Online Speed Scaling Based on Active Job Count to Minimize Flow plus Energy

Tak-Wah Lam[*] Lap-Kei Lee[†] Isaac K. K. To[‡] Prudence W. H. Wong[‡]

1 Introduction

This paper is concerned with online scheduling algorithms that aim at minimizing the total flow time plus energy usage. The results are divided into two parts. First, we consider the well-studied "simple" speed scaling model and show how to analyze a speed scaling algorithm (called AJC) that changes speed discretely. This is in contrast to the previous algorithms which change the speed continuously [4, 6]. More interestingly, AJC admits a better competitive ratio, and without using extra speed. In the second part, we extend the study to a more general speed scaling model where the processor can enter a sleep state to further save energy. A new sleep management algorithm called IdleLonger is presented. This algorithm, when coupled with AJC, gives the first competitive algorithm for minimizing total flow time plus energy in the general model.

1.1 Speed scaling, flow and energy

Energy usage has become a major issue in the design of microprocessors, especially for battery-operated devices. Many modern processors support dynamic speed scaling to reduce energy usage. Recent research on online job scheduling has gradually taken speed scaling and energy usage into consideration (see [14] for a survey). The challenge arises from the conflicting objectives of providing good quality of service and conserving energy. Among others, the study of minimizing flow time plus energy has attracted much attention [1, 4–6, 10, 12, 18, 19]. The results to date are based on a speed scaling model in which a processor can vary its speed dynamically, and when running at speed s, consumes energy at the rate of s^α, where α is typically 2 [20] or 3 (the cube-root rule [8]). Most results assume the *infinite speed model* [22] where the processor speed can be scaled arbitrarily high; some consider the more realistic *bounded speed model* [9], which imposes a maximum processor speed T.

Total flow time is a commonly used QoS measure for job scheduling. The flow time (or simply flow) of a job is the time elapsed from when the job arrives until it is completed. In the online setting, jobs with arbitrary sizes arrive at unpredictable times. They are to be scheduled on a processor which allows preemption without penalty. To understand the tradeoff between flow and

[*]Department of Computer Science, University of Hong Kong, Hong Kong. Email: twlam@cs.hku.hk; T.W. Lam is partially supported by HKU Grant 201007176149.

[†]MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation), Department of Computer Science, Aarhus University, Denmark. Email: lklee@madalgo.au.dk.

[‡]Department of Computer Science, University of Liverpool, UK. Email: {isaacto, pwong}@liverpool.ac.uk; I.K.K. To and P.W.H. Wong are partially supported by EPSRC Grant EP/E028276/1.

[0]The results in this paper have appeared in preliminary form in the proceedings of the 16th Annual European Symposium on Algorithms, 2008 and the proceedings of the 36th International Colloquium on Automata, Languages and Programming, 2009.

energy, Albers and Fujiwara [1] initiated the study of minimizing a linear combination of total flow and total energy. The intuition is that, from an economic viewpoint, both flow and energy can be compared with respect to their monetary value, and one can assume that users are willing to pay a certain (say, ρ) units of energy to reduce one unit of flow time. Thus, one would like to have a schedule that minimizes the total flow plus the total energy weighted with ρ. Furthermore, by changing the units of time and energy, one can further assume that $\rho = 1$.

Under the infinite speed model, Albers and Fujiwara [1] considered jobs of unit size. They proposed a speed scaling algorithm that changes the speed according to the number of active jobs which are jobs released but not yet completed, but they did not analyze it. Instead they considered a batched variation (in which newly arrived jobs will be ignored until all jobs of the current batch are completed) and showed that it is $8.3e(1+\Phi)^\alpha$-competitive for minimizing flow plus energy, where $\Phi = (1+\sqrt{5})/2$ is the Golden Ratio. Bansal, Pruhs and Stein [6] later proved that the more natural algorithm proposed in [1] is indeed 4-competitive for unit-sized jobs. More interestingly, [6] gave a new algorithm to handle jobs of arbitrary sizes, which scales the speed as a function of the total remaining work of active jobs and selects the job with the smallest (original) size to run. This algorithm, referred to as the BPS algorithm below, is $O((\frac{\alpha}{\ln \alpha})^2)$-competitive for minimizing flow plus energy; precisely, the competitive ratio is $\mu_\epsilon \gamma_1$, where ϵ is any positive constant, $\mu_\epsilon = \max\{(1+1/\epsilon), (1+\epsilon)^\alpha\}$ and $\gamma_1 = \max\{2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\}$. E.g., if $\alpha = 2$, the competitive ratio is 5.236.

Bansal et al. [4] later adapted the BPS algorithm to the bounded speed model, where a processor has a maximum processor speed T. The competitive ratio remains $O((\frac{\alpha}{\ln \alpha})^2)$ if the maximum speed is relaxed slightly; precisely, assuming that the BPS algorithm can vary its speed between 0 and $(1+\epsilon)T$, where $\epsilon > 0$, then the competitive ratio increases slightly to $\mu_\epsilon \gamma_2$, where $\gamma_2 = 2\alpha/(\alpha - (\alpha-1)^{1-1/(\alpha-1)})$. E.g., if $\alpha = 2$, the competitive ratio is 10.472 using maximum speed $1.618T$. Both results [4, 6] also hold for weighted flow time plus energy.

The BPS algorithm allows us to understand the tradeoff between flow and energy when scheduling jobs of arbitrary size, and it also gives rise to some interesting questions. First, the speed function of the BPS algorithm depends on the remaining work of active jobs, and a work-based speed function would demand the processor to change the speed continuously, which may not be desirable practically. Note the a speed function depending on the count of active jobs would change in a discrete manner and is stabler.

Second, the BPS algorithm requires the online algorithm to have extra speed. In contrast, the classic result on flow time scheduling is that SRPT (shortest remaining processing time) gives minimum flow time without requiring extra speed [3]. This is perhaps due to fact that sometimes BPS would be too slow. Precisely, we observe that the speed of BPS can be lower than the critical threshold $(\frac{n}{\alpha-1})^{1/\alpha}$, where n is the number of active jobs; whenever this happens, one can decrease the flow plus energy by increasing the speed. To see why this is the case, suppose a job J with $p(J)$ units of work is processed to completion using speed s, the energy usage is $P(s)p(J)/s$ and flow time accumulated during its execution is $n\,p(J)/s$, where $P(s) = s^\alpha$. The sum of energy and flow time is minimized if $P(s) + n = s \times P'(s)$, i.e., $s = (\frac{n}{\alpha-1})^{1/\alpha}$. It is sensible to ask whether a speed function that never goes below the critical threshold can work without extra speed and give a better competitive ratio.

In this paper, we consider an online algorithm AJC which selects jobs according to SRPT and sets its speed based on the number of active jobs (like the one proposed in [1] for unit-sized jobs). Thus its speed changes only at job arrival or completion. We show that AJC works for jobs with arbitrary sizes and that AJC is more competitive for minimizing flow plus energy than the

		$\alpha = 2$	$\alpha = 3$
Without sleep state	Infinite speed model ($T = \infty$)	5.236 [6] **2.667** [this paper]	7.940 [6] **3.252** [this paper]
	Bounded speed model	10.472 with max speed $1.618T$ [4] **3.6** with max speed T [this paper]	11.910 with max speed $1.466T$ [4] **4** with max speed T [this paper]
With multiple sleep states	Infinite speed model ($T = \infty$)	**7.333** [this paper]	**8.503** [this paper]
	Bounded speed model	**11** with max speed T [this paper]	**12** with max speed T [this paper]

Table 1: Results on scheduling with and without sleep states for minimizing flow time plus energy. Note that the new results in this paper do not demand extra speed.

BPS algorithm [4, 6]. More importantly, AJC does not demand extra processor speed. For the infinite and bounded speed model, the competitive ratios are respectively $\beta_1 = 2/(1 - \frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$ and $\beta_2 = 2(\alpha+1)/(\alpha - \frac{\alpha-1}{(\alpha+1)^{1/(\alpha-1)}})$. Table 1 compares these ratios with those in [4, 6]. The improvement is more significant for large α, as β_1 and β_2 are approximately $2\alpha/\ln\alpha$, while $\mu_\epsilon\gamma_1$ and $\mu_\epsilon\gamma_2$ [4, 6] are approximately $2(\alpha/\ln\alpha)^2$.

