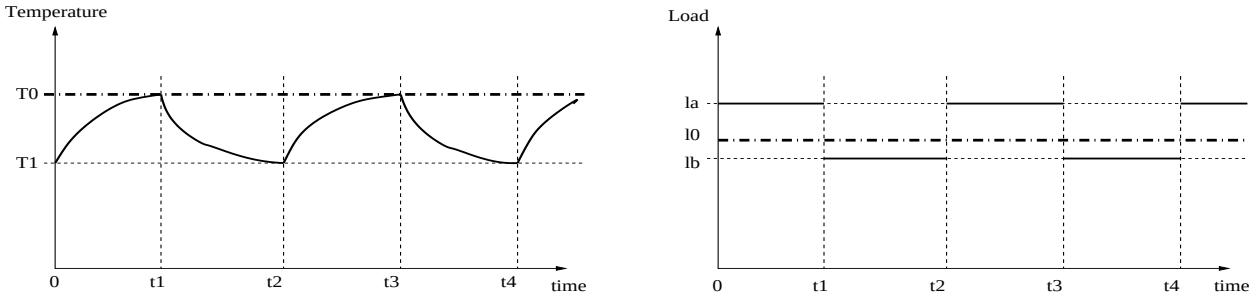

Figure 2: Zig-zag and Constant policies

$0 \leq \ell \leq l_0$ so as to increase the rate of cooling. This decision may again be made based on a variety of factors. In Section 3, we show that the simple system throttling is the most efficient operating principle for a large range of settings.

3 Analysis and results

In this section, we present the main results of this work. We argue that the operating rules presented in Section 2.3 maximize the amount of work done while ensuring that the system temperature does not exceed the threshold T_{max}. To make this case, consider an alternate strategy, which we shall refer to as the *Zig-zag* operating policy. In this strategy, the scheduler operates between alternate stages of cooling and heating, in an attempt to increase the amount of work done while ensuring that the maximum temperature is below T_{max}.

- The operating temperature range of this policy is $[T_1, T_0]$, where $T_0 < T_{max}$.
- The operating load of this policy is ℓ_a during cooling and ℓ_b during heating. $(\ell_b < \ell_0 < \ell_a)$.

We illustrate this policy in Figure 2 using solid lines. The system operates between temperatures T_1 and T_0, alternating periods of cooling (load ℓ_b) and heating (load ℓ_a). We remark that all policies that do not maintain a constant load can be characterized using an appropriately combination of Zig-zag policies, one corresponding to each cooling and heating phase.

We refer to a policy that keeps the temperature constant as *Constant*. In such a policy, if the current temperature is T_0, the system operates at the load (ℓ_0) that maintains the temperature at T_0. We illustrate this policy in Figure 2 using dashed-and-dotted lines.

3.1 Outline of proof

We argue that system throttling dominates any zig-zagging strategy in two parts. First, we prove, in Section 3.2, that given the current temperature T_0, it is more effective to continue running at the speed

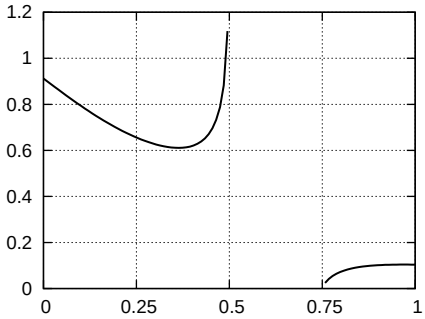

Figure 3: Function $\mathcal{F}(x, 0.75, 0.5, 3)$

that maintains this temperature, than to reduce the temperature to T_1 and increase it back to T_0. Let the load that attains stable temperature T_0 be ℓ_0; we show that it is not possible to get more work done by alternating between stages of system cooling (load $\leq \ell_0$) and heating (load $\geq \ell_0$) until T_0 is reached. This proves that the Constant policy dominates the Zig-zag policy; see Figure 2.

Given that it is better to be at a constant temperature, it is clearly best to stay at the highest possible temperature, which is T_{max}. Therefore, it behooves us to attain this temperature soon. The simple system-throttling rule presented in Section 2.3 achieves this goal by going at full-blast. We show in Section 3.3 that this may be sub-optimal, and that increased performance may be achieved by operating at load $\ell < 1$, while increasing the system temperature to T_{max}. We show this by proving that there exists an optimal load $1 \geq \ell^* > \ell_0$ which the system should use at any point when the temperature is less than T_{max}. This load should be maintained until T_{max} is attained. In Section 3.3, we show how the optimal value of ℓ^* can be evaluated.

But first, we introduce the following function, which will be used repeatedly in our proofs to characterize the trade-offs involved in operating at alternating periods of cooling and heating. To illustrate, we plot the function $\mathcal{F}(x, 0.75, 0.5, 3)$ in Figure 3. We state the technical result in Lemma 1 without proof since it is a tedious, but straightforward, application of univariate calculus.

$$\mathcal{F}(x, p, q, \alpha) = (x - p) log \frac{x^\alpha - q^\alpha}{x^\alpha - p^\alpha} \tag{7}$$

Lemma 1. *For given p, q, α such that $p > q$ and $\alpha > 1$, then $\mathcal{F}(x)$ has a maximizer x^* for $x > p$, and a minimizer $\hat{x}$ for $x < q$. Furthermore, $\mathcal{F}(x_1) > \mathcal{F}(x_2)$, for all $x_1 < q$ and $x_2 > p$. Finally, for $x > p$, the function $\mathcal{F}$ increases monotonically to x^*.*

3.2 Keeping temperature constant

Now, we are ready to prove the main result in this paper. We show that more work is done by keeping the temperature constant, than by decreasing it and increasing it back to the initial level. In other words, we show that the Constant policy dominates the Zig-zag policy; see Figure 2.

Theorem 1. *Let the temperature of the system be T_0. Let W_z be the amount of work done by cooling the system to temperature $T_1 < T_0$ and then increasing it back to temperature T_0. Let W_c be the work done by keeping the temperature constant at T_0. Then, $W_c > W_z$.*

Proof. Let ℓ_a be the load during the cooling phase, and ℓ_b be the load when the system heats back up to temperature T_0. Also, let ℓ_0 and ℓ_1 be the load that maintain the system temperature at T_0 and T_1, respectively; in other words $\ell_0 = \ell(T_0)$ and $\ell_1 = \ell(T_1)$. Now, $\ell_b > \ell_0 > \ell_1 > \ell_a$. Also, let t_a be the time elapsed in cooling the system from T_0 to T_1, and t_b be the time during the heating from T_1 back to T_0. Therefore, $W_z = t_a\ell_a + t_b\ell_b$ and $W_c = (t_b + t_a)\ell_0$, and we need to prove that

$$(t_b + t_a)\ell_0 > t_a\ell_a + t_b\ell_b$$

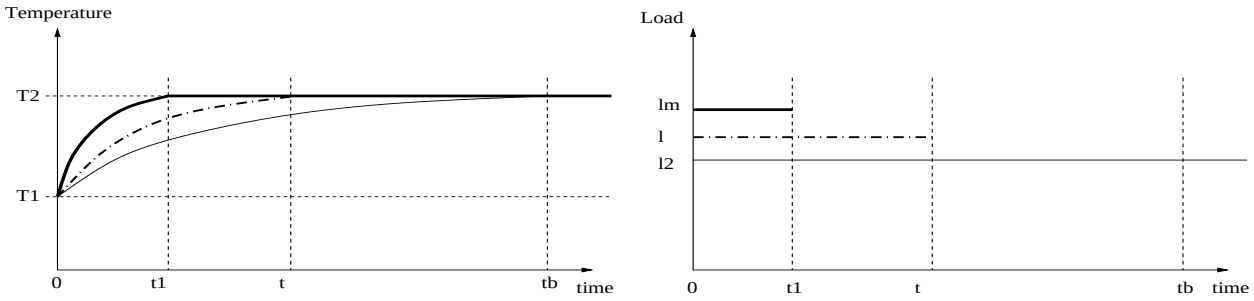

Figure 4: Temperature rise at various loads

Alternatively, we need to show that $t_a(\ell_0 - \ell_a) > t_b(\ell_b - \ell_0)$. Now, from equation 4, we can evaluate