Technically speaking, the results of AJC stem from a more direct analysis. Previous analysis of the BPS algorithm [4, 6] makes use of potential functions that are based on the fractional amount of unfinished work, which lead to good upper bounds on an intermediate notion called fractional flow which varies continuously with time. Extra speed is needed when transforming the result on fractional flow to (integral) flow. A technical contribution of this paper is a different potential function, which is based on the (integral) number of unfinished jobs. It allows us to directly compare the flow of the online algorithm against an optimal offline algorithm, and to obtain a tighter analysis.

1.2 Sleep management plus speed scaling

In earlier days when the speed scaling technology was not available, energy reduction was mostly achieved by allowing a processor to enter a low-power *sleep* state, (e.g., an Intel Xeon E5320 server requires 240W when working, 150W when idling and 10W and 0W for two low-power sleep states [11]) yet waking up requires extra energy [15]. In the (embedded systems) literature, there are different energy-efficient strategies to determine when to bring a processor to sleep during a period of zero load [7]. This is an online problem, usually referred to as *dynamic power management*. The input is the length of a zero-load period, known only when the period ends. There are several interesting results with competitive analysis (e.g., [2, 15, 17]). In its simplest form, the problem assumes the processor is in either the *awake* state or the *sleep* state. The awake state always requires a static power $\sigma > 0$. To have zero energy usage, the processor must enter the sleep state, but a wake-up back to the awake state requires $\omega > 0$ energy. In general, there can be multiple intermediate sleep states, which demand some static power but less wake-up energy.

It is natural to study job scheduling on a processor that allows both sleep states and speed scaling. More specifically, a processor in the awake state can run at any speed $s \geq 0$ and consumes energy at the rate $s^\alpha + \sigma$, where $\sigma > 0$ is the static power and s^α is the dynamic power[1]. In this model, job scheduling requires two components: a *sleep management algorithm* to determine

[1]Static power is dissipated due to leakage current and is independent of processor speed, and dynamic power is due to dynamic switching loss and increases with the speed.

when to sleep or work, and a *speed scaling algorithm* to determine which job and at what speed to run. Notice that sleep management here is not the same as in dynamic power management; in particular, the length of a sleep or idle period is part of the optimization (rather than the input), and we care about both flow and energy. Technically, adding a sleep state changes the nature of speed scaling. If there is no sleep state, running a job slower is a natural way to save energy. Now one can also save energy by sleeping more and working faster later. It is even more complicated when flow is concerned. Prolonging a sleeping period by delaying job execution can save energy, yet it also incurs extra flow. Striking a balance is not trivial.

In the theory literature, the only relevant work is by Irani et al. [16]; they studied deadline scheduling on a processor with one sleep state and infinite speed scaling. They showed an $O(1)$-competitive algorithm to minimize the energy for meeting the deadlines of all jobs. Their idea is to delay execution of jobs when the processor is in the sleep state and compensate for such delay by using extra speed on top of that recommended by a speed scaling algorithm. This approach, however, is difficult to use in the bounded speed model because we cannot increase the speed above T. More importantly, the idea is not applicable to flow-energy scheduling as we need new ideas to tackle the dilemma over delaying job execution while saving energy incurs extra flow time. In the context of flow-energy scheduling, it has been open how to design a sleep management algorithm and a speed scaling algorithm to achieve $O(1)$-competitiveness.

In this paper, we initiate the study of flow-energy scheduling that exploits both speed scaling and multiple sleep states. We give a sleep management algorithm called IdleLonger, which works for a processor with one or more levels of sleep states. Under the infinite speed model, the speed scaling algorithm AJC together with IdleLonger is shown to be $O(\frac{\alpha}{\ln\alpha})$-competitive for minimizing flow plus energy (precisely, the ratio is $2\beta_1 + 2 = O(\frac{\alpha}{\ln\alpha})$). For the bounded speed model, the problem becomes more difficult because the processor, once overslept, cannot rely on unlimited extra speed to catch up the delay. Nevertheless, we are able to make IdleLonger more conservative to observe the maximum processor speed, and IdleLonger remains $O(\frac{\alpha}{\ln\alpha})$-competitive under the bounded speed model (precisely, the ratio is $\max\{\beta_2(2+1/\alpha)+\zeta+1, 3\beta_2+3\}$, where $\zeta = \max\{4, \beta_2(1-1/\alpha)\}$). Table 1 shows the competitive ratios when $\alpha = 2$ and 3.

The design of IdleLonger is interesting. When the processor is sleeping, it is natural to delay waking up until sufficient jobs have arrived. The non-trivial case is when the processor is idle (i.e., awake but at zero speed), IdleLonger has to determine when to start working again or go to sleep. If there are no new jobs arriving and we only need to determine when the processor goes to sleep, it is like a continuous version of the ski rental problem and it makes sense to sleep if the idle energy reaches the wakeup cost. A dilemma arises when there are new jobs arriving. At first glance, if some new jobs arrive while the processor is idle, the processor should run the jobs immediately so as to avoid extra flow. Yet this would allow the adversary to easily keep the processor awake, and it is difficult to achieve $O(1)$-competitiveness. During an idle period, IdleLonger considers static energy consumed (due to static power) and flow accumulated as two competing quantities. Only if the flow exceeds the energy does IdleLonger start to work. Otherwise, IdleLonger will remain idle until the energy reaches to a certain level; then the processor goes to sleep even in the presence of jobs. The latter is perhaps counter-intuitive.

Apparently, a sleep management algorithm and a speed scaling algorithm would affect each other; analyzing their relationship and their total cost could be a complicated task. Interestingly, the results of this paper stem from the fact that we can isolate the analysis of these algorithms. We divide the total cost (flow plus energy) into two parts, *working cost* (incurred while working on jobs) and *inactive cost* (incurred at other times). We upper bound the inactive cost of IdleLonger independently of the speed scaling algorithm.

Although the working cost does depend on both the speed scaling and sleep management algorithms, our potential analysis of the speed scaling algorithm reveals that the dependency on the sleep management algorithm is limited to a simple quantity called *inactive flow*, which is the flow part of the inactive cost. Intuitively, large inactive flow means many jobs are delayed due to prolonged sleep, and hence the processor has to work faster later to catch up, incurring a higher working cost. It is easy to minimize inactive flow at the sacrifice of the energy part of the inactive cost. IdleLonger is designed to maintain a good balance between them. In conclusion, coupling IdleLonger with AJC, we obtain competitive algorithms for flow plus energy.

Organization of the paper. Section 2 defines the model formally. Sections 3 and 4 focus on the infinite speed model and discuss the speed scaling algorithm AJC and the sleep management algorithm IdleLonger, respectively. Section 5 presents our results in the bounded speed model. Finally, we conclude with discussion in Section 6.

2 Preliminaries

The input is a sequence of jobs arriving online. We denote the release time and work requirement (or size) of a job J as $r(J)$ and $p(J)$, respectively. All jobs are to be executed on a single processor, where preemption is allowed, and a preempted job can resume at the point of preemption.

Speed, static and dynamic power. We first consider the speed scaling model that allows one sleep state. At any time, the processor is in either the *awake* state or the *sleep* state. In the former, the processor can run at any speed $s \geq 0$ and demands power in the form $s^\alpha + \sigma$, where $\alpha > 1$ and $\sigma \geq 0$ are constants. We call s^α the dynamic power and σ the static power. In the sleep state, the speed is zero and the power is zero. State transition requires energy; without loss of generality, we assume a transition from the sleep state to the awake state requires an amount ω of energy, and the reverse takes zero energy. To simplify our study, we follow the previous work [16] to assume state transition takes no time. Initially, the processor is in the sleep state. It is useful to differentiate two types of awake state, namely, with zero speed and with positive speed. The former is called the *awake-idle* or simply *idle* state and the latter is called *awake-working* or simply *working* state.

Next we generalize the above setting to $m > 1$ levels of sleep. A processor is in either the *awake* state or a *sleep-i* state, where $1 \leq i \leq m$. The awake state is the same as before, demanding static power σ and dynamic power s^α. For convenience, we let $\sigma_0 = \sigma$. The sleep-m state is the only "real" sleep state, which has static power $\sigma_m = 0$; other sleep-i states have decreasing positive static power σ_i such that $\sigma = \sigma_0 > \sigma_1 > \sigma_2 > \cdots > \sigma_{m-1} > \sigma_m = 0$. We denote the wake-up energy from the sleep-i state to the awake state as ω_i. Note that $\omega_m > \omega_{m-1} > \cdots > \omega_1 > 0$. Initially, the processor is in the sleep-m state. Note that this model of having multiple sleep states is similar to that in [2].