$$t_a = -\frac{1}{\rho} \log \frac{\ell_1^\alpha - \ell_a^\alpha}{\ell_0^\alpha - \ell_a^\alpha}$$
$$t_b = -\frac{1}{\rho} \log \frac{\ell_b^\alpha - \ell_0^\alpha}{\ell_b^\alpha - \ell_1^\alpha}$$

Observe that ℓ_0 and ℓ_1 are fixed data, given T_1 and T_2. Furthermore, given ℓ_0 and ℓ_1, we can consider t_a as a function of ℓ_a and t_b as a function of ℓ_b. In equation 7, if we set $p = \ell_0$ and $q = \ell_1$, we have $t_a(\ell_0 - \ell_a) = \mathcal{F}(\ell_a, \ell_0, \ell_1, \alpha)/\rho$, and $t_b(\ell_b - \ell_0) = \mathcal{F}(\ell_b, \ell_0, \ell_1, \alpha)/\rho$, Therefore, from Lemma 1, since $\ell_b > \ell_0$ and $\ell_a < \ell_1$, we have $t_a(\ell_0 - \ell_a) > t_b(\ell_b - \ell_0)$. For all values of $\ell_b > \ell_0 > \ell_1 > \ell_a$, we have shown that $W_c > W_z$. □

3.3 Aiming for T_{max}

We are now interested in calculating the load that maximizes the net work done while heating the system to attain temperature T_{max}. In Theorem 2, we prove that the amount of work done is maximized for a particular choice of $\ell = \ell^*$, and show how to calculate this value.

Theorem 2. *Let the current temperature be T_1. Assume that we run the system at a load ℓ till the system reaches temperature T_2, and then operate the system such that the temperature is maintained at T_2. Let the net work done (until time infinity) be $W(\ell)$. Then, W_ℓ has a unique maximizer which is obtained by setting $\ell = \ell^*$, where $\ell^* = \min\{1, \max\{\mathcal{F}(\ell, \ell_2, \ell_1, \alpha)\}\}$.*

Proof. Observe that $\ell_1 \leq \ell_2$. We consider load ℓ such that $\ell > \ell_2$. By operating at load ℓ, let the system reach temperature T_2 at time t. For time $t_b > t$, let W be the work done by the time t_b by operating at load ℓ until time t and load ℓ_2 for the remainder. Let W_b be the work done by the time t_b by operating at load ℓ_2.

We illustrate these operating decisions in Figure 4. In the dashed-and-dotted line, we plot the temperature and load of the system as a function of time if we operate at load ℓ until temperature T_2 is reached, and load ℓ_2 subsequently. With the thin solid line, we present the system operating at load ℓ_2 at all times. While the proof does not require this, we also plot the system running at full-blast (load $\ell_m = 1$) until temperature T_{max} is reached using the thick solid line.

To calculate the load that results in maximum work, we need to maximize W. Since W_b does not depend on ℓ, we instead maximize $W - W_b$, since this quantity is easier to characterize. We prove that $W - W_b$ is maximized by ℓ^*. Now, $W = \ell t + \ell_2(t_b - t)$. On the other hand, $W_b = \ell_2 t_b$. Therefore, $W - W_b = t(\ell - \ell_2)$.

From equation 4, we know that time $t = -\frac{1}{\rho} \log \frac{\ell^\alpha - \ell_2^\alpha}{\ell^\alpha - \ell_1^\alpha}$. Therefore, from equation 7

$$W - W_b = \mathcal{F}(\ell, \ell_2, \ell_1, \alpha)/\rho$$

From Lemma 1, we know that this function has a unique maximizer ℓ^* for $\ell > \ell_2$. If this maximizer is greater than 1, then ℓ^* is set to 1, since $\mathcal{F}$ increases monotonically from ℓ_2 to its maximum. □

Next, we prove that this unique maximizer ℓ^* can be calculated efficiently. In particular, this can be evaluated to any accuracy ϵ in time that is a polynomial function of $1/\epsilon$.

Theorem 3. *For any load $\ell_1 \leq \ell_2 \leq 1$ and desired accuracy of solution ϵ, the maximizer $\ell^* = \max_{\ell_2 \leq \ell \leq 1}\{\mathcal{F}(\ell, \ell_2, \ell_1, \alpha)\}\}$ can be calculated to the nearest ϵ in time $\mathcal{O}(\log((1-\ell_2)/\epsilon))$*

Proof. From Lemma 1, we know that $\mathcal{F}$ increases monotonically from ℓ_2 to ℓ^*; in other words there exists a unique maximizer $\hat{\ell}^*$ to $\mathcal{F}(\ell, \ell_2, \ell_1, \alpha)$ for $\ell_2 \leq \ell \leq 1$. This maximizer corresponds to load ℓ where the gradient of $\mathcal{F}$ changes from positive to negative; this can happen at most once for $\ell \geq \ell_2$. As a result, this maximum can be computed efficiently by bisection search on the gradient of $\mathcal{F}$. For desired accuracy ϵ, this can be done in $\mathcal{O}(\log((1-\ell_2)/\epsilon))$ time. □

4 Extensions

In this section, we discuss some examples where system-throttling is the optimal strategy for maximizing work output subject to a temperature threshold.

4.1 Multi-processor systems

Here, we argue that system-throttling continues to dominate other scheduling policies so long as the tasks on different processors can not be moved from one processor to another. In this setting, we assume that each processor has a separate temperature sensor; therefore, the temperature of each processor must not exceed T_{max}. Furthermore, we assume that the loads of each processor can be chosen independent of the load(s) of the other processor(s).

Each processor maximizes the amount of work done by choosing its operating load based on system-throttling. As long as the objective is to maximize the amount of work done, system-throttling is still the best scheduling policy.

4.2 Imperfect cooling

Perfect cooling assumes that the cooling system behaves ideally. This essentially corresponds to three key assumptions made in Section 2 about the parameters in the heat dissipation model. These assumptions characterize the cooling system, and are the following:

1. ρ is a constant. This assumes that parameter ρ does not depend on the temperature, and therefore the rate of heat dissipation is linear in the temperature.

2. T_{amb} is a constant. This assumes that the temperature of the cooling medium does not change.

3. The rate of change of temperature may vary as a delayed function of the temperature; in other words, it may depend on the temperature some t_0 time units ago.

Relaxing these assumption is much harder to incorporate into the heat model, since differential equation 2 is a function of T_{amb} and ρ, which now depend on T. Furthermore, they may depend on historical values of T. (To solve this, we need to model the rate of change of T_{amb} and ρ as a function of temperature and time.)

However, things are not as bad as they seem. If these assumptions are relaxed, this really manifests itself as the power dissipation ability of the cooling system being dependent on whether the system temperature is increasing or decreasing. We consider this state of the system as imperfect cooling. In a

simplified setting, one can think of imperfect cooling as the phenomenon that the system takes longer to cool down than to heat up (all other things equal); this makes intuitive sense.

One way to model this, as a reasonable approximation, is to assume that ρ can be either of two values, ρ_h or ρ_c. Since the system takes longer to cool down than to heat up, imperfect cooling is modeled by setting $\rho_h > \rho_c$. Let T be the current temperature. When the temperature is rising (i.e., when the load $\ell > \ell(T)$), we use ρ_h. Similarly, when the temperature is dropping ($\ell < \ell(T)$), we use ρ_c. This model has been used in the literature; in [4], the authors show that such a model is accurate by comparing estimated and measured temperature.