In the literature most of the speed scaling results are based on a model with no sleep state, which we will refer to as the simple speed scaling model. The speed scaling model with sleep states will be referred to as the general speed scaling model. The simple speed scaling model is a special case of the general speed scaling model, where the static power σ and the wake-up energy ω are both zero. In other words, the power consumption is modeled as s^α. Any speed scaling algorithm for the general model can be carried to the simple model.

Flow, energy, inactive and working cost. Consider any schedule of jobs. At any time t, for any job J, we let $q(J)$ denote the remaining work of J, and J is said to be *active* if $r(J) \leq t$ and $q(J) > 0$. The flow $F(J)$ of a job J is the time elapsed since it arrives and until it is completed.

The total flow is $F = \sum_J F(J)$. Note that $F = \int_0^\infty n(t)\ \mathrm{d}t$, where $n(t)$ is the number of active jobs at time t. Based on this view, we divide F into two parts: F_W is the *working* flow incurred during all the times in the working state, and F_I is the *inactive* flow incurred during all the times in the idle or sleep state. The energy usage is also divided into three parts: W denotes the energy due to wake-up transitions, E_I is the idling energy (static power consumption in the idle or intermediate sleep states), and E_W is the working energy (static and dynamic power consumption in the working state). Our objective is to minimize the *total cost* $G = F_\mathrm{W} + F_\mathrm{I} + E_\mathrm{I} + E_\mathrm{W} + W$. It is useful to define the *working cost* $G_\mathrm{W} = F_\mathrm{W} + E_\mathrm{W}$, and the *inactive cost* $G_\mathrm{I} = F_\mathrm{I} + E_\mathrm{I} + W$.

Below we use OPT to denote the optimal offline algorithm, and we let G^* denote OPT's total cost, and W^* its total wake-up energy. For the purpose of analysis, we also define $C^* = G^* - W^*$.

SRPT. It is well known that SRPT (shortest remaining work) is optimal for classic flow time optimization. SRPT is still optimal when speed scaling is allowed and sleep states are considered. We provide a proof (Lemma 1) for the sake of completeness. With this, we assume that, without loss of generality, OPT uses the SRPT policy for job selection.

Lemma 1. *To minimize flow plus energy, the optimal schedule selects jobs in accordance with SRPT whenever it works on jobs.*

Proof. Consider a job sequence $\mathcal{J}$. Suppose there is an optimal schedule S for $\mathcal{J}$ and S does not follow SRPT. We modify S in multiple steps to a schedule which uses SRPT for job selection. In each step the new schedule S' has total flow time reduced with energy usage preserved. Then the lemma follows.

Let t be the first time when S does not follow SRPT, running job J_ℓ instead of job J_s with the shortest remaining work. S' differs from S during the time intervals after t when S runs either job: J_s is run to completion before J_ℓ, using the same speed and state transitions at any time; this guarantees that energy usage is preserved. J_s thus completes in S' earlier than J_ℓ does in S. Therefore, the sum of their completion time and hence the total flow time in S' are less than that in S. □

3 Speed Scaling based on Active Job Count

This section gives the details of the speed scaling algorithm AJC (active job count). The discussion is based on the general speed scaling model with $m \geq 1$ sleep states. We analyze the working cost G_W of AJC when coupled with an arbitrary sleep management algorithm. Note that a sleep management algorithm determines when the processor should sleep, idle, and work, while AJC specifies which job and at what speed the processor should run when the processor is working.

Under the simple speed scaling model (which assumes no sleep states and zero static power), the result of this section already yields a competitive algorithm for minimizing flow plus energy. To this end, we consider AJC coupled with the trivial sleep management algorithm which always keeps the processor in the working state whenever there are active jobs. Then the inactive cost is always zero, and the working cost of AJC is equal to the total cost. More details will be given at the end of this section.

To obtain a competitive result for the general speed scaling model, we also need a sleep management algorithm to work with AJC. In Section 4, we present a sleep management algorithm called IdleLonger and derive an upper bound of its inactive cost.

3.1 AJC and analysis of working cost

Let $n(t)$ be the number of active jobs at time t. Suppose that there is no sleep state and the static power σ is zero. Running at the speed $n(t)^{1/\alpha}$ can maintain a good balance between flow and energy because, at any time t, both the flow and the energy are incurred at the same rate of $n(t)$. However, when the static power σ is bigger than zero, the balance cannot be maintained. In general, if σ is large, running at speed $n(t)^{1/\alpha}$ would be too slow to be cost effective as the dynamic power could be way smaller than σ. In fact, one can show that running at this speed has unbounded competitive ratio no matter what sleep management algorithm is used. This motivates us to take σ into the design of the speed function of AJC.

> **Algorithm AJC.** At any time t, AJC runs the active job with the shortest remaining work (SRPT) at the speed $(n(t) + \sigma)^{1/\alpha}$.

We remark that in Section 5, when the speed is bounded, we cap the speed at T, i.e., at any time t, the processor runs at the speed $\min\{(n(t) + \sigma)^{1/\alpha}, T\}$. To assist understanding in later sections, in the analysis below, we state when the analysis follows directly in the bounded speed model and when further arguments are needed, with details given in Section 5.

As mentioned earlier, the analysis of AJC will be done under the general speed scaling model with sleep states, and the following argument of AJC's working cost is valid no matter what sleep management algorithm, denoted Slp, is being used together with AJC. We use Slp(AJC) to denote the resulting algorithm of AJC coupled with Slp.

Ideally we want to upper bound the working cost of Slp(AJC) solely by the total cost of the optimal offline schedule OPT, yet this is not possible as the working cost also depends on the sleeping strategy Slp. Intuitively, if Slp prefers to sleep even when there are many active jobs, the working cost would be very high. Below we give an analysis of the working cost in which the dependency on Slp is bounded by the *inactive flow* F_{I} incurred by Slp(AJC). Recall that the inactive flow refers to the flow incurred while the processor is in the idle or sleep state. The main result of this section is a potential analysis showing that the working cost of Slp(AJC), denoted G_{W}, is $O(C^* + F_{\text{I}})$, where C^* is the total cost of OPT minus its wake-up energy (i.e., $C^* = G^* - W^*$).

Theorem 2. *With respect to* Slp(AJC)*,* $G_{\text{W}} \leq \beta C^* + (\beta - 2)F_{\text{I}}$*, where* $\beta = 2/(1 - \frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$*.*

The rest of this section is devoted to proving Theorem 2. Before showing the technical details, it is useful to have a digest of the relationship between G_{W}, C^* and F_{I} as stipulated in Theorem 2. Suppose that Slp always switches to working state whenever there are active jobs, then $F_{\text{I}} = 0$. In this case, Theorem 2 implies that $G_{\text{W}} = O(C^*)$. However, the inactive cost of Slp and the total cost of Slp(AJC) may be unbounded. In Section 4, we will present a sleep management algorithm IdleLonger that prefers to wait for more jobs before waking up to work. Then AJC would start at a higher speed and G_{W} can be much larger than C^*. The excess is due to the fact that sometimes the online algorithm is sleeping while OPT is working. Nevertheless, Theorem 2 states that the excess can be upper bounded by $O(F_{\text{I}})$ (intuitively, the cost to catch up is at a rate depending on $n(t)$). In Section 4, we will show that IdleLonger has nice upper bounds on the inactive flow F_{I} and the inactive cost G_{I}. This leads to the result that IdleLonger(AJC) has a competitive total cost.

We are ready to look into the technical details. Our analysis exploits amortization and potential functions (e.g., [6, 9]). As mentioned in [6], a common technique to analyze the performance of an online algorithm is to show that the algorithm is locally competitive, i.e., to show that the algorithm is competitive at all times. Yet it is not always possible to achieve local competitiveness, in which case, we turn to amortization and make use of potential functions. Roughly speaking,

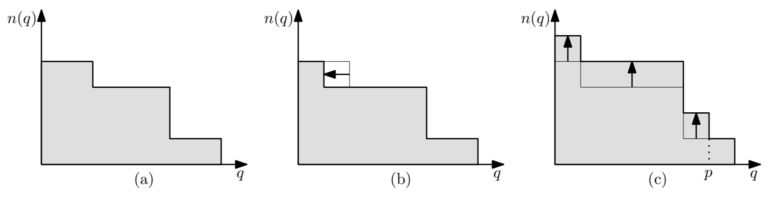

Figure 1: (a) At any time, $n_{\mathrm{a}}(q)$ or $n_{\mathrm{o}}(q)$ (denoted by $n(q)$ above) is a step function containing unit height stripes, and the area under $n(q)$ is the total remaining work. (b) If we run the job with the smallest remaining processing time at speed s for a period of time Δ, the top stripe shrinks by $s\Delta$. (c) When a job of size p is released, $n(q)$ increases by 1 for all $q \leq p$.

a potential function attempts to capture some accumulated difference between the online and the optimal algorithms, which allows one to show that the cost of the online algorithm at a certain time together with the accumulated difference is competitive against the cost of the optimal algorithm.

Consider any time t. Let $G_{\mathrm{W}}(t)$, $F_{\mathrm{I}}(t)$ and $C^*(t)$ denote the corresponding value of G_{W}, F_{I} and C^* incurred up to t. We drop the parameter t when it is clear that t is the current time. We will define a potential function $\Phi(t)$, which is a function of time satisfying the following conditions: (i) Φ is zero initially and finally; (ii) Φ is a continuous function except at some discrete times (when a job arrives, or AJC or OPT finishes a job or changes speed), where Φ can never increase; and more importantly, (iii) at any other time, the rate of change of $G_{\mathrm{W}}(t)$ and $\Phi(t)$ can be upper bounded as

$$\frac{\mathrm{d}G_{\mathrm{W}}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\Phi(t)}{\mathrm{d}t} \leq \beta \frac{\mathrm{d}C^*(t)}{\mathrm{d}t} + (\beta - 2)\frac{\mathrm{d}F_{\mathrm{I}}(t)}{\mathrm{d}t}, \tag{1}$$

where β is a constant. Condition (iii) is also called the running condition. By integrating the running condition over all times, we can conclude that $G_{\mathrm{W}} + \mathit{net}\text{-}\Phi \leq \beta C^* + (\beta-2)F_{\mathrm{I}}$, where $\mathit{net}\text{-}\Phi$ is the net change of Φ excluding those discrete changes occurring at times specified in Condition (ii). Note that Conditions (i) and (ii) together guarantee that $\mathit{net}\text{-}\Phi$ cannot be negative. It follows that $G_{\mathrm{W}} \leq \beta C^* + (\beta - 2)F_{\mathrm{I}}$.

3.2 Potential analysis

Potential function $\boldsymbol{\Phi(t)}$. The potential function Φ attempts to capture the difference of the remaining work of Slp(AJC) and OPT. The formal definition is as follows. Consider any time t. For any $q \geq 0$, let $n_{\mathrm{a}}(q)$ be the current number of active jobs of Slp(AJC) with remaining work at least q, and similarly $n_{\mathrm{o}}(q)$ for OPT. It is useful to consider $n_{\mathrm{a}}(q)$ and $n_{\mathrm{o}}(q)$ as functions of q. These two functions would change over time, e.g., when a new job arrives or after a job has run for some time (see Figure 1). Let $\beta > 0$ be a constant (to be fixed later). We define the potential function as

$$\Phi(t) = \beta \int_0^\infty \phi(q)\mathrm{d}q \ , \text{ where } \phi(q) = \left(\sum_{j=1}^{n_{\mathrm{a}}(q)} (j+\sigma)^{1-1/\alpha} \right) - (n_{\mathrm{a}}(q)+\sigma)^{1-1/\alpha} n_{\mathrm{o}}(q) \ .$$

Intuitively, $\phi(q)$ and hence $\Phi(t)$ is composed of two parts. The first term of $\phi(q)$ when integrated over all q is proportional to the cost (flow plus energy) required by AJC to complete all the active jobs if the processor keeps working after time t and no more job arrives (precisely, the exact amount is $2\int_0^\infty \sum_{i=1}^{n_{\mathrm{a}}(q)} (i+\sigma)^{1-1/\alpha}\,\mathrm{d}q$). The second term of $\phi(q)$ guarantees that Φ does not increase when a job arrives (Lemma 3).

Boundary conditions and discrete events. By definition, $\Phi(t) = 0$ initially and when Slp(AJC) and OPT both finish all jobs. At any time t, when either AJC or OPT finishes a job or changes the speed, $\phi(q)$ only changes at a discrete point of q, and hence $\Phi(t)$ does not change at all. Lemma 3 below further shows that when a job arrives, $\phi(q)$ cannot increase for any q, and hence $\Phi(t)$ does not increase. As can be shown in the proof, this lemma holds because of the design of the potential function; this potential function is also used in Section 5 for the bounded speed model in which the lemma also holds.

Lemma 3. *When a job arrives, the change of Φ is non-positive.*

Proof. Suppose a job J arrives and $n_{\mathrm{a}}(q)$ and $n_{\mathrm{o}}(q)$ denote the values just before J arrives. For $q > p(J)$, $n_{\mathrm{a}}(q)$ and $n_{\mathrm{o}}(q)$, and hence $\phi(q)$, are unchanged. For $q \le p(J)$, both $n_{\mathrm{a}}(q)$ and $n_{\mathrm{o}}(q)$ increase by 1 (see Figure 1(c)). Thus the first term of $\phi(q)$ increases by $(n_{\mathrm{a}}(q) + 1 + \sigma)^{1-1/\alpha}$. The increase of the second term $(n_{\mathrm{a}}(q) + \sigma)^{1-1/\alpha} n_{\mathrm{o}}(q)$ can be interpreted in two steps: (i) increase to $(n_{\mathrm{a}}(q) + 1 + \sigma)^{1-1/\alpha} n_{\mathrm{o}}(q)$, and (ii) increase to $(n_{\mathrm{a}}(q) + 1 + \sigma)^{1-1/\alpha}(n_{\mathrm{o}}(q) + 1)$. The increase in step (ii), i.e., $(n_{\mathrm{a}}(q) + 1 + \sigma)^{1-1/\alpha}$, covers the increase in the first term of $\phi(q)$, so $\phi(q)$ cannot increase when J arrives. □

The change rate of $\boldsymbol{\Phi}$. At any other time t (when there is no job arrival/completion or speed change), $\Phi(t)$ may increase, but the rate of increase is always within a bound, which would allow us to prove the running condition.

At time t, denote the total number of active jobs of Slp(AJC) as n_{a}, its current speed as s_{a}, and the remaining work of the current job as q_{a}. Similarly, we define n_{o}, s_{o} and q_{o} for OPT. To analyze $\frac{\mathrm{d}\Phi}{\mathrm{d}t}$, we consider how the function $\phi(q)$ changes over an infinitesimal amount of time (from t to $t + \mathrm{d}t$), which would then lead to the change of Φ. Conceptually, $\phi(q)$ changes in two ways: (i) the execution of Slp(AJC) affects $\phi(q)$ for q in $(q_{\mathrm{a}} - s_{\mathrm{a}}\mathrm{d}t, q_{\mathrm{a}}]$; and (ii) the execution of OPT affects $\phi(q)$ for q in $(q_{\mathrm{o}} - s_{\mathrm{o}}\mathrm{d}t, q_{\mathrm{o}}]$. We denote the change of Φ due to these two ways as $\mathrm{d}\Phi_1$ and $\mathrm{d}\Phi_2$, respectively. Note that $\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \frac{\mathrm{d}\Phi_1}{\mathrm{d}t} + \frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$. We first upper bound $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$. Note that the following lemmas hold for any fixed positive value of β.