We now prove that system-throttling is optimal even if cooling is imperfect. The argument is as follows: If system cooling occurs at a faster rate than system heating, then any Zig-zag policy under imperfect cooling will spend longer periods of time in the cooling phase; in Figure 2, the cooling phase would be longer for imperfect cooling. As a result, the time-average work done during a cycle (cooling followed by heating) is lesser for imperfect cooling. Therefore, Zig-zag policies continue to be dominated by Constant policies when there is imperfect cooling.

5 When is system-throttling not enough?

In Section 3, we argued that system-throttling is the best operating rule under a variety of assumptions. Naturally, this is not always true. In this section, we list each of our assumptions, and discuss the consequences of relaxing them. We remark that these scenarios do not exclude the possibility that system-throttling is optimal for particular problem instances; but that a carefully constructed scheduling policy may outperform system-throttling.

5.1 Forced Zig-zag

In most practical implementations, the speed of the processor is not continuously variable; it can operate only at certain states. In this case, one if forced to Zig-zag in an attempt to simulate the best operating load. Some of our work in this direction is presented in [19]. In this paper, we consider settings where the processor is forced to Zig-zag due to design constraints. We use our analysis to describe the best Zig-zag algorithm, and present computational results that illustrate the effectiveness of our algorithm.

5.2 Non-homogeneous jobs

In many scenarios, the tasks have different performance profiles, in the sense that they have different processor, memory and I/O requirements. The power dissipated by the system is no longer just the power dissipated by the processor.

In such settings, it is possible to mix and match tasks with differing characteristics to reduce net system power utilization. In [12], the authors incorporate such ideas to develop efficient an efficient algorithm for CPU power management. In [21], the authors implement an algorithm that utilizes the characteristics of the active jobs to scale the processor whenever it switches threads.

5.3 Movable jobs

In many multi-processor systems, it is possible to move jobs among the different processors. Thus, based on the temperature of the various processors, a scheduling algorithm can outperform system throttling by moving jobs between hot and cold processors. Furthermore, if the jobs are non-homogeneous, performance can be further improved by utilizing the performance profiles of the jobs. In [15], the authors assume system-throttling, and attempt to avoid scenarios where throttling may occur, either by scaling the load, or by swapping jobs. They show that the benefits obtained outweigh the overhead of moving

jobs around. This concept, of moving jobs between processors based on the temperature, can be thought of as a variation on the Heat-and-Run scheduling policy, first developed in [11].

5.4 Conserving energy

In some settings, the net energy available is limited. This is more relevant to mobile computing, especially hand-held devices and laptops. Here, one is more concerned at optimizing the metric (work, make-span, etc.) subject to a net energy availability and the maximum system temperature. In this framework, if we have limited energy, it may make sense to run the processor at a lower load to conserve energy. Not surprisingly, this problem is also referred to in literature as the laptop problem. It is important to re-iterate that energy-constrained problems are structurally very different from temperature-constrained problems, and require very different solution strategies.

The laptop problem has studied in a theoretical setting, where the goal is to develop approximation algorithms [9, 2]. Recently, there has been some theoretical results which also consider the system temperature [1]; however, most of the papers that we are aware of consider only the energy constraint, and not both temperature and energy thresholds.

In the practical setting, the problem has been widely studied, and many algorithms have been proposed which extend the battery life while still satisfying various performance criteria [13, 10].

5.5 Jobs with differing importances

In the preceding sections of this paper, we have assumed that the goal is to maximize the amount of work done over time. However, there are some scenarios where this is not the case.

The most common example where this is the case is when the jobs are heterogeneous with respect their importance. For instance, one may wish to minimize weighted response time of various tasks in the system. In such a setting, it might make sense to run at a lower load during low-weight jobs such that the system can operate at a higher load when high-weight jobs arrive [19].

5.6 Temperature-aware computing centers

In this paper, we have focused mainly on temperature management of single-processor systems, and discussed multi-processor systems briefly. We do not consider the effect of processing on the temperature of computing centers as a whole, which has been widely studied by a variety of authors (see [5] for an extensive survey). Some of these strategies minimize global power consumption in an effort to reduce the cooling requirements [6, 18]. Other, more recent papers incorporate a temperature-aware workload placement policy to increase energy savings [16].