Lemma 4. *Assume that $\beta > 0$. Then $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}})\frac{s_{\mathrm{a}}}{(n_{\mathrm{a}}+\sigma)^{1/\alpha}}$.*

Proof. Note that $n_{\mathrm{a}}(q_{\mathrm{a}}) = n_{\mathrm{a}}$ and $n_{\mathrm{a}}(q)$ decreases from n_{a} to $n_{\mathrm{a}} - 1$ for all $q \in (q_{\mathrm{a}} - s_{\mathrm{a}}\mathrm{d}t, q_{\mathrm{a}}]$ (see Figure 1(b)). For any $q \in (q_{\mathrm{a}} - s_{\mathrm{a}}\mathrm{d}t, q_{\mathrm{a}}]$, $\phi(q)$ changes by

$$\left(\sum_{i=1}^{n_{\mathrm{a}}-1} (i+\sigma)^{1-1/\alpha}\right) - (n_{\mathrm{a}} - 1 + \sigma)^{1-1/\alpha} n_{\mathrm{o}}(q) - \left(\sum_{i=1}^{n_{\mathrm{a}}} (i+\sigma)^{1-1/\alpha}\right) + (n_{\mathrm{a}} + \sigma)^{1-1/\alpha} n_{\mathrm{o}}(q)$$
$$= -(n_{\mathrm{a}} + \sigma)^{1-1/\alpha} + ((n_{\mathrm{a}} + \sigma)^{1-1/\alpha} - (n_{\mathrm{a}} + \sigma - 1)^{1-1/\alpha}) n_{\mathrm{o}}(q).$$

Note that $x^{1-1/\alpha} - (x-1)^{1-1/\alpha} \le x^{-1/\alpha}$ for all $x \ge 1$. By setting $x = n_{\mathrm{a}} + \sigma$ and the fact that $n_{\mathrm{o}}(q) \le n_{\mathrm{o}}$, $\phi(q)$ changes by at most $-(n_{\mathrm{a}} + \sigma)^{1-1/\alpha} + (n_{\mathrm{a}} + \sigma)^{-1/\alpha} n_{\mathrm{o}}$. Integrating over all $q \in (q_{\mathrm{a}} - s_{\mathrm{a}}\mathrm{d}t, q_{\mathrm{a}}]$, we have

$$\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le \beta(-(n_{\mathrm{a}} + \sigma)^{1-1/\alpha} + (n_{\mathrm{a}} + \sigma)^{-1/\alpha} n_{\mathrm{o}}) s_{\mathrm{a}} = -\beta \frac{s_{\mathrm{a}}}{(n_{\mathrm{a}}+\sigma)^{1/\alpha}} (n_{\mathrm{a}} + \sigma - n_{\mathrm{o}}) \ .$$ □

Corollary 5. *Assume that $\beta > 0$. If $s_{\mathrm{a}} > 0$, $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}})$; if $s_{\mathrm{a}} = 0$, $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le 0$.*

Proof. If $s_{\mathrm{a}} = 0$, then Lemma 4 implies that $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le 0$. If $s_{\mathrm{a}} > 0$, then by the definition of AJC, $s_{\mathrm{a}} = (n_{\mathrm{a}} + \sigma)^{1/\alpha}$, and by Lemma 4 again, $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}})$. □

We note that Lemma 4 continues to hold in the bounded speed model in Section 5 as long as we choose a positive β. However, this is not the case for Corollary 5 since the speed s_a may be capped at T and smaller than $(n_\mathrm{a}+\sigma)^{1/\alpha}$, which is required in the argument of the corollary. In such case, we bound $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$ differently in Lemma 17.

Next, we consider the upper bound of $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$.

Lemma 6. *Assume that $\beta > 0$. Then $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta(n_\mathrm{a}+\sigma)^{1-1/\alpha}s_\mathrm{o}$.*

Proof. From t to $t+\mathrm{d}t$, $\phi(q)$ changes only at those points $q \in (q_\mathrm{o} - s_\mathrm{o}\mathrm{d}t, q_\mathrm{o}]$. Note that $n_\mathrm{a}(q) \le n_\mathrm{a}$ for all q. For any $q \in (q_\mathrm{o} - s_\mathrm{o}\mathrm{d}t, q_\mathrm{o}]$, $\phi(q)$ changes by $(n_\mathrm{a}(q)+\sigma)^{1-1/\alpha} \le (n_\mathrm{a}+\sigma)^{1-1/\alpha}$. Integrating over all $q \in (q_\mathrm{o} - s_\mathrm{o}\mathrm{d}t, q_\mathrm{o}]$, we have $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta(n_\mathrm{a}+\sigma)^{1-1/\alpha}s_\mathrm{o}$. □

Using the Young's inequality [13], we can introduce any constant $\mu > 0$ into the above bound of $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$ as follows.[2]

$$\begin{aligned}\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta((n_\mathrm{a}+\sigma)^{1-1/\alpha}\mu)\left(\frac{s_\mathrm{o}}{\mu}\right) &\le \beta\left(\int_0^{\frac{s_\mathrm{o}}{\mu}} x^{\alpha-1}\,\mathrm{d}x + \int_0^{(n_\mathrm{a}+\sigma)^{1-1/\alpha}\mu} x^{1/(\alpha-1)}\,\mathrm{d}x\right)\\ &\le \frac{\beta s_\mathrm{o}^\alpha}{\alpha\mu^\alpha} + \beta(1-\frac{1}{\alpha})\mu^{\frac{\alpha}{\alpha-1}}(n_\mathrm{a}+\sigma)\ . \end{aligned} \tag{2}$$

In particular, for any $\mu > 0$ and $\beta \ge 2/(1-(1-\frac{1}{\alpha})\mu^{\frac{\alpha}{\alpha-1}})$, we have $\beta(1-\frac{1}{\alpha})\mu^{\alpha/(\alpha-1)} \le \beta - 2$ and Inequality (2) can be simplified to Corollary 7(i). In this section, we set $\beta = 2/(1-\frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$ and $\mu = \alpha^{-1/\alpha}$. Then $\alpha\mu^\alpha = 1$ and $(1-\frac{1}{\alpha})\mu^{\alpha/(\alpha-1)} = 1-2/\beta$. The upper bound of $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$ can be simplified as in Corollary 7(ii) below. In Section 5, we will use a smaller $\beta = 2/(1-\frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ and a bigger $\mu = (\alpha+1)^{-1/\alpha}$. The relation $(1-\frac{1}{\alpha})\mu^{\alpha/(\alpha-1)} = 1-2/\beta$ remains valid, though $\alpha\mu^\alpha = \alpha/(\alpha+1)$. Then we have Corollary 7(iii).

Corollary 7. **(i)** *Assume that $\mu > 0$ and $\beta \ge 2/(1-(1-\frac{1}{\alpha})\mu^{\frac{\alpha}{\alpha-1}})$. Then $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \frac{\beta s_\mathrm{o}^\alpha}{\alpha\mu^\alpha} + (\beta-2)(n_\mathrm{a}+\sigma)$.* **(ii)** *Assume that $\beta = 2/(1-\frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$. Then $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta s_\mathrm{o}^\alpha + (\beta-2)(n_\mathrm{a}+\sigma)$.* **(iii)** *Assume that $\beta = 2/(1-\frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$. Then $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta(1+1/\alpha)s_\mathrm{o}^\alpha + (\beta-2)(n_\mathrm{a}+\sigma)$.*

3.3 Running condition

With Corollaries 5 and 7, we can prove the running condition. Recall that we set $\beta = 2/(1-\frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$, which is at least 2 for any $\alpha \ge 1$.

Lemma 8. *At any time when no discrete events (i.e., a job arrives, or* AJC *or* OPT *finishes a job or changes speed) occur, $\frac{\mathrm{d}G_\mathrm{w}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le \beta\frac{\mathrm{d}C^*}{\mathrm{d}t} + (\beta-2)\frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t}$, where $\beta = 2/(1-\frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$.*

Proof. When Slp(AJC) is working, $\frac{\mathrm{d}G_\mathrm{w}}{\mathrm{d}t} = s_\mathrm{a}^\alpha + \sigma + n_\mathrm{a} = 2(n_\mathrm{a}+\sigma)$ and $\frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t} = 0$; otherwise, $\frac{\mathrm{d}G_\mathrm{w}}{\mathrm{d}t} = 0$ and $\frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t} = n_\mathrm{a}$. When OPT is working, $\frac{\mathrm{d}C^*}{\mathrm{d}t} = s_\mathrm{o}^\alpha + \sigma + n_\mathrm{o}$; otherwise, $\frac{\mathrm{d}C^*}{\mathrm{d}t} \ge n_\mathrm{o}$. Below we give a case analysis depending on whether Slp(AJC) or OPT are working. The interesting case is Case (iii),

[2] Young's Inequality is stated as follows. Let f be a real-valued, continuous and strictly increasing function such that $f(0) = 0$. Then, for all $g, h \ge 0$, $\int_0^g f(x)\,\mathrm{d}x + \int_0^h f^{-1}(x)\,\mathrm{d}x \ge gh$, where f^{-1} is the inverse function of f. When using Young's inequality to adapt the bound of $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$, we set $f(x) = x^{\alpha-1}$, $f^{-1}(x) = x^{1/(\alpha-1)}$, $g = s_\mathrm{o}/\mu$, and $h = (n_\mathrm{a}+\sigma)^{1-1/\alpha}\mu$.

where OPT is working but Slp(AJC) is not. In this case the increase of potential is partially offset by the increase of inactive flow.

Case i. $s_a > 0$, $s_o > 0$: In this case, $\frac{dG_W}{dt} = 2(n_a + \sigma)$, $\frac{dC^*}{dt} = s_o^\alpha + \sigma + n_o$, and $\frac{dF_I}{dt} = 0$. We can use Corollary 5 and Corollary 7(ii) to upper bound $\frac{d\Phi_1}{dt}$ and $\frac{d\Phi_2}{dt}$. It follows that

$$\begin{aligned}\frac{dG_W}{dt} + \frac{d\Phi}{dt} &\le 2(n_a + \sigma) - \beta(n_a + \sigma - n_o) + \beta s_o^\alpha + (\beta - 2)(n_a + \sigma) \\ &= \beta s_o^\alpha + \beta n_o \\ &\le \beta \frac{dC^*}{dt} \ .\end{aligned}$$

Case ii. $s_a > 0$, $s_o = 0$: In this case, $\frac{dG_W}{dt} = 2(n_a + \sigma)$, $\frac{dC^*}{dt} \ge n_o$, $\frac{dF_I}{dt} = 0$, and $\frac{d\Phi_2}{dt} \le 0$ (by Lemma 6). Using Corollary 5 and the fact that $\beta \ge 2$, we have

$$\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le 2(n_a + \sigma) - \beta(n_a + \sigma) + \beta n_o \le \beta n_o \le \beta \frac{dC^*}{dt} \ .$$

Case iii. $s_a = 0$, $s_o > 0$: In this case, $\frac{dG_W}{dt} = 0$, $\frac{dC^*}{dt} = s_o^\alpha + \sigma + n_o$, $\frac{dF_I}{dt} = n_a$, and $\frac{d\Phi_1}{dt} \le 0$ (by Corollary 5). To upper bound $\frac{d\Phi_2}{dt}$, we use Corollary 7(ii) again. Therefore,

$$\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le \beta s_o^\alpha + (\beta - 2)(n_a + \sigma) \le \beta(s_o^\alpha + \sigma) + (\beta - 2)n_a \le \beta \frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt} \ .$$

Case iv. $s_a = s_o = 0$: In this case, $\frac{dG_W}{dt} = 0$, $\frac{dC^*}{dt} \ge n_o$, $\frac{dF_I}{dt} = n_a$, and $\frac{d\Phi}{dt} = 0$. Therefore, $\frac{dG_W}{dt} + \frac{d\Phi}{dt} = 0 \le \beta \frac{dC^*}{dt}$. □

Generalized running condition. The above case analysis can be generalized to prove a more general running condition, which holds for other choices of β. In particular, in Cases (i) and (iii), we apply Corollary 7(i) instead, then we have $\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le \frac{\beta}{\alpha\mu^\alpha}\frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt}$. Combining with Cases (ii) and (iv), we have the following bound.

$$\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le \max\{\beta, \frac{\beta}{\alpha\mu^\alpha}\}\frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt}, \text{where } \mu > 0 \text{ and } \beta \ge 2/(1 - (1 - \tfrac{1}{\alpha})\mu^{\frac{\alpha}{\alpha-1}}). \tag{3}$$

In Lemma 8, we set $\mu = \alpha^{-1/\alpha}$ and $\beta = 2/(1 - \frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$, then $\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le \beta\frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt}$. In Section 5, when we consider bounded maximum speed, we will use a smaller $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ and a bigger $\mu = (\alpha+1)^{-1/\alpha}$. The running condition becomes $\frac{dG_W}{dt} + \frac{d\Phi}{dt} \le \beta(1+1/\alpha)\frac{dC^*}{dt} + (\beta-2)\frac{dF_I}{dt}$.

3.4 Summary

Under the general speed scaling model, we have shown a potential analysis of AJC; in particular, the running condition as stated in Lemma 8, the boundary and discrete events conditions lead to an upper bound on the working cost of AJC, completing the proof of Theorem 2. In the next section, we will analyze the sleep management algorithm IdleLonger. Then we can conclude the overall performance of IdleLonger(AJC).

Remark on the simple speed scaling model. Recall that in the simple speed scaling model, there is no sleep state and the static power is zero. In this case, AJC itself can produce a schedule with competitive performance on minimizing flow plus energy. The idea is to apply Theorem 2 with a trivial sleep management algorithm Slp$_o$ that always keeps the processor working whenever there are active jobs. Then both the inactive flow F_I and the inactive cost are zero, and hence the total cost of Slp$_o$(AJC) is equal to the working cost, which is at most β times the total cost of OPT (of the simple speed scaling model). The result is summarized below.

Corollary 9. *When there is no sleep state and the power function is in the form of s^α (i.e., $\sigma = 0$),* AJC *is β-competitive for flow plus energy, where $\beta = 2/(1 - \frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$.*

4 Sleep Management Algorithm IdleLonger

This section presents a sleep management algorithm called IdleLonger that determines when the processor should sleep, idle, and work. IdleLonger can be coupled with any speed scaling algorithm. In this section, we derive an upper bound on the inactive cost (as well as the inactive flow) of IdleLonger independent of the choice of the speed scaling algorithm. As a warm-up, we first consider the case with a single sleep state. Afterwards, we consider the general case of multiple sleep states.

4.1 Sleep management for a single sleep state

When the processor is in the working state or sleep state, it is relatively simple to determine the next transition. In the former, the processor keeps on working as long as there is an active job; otherwise it switches to the idle state. In the sleep state, we avoid waking up immediately after a new job arrives as this requires energy. It is natural to wait until the new jobs have accumulated enough flow, say, at least the wake-up energy ω, then we let the processor switch directly to the working state. Below we refer to the flow accumulated due to new jobs over a period of idle or sleep state as the *inactive flow* of that period.

When the processor is in the idle state, it is non-trivial when to switch to the sleep or working state. Intuitively, the processor should not stay idle too long, because it consumes energy (at the rate of σ) but does not get any work done. Yet to avoid frequent wake-up in future, the processor should not sleep immediately. Instead the processor should wait for possible job arrival and sleep only after the idling energy (i.e., σ times the length of idling interval) reaches the wake-up energy ω. When a new job arrives in the idle state, a naive idea is to let the processor switch to the working state to process the job immediately; this avoids accumulating inactive flow. Yet this turns out to be a bad strategy as it becomes too difficult to sleep; e.g., the adversary can use some tiny jobs sporadically, then the processor would never accumulate enough idling energy to sleep.

It is perhaps counter-intuitive that IdleLonger always prefers to idle a bit longer, and it can switch to the sleep state even in the presence of active jobs. The idea is to consider the inactive flow and idling energy at the same time. Note that when an idling period gets longer, both the inactive flow and idling energy increase, but at different rates. We imagine that these two quantities are competing with each other.

> The processor switches from the idle state to the working state once the inactive flow incurred during the current idle period catches up with the idling energy. If the idling energy reaches ω before the inactive flow catches up with the idling energy, the processor switches to the sleep state.

Algorithm 1 gives a summary of the above discussion. For simplicity, IdleLonger is written in a way that it is being executed continuously. In practice, we can rewrite the algorithm such that the execution is driven by discrete events like job arrival, job completion and wake-up. Furthermore, one might implement the algorithm slightly differently by running a newly arrived job immediately when the processor is in idle state, while keeping the same accounting of inactive flow to determine when to sleep. This makes the implementation and calculation more complicated, but cannot improve the competitive ratio in the worst case. We adopt the rules as shown in Algorithm 1.