6 Conclusions

In this paper, we argued that simple system-throttling rules are often very effective scheduling policies. This theoretical justification for the implementation of system-throttling as an effective (often optimal) first-pass algorithm is the key contribution of this paper.

To illustrate, we considered a single-processor system where the goal is to maximize the amount of work done, while ensuring that that the temperature constraint is not violated. We proved that the simple system-throttling strategy outperforms any other scheduling policy. Even though we illustrate our arguments in a particular setting for a particular objective, the ideas presented here (in Section 3) were shown to be applicable to a variety of scenarios.

However, our assumptions may not always hold. We also consider the effect when each of our assumptions may not be true, discuss the kind of scheduling algorithms that perform better than system-throttling in these settings, and list some related literature. In the future, we would like to

consider other scenarios where system-throttling is not sufficient, and develop scheduling algorithms that minimize other objectives subject to a temperature threshold. In such scenarios, it would also be interesting to calculate some worst-case performance metrics for system-throttling and compare it against the best Zig-zag algorithm.

References

[1] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In V. Diekert and B. Durand, editors, *STACS*, volume 3404 of *Lecture Notes in Computer Science*, pages 460–471. Springer, 2005.

[2] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *SODA*, 2007.

[3] C. Belady. Cooling and power consideration for semiconductors into the next century. In E. Macii, V. De, and M. J. Irwin, editors, *ISLPED*, pages 100–105. ACM, 2001.

[4] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03)*, New Orleans, LA, Sept. 27 2003.

[5] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, 37:68–74, 2004.

[6] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload-based power management for parallel computer systems. *IBM J. Res. Dev.*, 47(5-6):703–718, 2003.

[7] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *HPCA*, 2001.

[8] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[9] D. P. Bunde. Power-aware scheduling for makespan and flow. In P. B. Gibbons and U. Vishkin, editors, *SPAA*, pages 190–196. ACM, 2006.

[10] K. Flautner and T. Mudge. Vertigo: Automatic performance-setting for linux. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 105–116. ACM Press, 2002.

[11] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In *ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems*, pages 260–270. ACM Press, 2004.

[12] C. Hsu and W. Feng. Reducing overheating-induced failures via performance-aware CPU power management. In *The 6th International Conference on Linux Clusters: The HPC Revolution 2005*, April 2005.

[13] J. Lorch and A. J. Smith. PACE: A new approach to dynamic voltage scaling. *IEEE Transactions on Computers*, 53(7):856–869, 2004.

[14] M. Ma, S. H. Gunther, B. Greiner, N. Wolff, C. Deutschle, and T. Arabi. Enhanced thermal management for future processors. In *2003 Symposium on VLSI Circuits*, pages 201–204, 2003.

[15] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. In *First Association for Computing Machinery (ACM) SIGOPS EuroSys Conference*, pages 403–414. ACM Press, Apr. 18–21 2006.

[16] J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma. Making scheduling "Cool": Temperature-aware workload placement in data centers. In *USENIX Annual Technical Conference, General Track*, pages 61–75. USENIX, 2005.

[17] A. D. Polyanin and V. F. Zaitsev. *Handbook of Exact Solutions for Ordinary Differential Equations*. Chapman & Hall/CRC Press, 2003.

[18] K. Rajamani and C. Lefurgy. On evaluating request-distribution schemes for saving energy in server clusters. In *ISPASS*, pages 111–122. IEEE Computer Society, 2003.

[19] D. Rajan and P. S. Yu. On temperature-aware scheduling for single-processor systems. Submitted to HIPC, 2007.

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

[21] A. Weissel and F. Bellosa. Process cruise control: Event-driven clock scaling for dynamic power management. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES 2002)*, Grenoble, France, Oct. 8–11 2002.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *FOCS '95: Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, page 374, Washington, DC, USA, 1995. IEEE Computer Society.