Algorithm 1 IdleLonger(A): A is any speed scaling algorithm

```
At any time $t$, let $n(t)$ be the number of active jobs.
In working state: If $n(t) > 0$, keep working on jobs according to the algorithm $A$; else (i.e., $n(t) = 0$), switch to idle state.
In idle state: Let $t' \leq t$ be the last time in working state ($t' = 0$ if undefined). If the inactive flow over $[t', t]$ equals $(t - t')\sigma$, then switch to working state;
Else if $(t - t')\sigma = \omega$, switch to sleep state.
In sleep state: Let $t' \leq t$ be the last time in working state ($t' = 0$ if undefined). If the inactive flow over $[t', t]$ equals $\omega$, switch to working state.
```

Now we upper bound the inactive cost of IdleLonger. It is useful to define three types of time intervals. An $\mathrm{I_w}$-interval is a maximal interval in idling state with a transition to the working state at the end, and similarly an $\mathrm{I_s}$-interval for that with a transition to the sleep state. Furthermore, an $\mathrm{IS_w}$-interval is a maximal interval comprising an $\mathrm{I_s}$-interval, a sleeping interval, and finally a wake-up transition. As the processor starts in the sleep state, we allow the first $\mathrm{IS_w}$-interval containing no $\mathrm{I_s}$-interval.

Consider a schedule of IdleLonger(A). Recall that the inactive cost is composed of W (wake-up energy), F_{I} (inactive flow), and E_{I} (idling energy). We further divide E_{I} into two types: E_{IW} is the idling energy incurred in all $\mathrm{I_w}$-intervals, and E_{IS} for all $\mathrm{I_s}$-intervals.

By the definition of IdleLonger, we have the following property.

Property 10. (i) $F_{\mathrm{I}} \leq W + E_{\mathrm{IW}}$, *and* (ii) $E_{\mathrm{IS}} = W$.

By Property 10, the inactive cost of IdleLonger, defined as $W + F_{\mathrm{I}} + E_{\mathrm{IW}} + E_{\mathrm{IS}}$, is at most $3W + 2E_{\mathrm{IW}}$. The non-trivial part is to upper bound W and E_{IW}. Our main result is stated below (Theorem 11). For the optimal offline algorithm OPT, we divide its total cost G^* into two parts: W^* is the total wake-up energy, and $C^* = G^* - W^*$ (i.e., the total flow plus the working and idling energy).

Theorem 11. $W + E_{\mathrm{IW}} \leq C^* + 2W^*$.

Corollary 12. *The inactive cost of* IdleLonger *is at most* $3C^* + 6W^*$.

Before we move on to the detailed analysis, we discuss the above framework for the cases of multiple sleep states and bounded speed model. In Sections 4.2 and 5.2, we show that Property 10 remains true, implying that the inactive cost is still at most $3W + 2E_{\mathrm{IW}}$. Furthermore, we show that Theorem 11 continues to hold for multiple sleep states (see Theorem 14) and so does Corollary 12, which is a result of simple arithmetic as long as Theorem 11 holds. On the other hand, in the bounded speed model, we have to relax the bound in Theorem 11 to Theorem 22(i), leading to a larger bound on the inactive cost in Theorem 22(ii).

The rest of this section is devoted to proving Theorem 11. Note that W is the wake-up energy consumed at the end of all $\mathrm{IS_w}$-intervals, and E_{IW} is the idling energy of all $\mathrm{I_w}$-intervals. All these intervals are disjoint. Below we show a charging scheme such that, for each $\mathrm{IS_w}$-interval, we charge OPT a cost at least ω, and for each $\mathrm{I_w}$-interval, we charge OPT at least the idling energy of this interval. Thus, the total charge to OPT is at least $W + E_{\mathrm{IW}}$. On the other hand, we argue that the total charge is at most $C^* + 2W^*$. Therefore, $W + E_{\mathrm{IW}} \leq C^* + 2W^*$. Roughly speaking, the total flow plus the working and idling energy C^* of OPT is charged at most once while the wake-up energy W^* is charged at most twice. The detailed accounting is given after we describe the charging scheme.

The charging scheme for an $\mathrm{IS_w}$-interval $[t_1, t_2]$ is as follows. The target is at least ω.

Case 1. If OPT switches from or to the sleep state in $[t_1, t_2]$, we charge OPT the cost ω of the first wake-up in $[t_1, t_2]$ (if it exists) or of the last wake-up before t_1.

Case 2. If OPT is awake throughout $[t_1, t_2]$, we charge OPT the static energy $(t_2 - t_1)\sigma$. Note that in an $\mathrm{IS_w}$-interval, IdleLonger has an idle-sleep transition, and hence $(t_2 - t_1)\sigma > \omega$.

Case 3. If OPT is sleeping throughout $[t_1, t_2]$, we charge OPT the inactive flow (i.e., the flow incurred by new jobs) over $[t_1, t_2]$. In this case, OPT and IdleLonger have the same amount of inactive flow during $[t_1, t_2]$ (because IdleLonger only transits to idle when there are no active jobs and all its inactive flow during $[t_1, t_2]$ is due to new jobs), which equals ω (because IdleLonger wakes up at t_2).

For an $\mathrm{I_w}$-interval, we use the above charging scheme again. The definition of $\mathrm{I_w}$-interval allows the scheme to guarantee a charge of $(t_2 - t_1)\sigma$ instead of ω. Specifically, as an $\mathrm{I_w}$-interval ends with an idle-working transition, the inactive flow accumulated in $[t_1, t_2]$ is $(t_2 - t_1)\sigma$, and the latter cannot exceed ω. Therefore, the charge of Case 1, which equals ω, is at least $(t_2 - t_1)\sigma$. Case 2 charges exactly $(t_2 - t_1)\sigma$. For Case 3, we charge OPT the inactive flow during $[t_1, t_2]$. Note that OPT and IdleLonger accumulate the same inactive flow, which is $(t_2 - t_1)\sigma$.

Summing over all $\mathrm{I_w}$- and $\mathrm{IS_w}$-intervals, we have charged OPT at least $W + E_{\mathrm{IW}}$. On the other hand, since all these intervals are disjoint, in Cases 2 and 3, the charge comes from non-overlapping flow and energy of C^*. In Case 1, each OPT's wake-up from the sleep state is charged for ω at most twice, thus the total charge is at most $2W^*$. In conclusion, $W + E_{\mathrm{IW}} \le C^* + 2W^*$.

4.2 Sleep management for $m \ge 2$ levels of sleep states

We extend the previous sleep management algorithm to allow intermediate sleep states, which demand less idling (static) energy than the idle state, and also less wake-up energy than the final sleep state (i.e., sleep-m state). We treat the sleep-m state as the only sleep state in the single-level setting, and adapt the transition rules of the idle state for the intermediate sleep states. The key idea is again to compare inactive flow against idling energy continuously. To ease our discussion, we treat the idle state as the sleep-0 state with wake-up energy $\omega_0 = 0$. Details are given in Algorithm 2.

Algorithm 2 IdleLonger(A): A is any speed scaling algorithm

```
At any time $t$, let $n(t)$ be the number of active jobs.
In working state: If $n(t) > 0$, keep working on the jobs according to the algorithm $A$; else if $n(t) = 0$, switch to idle state.
In sleep-$j$ state, where $0 \le j \le m-1$: Let $t' \le t$ be the last time in the working state, and let $t''$, where $t' \le t'' \le t$, be the last time switching from sleep-$(j-1)$ state to sleep-$j$ state. If the inactive flow over $[t', t]$ equals $(t - t'')\sigma_j + \omega_j$, then wake up to the working state;
Else if $(t - t'')\sigma_j = (\omega_{j+1} - \omega_j)$, switch to sleep-$(j+1)$ state.
In sleep-$m$ state: Let $t' \le t$ be the last time in the working state. If the inactive flow over $[t', t]$ equals $\omega_m$, then wake up to the working state.
```

When we analyze the multi-level algorithm, the definition of W (total wake-up cost) and F_{I} (total inactive flow) remain the same, but E_{IS} and E_{IW} have to be generalized. Below we refer a maximal interval during which the processor is in a particular sleep-j state, where $0 \le j \le m$, as a *sleep interval* or more specifically, a sleep-j interval. Note that all sleep intervals, except sleep-m intervals, demand idling (static) energy. We denote $\boldsymbol{E}_{\text{IW}}$ as the idling energy for all sleep intervals that end with a wake-up transition, and $\boldsymbol{E}_{\text{IS}}$ the idling energy of all sleep intervals ending with a (deeper) sleep transition.

IdleLonger imposes a rigid structure of sleep intervals. Define $\ell_j = (\omega_{j+1} - \omega_j)/\sigma_j$. A sleep-$j$ interval can appear only after a sequence of lower level sleep intervals, which starts with an sleep-0 interval of length ℓ_0, followed by a sleep-1 interval of length $\ell_1, \ldots$, and finally a sleep-$(j-1)$ interval of length ℓ_{j-1}. Consider a maximum sequence of such sleep intervals that ends with a transition to the working state. We call the entire time interval enclosed by this sequence an $\text{IS}_\text{w}[j]$-interval for some $0 \le j \le m$ if the deepest (also the last) sleep subinterval is of level j. It is useful to observe the following lemma about an $\text{IS}_\text{w}[j]$-interval $[t_1, t_2]$. Intuitively, this lemma gives an upper bound of the inactive flow over the $\text{IS}_\text{w}[j]$-interval, in terms of the cost of the optimal offline schedule OPT. Roughly speaking, suppose the deepest sleep state of OPT during $[t_1, t_2]$ is sleep-k. The lemma says that the inactive flow incurred by IdleLonger over the $\text{IS}_\text{w}[j]$-interval is at most the static energy of OPT over the whole interval $[t_1, t_2]$ plus the wake-up energy required by OPT to switch back from sleep-k to working.

Lemma 13. *Consider any* $\text{IS}_\text{w}[j]$*-interval* $[t_1, t_2]$*, where* $0 \le j \le m$*. Assume that the last sleep-*j *subinterval is of length* ℓ*. Then,* $\omega_j + \ell\sigma_j \le \omega_k + (t_2 - t_1)\sigma_k$ *for any* $0 \le k \le m$*.*

Proof. We consider two cases depending on whether $k > j$.

If $k > j$, then $j \le k-1 \le m-1$. Since $j \le m-1$, by the definition of IdleLonger, $\ell \le \ell_j$. Then $\omega_j + \ell\sigma_j \le \omega_j + \ell_j\sigma_j = \omega_{j+1} \le \omega_k$ and thus $\omega_j + \ell\sigma_j \le \omega_k + (t_2 - t_1)\sigma_k$.

Otherwise, $k \le j$ and we count the energy usage of the $\text{IS}_\text{w}[j]$-interval in two different ways; one count is exactly $\omega_j + \ell\sigma_j$ and the other is at most $\omega_k + (t_2 - t_1)\sigma_k$, which implies $\omega_j + \ell\sigma_j \le \omega_k + (t_2 - t_1)\sigma_k$. Note that for $0 \le i \le j-1$, the energy usage in a sleep-i interval is $\ell_i\sigma_i = \omega_{i+1} - \omega_i$. Thus, the energy usage in the $\text{IS}_\text{w}[j]$-interval is $\sum_{0 \le i \le j-1} \ell_i\sigma_i + \ell\sigma_j = (\omega_j - \omega_0) + \ell\sigma_j = \omega_j + \ell\sigma_j$, where the last equality is due to $\omega_0 = 0$. On the other hand, the energy usage in the first k sleep intervals is $\sum_{0 \le i \le k-1} \ell_i\sigma_i = \omega_k - \omega_0 = \omega_k$, while the other energy usage is $\sum_{k \le i \le j-1} \ell_i\sigma_i + \ell\sigma_j \le (\sum_{k \le i \le j-1} \ell_i + \ell) \cdot \sigma_k \le (t_2 - t_1)\sigma_k$. In conclusion, $\omega_j + \ell\sigma_j \le \omega_k + (t_2 - t_1)\sigma_k$. $\square$

Adaptation of analysis. We now adapt the analysis in Section 4.1. First of all, we show that the rigid sleeping structure of IdleLonger allows us to maintain Property 10 as before. That is, (i) $F_\text{I} \le W + E_{\text{IW}}$, and (ii) $E_{\text{IS}} = W$. We further show in Theorem 14 that W and E_{IW} have the same upper bound as in Theorem 11. Theorem 14 and Property 10 together imply that the inactive cost, which is equal to which is equal to $F_\text{I} + E_{\text{IW}} + E_{\text{IS}} + W$, is still at most $3W + 2E_{\text{IW}}$, i.e., Corollary 12 still holds. This bound on the inactive cost together with the bound on working cost in Theorem 2 give the overall competitiveness of IdleLonger(AJC) in Theorem 15.

Details of analysis. We now give the proof of Property 10 and Theorem 14.

Proof of Property 10. (i) Note that F_I is equal to the inactive flow incurred in all $\text{IS}_\text{w}[j]$-intervals, where $0 \le j \le m$. Consider any $\text{IS}_\text{w}[j]$-interval. Assume that the last sleep-j subinterval is of length ℓ. By definition of IdleLonger, the inactive flow is at most the idling energy $\ell\sigma_j$ of the last sleep subinterval plus the energy ω_k of the wake-up at the end. Summing over all $\text{IS}_\text{w}[j]$-intervals, we have $F_\text{I} \le E_{\text{IW}} + W$.

(ii) Consider all $\text{IS}_\text{w}[j]$-intervals. The first $\text{IS}_\text{w}[j]$-interval is simply a sleep-m interval, which does not incur any idling energy, while in any other $\text{IS}_\text{w}[j]$-interval, the total idling energy of all sleep subintervals except the last subinterval is $\sum_{0\le i\le j-1} \ell_i\sigma_i = \omega_j - \omega_0 = \omega_j$. Note that E_{IS} is the sum of such idling energy ω_j of all $\text{IS}_\text{w}[j]$-intervals (except the first $\text{IS}_\text{w}[j]$-interval) plus the idling energy incurred in the sleep intervals which occur after the last wake up. By the rigid sleeping structure of IdleLonger, the latter term equals $\sum_{0\le i\le m-1} \ell_i\sigma_i = \omega_m - \omega_0 = \omega_m$. On the other hand, W is the sum of the wake-up energy ω_m of the first $\text{IS}_\text{w}[j]$-interval and the wake-up energy ω_j of the other $\text{IS}_\text{w}[j]$-intervals. In conclusion, we have $E_{\text{IS}} = W$. □

Theorem 14. *In the setting of $m \ge 2$ sleep states, $W + E_{\text{IW}} \le C^* + 2W^*$.*

Proof. To account for W and E_{IW}, it suffices to look at all $\text{IS}_\text{w}[j]$-intervals, where $0 \le j \le m$. For each $\text{IS}_\text{w}[j]$-interval, we show how to charge OPT a cost $\omega_j + \ell\sigma_j$, where ℓ is the length of the deepest sleep subinterval (it is useful to recall that $\omega_0 = 0$ and $\sigma_m = 0$). Then we argue that the total cost charged is at least $W + E_{\text{IW}}$ and at most $C^* + 2W^*$ (similar to Section 4.1, the total flow plus the working and idling energy C^* of OPT is charged at most once while the wake-up energy W^* is charged at most twice).

Without loss of generality, we can assume that in a maximal interval $[r_1, r_2]$ that OPT is not working, if OPT has ever slept (in sleep-1 or deeper sleep state), then $[r_1, r_2]$ contains only one sleep transition, which occurs at r_1, and the processor remains in the same sleep state until r_2.

Charging scheme. Consider any $\text{IS}_\text{w}[j]$-interval $[t_1, t_2]$, where $0 \le j \le m$. Let ℓ be the length of the sleep-j subinterval in this interval.

Case 1. If OPT has ever switched from or to the sleep-1 or deeper sleep state in $[t_1, t_2]$, let $k \ge 1$ be the deepest sleep level involved in the entire interval. Note that OPT uses static energy at least $(t_2 - t_1)\sigma_k$ during $[t_1, t_2]$. We charge OPT the sum of $(t_2 - t_1)\sigma_k$ and ω_k (in view of a wake-up from sleep-k state inside $[t_1, t_2]$ or after t_2; if there is no-wake up after t_2, then we charge OPT the first wake-up). By Lemma 13, this charge is at least $\omega_j + \ell\sigma_j$.

Case 2. If OPT is working or idle throughout $[t_1, t_2]$, we charge OPT the static energy $(t_2 - t_1)\sigma_0$, which, by Lemma 13, is at least $\omega_j + \ell\sigma_j$.

Case 3. If OPT is sleeping (at any level except zero) throughout $[t_1, t_2]$, we charge OPT the inactive flow over $[t_1, t_2]$. Note that OPT has the same amount of inactive flow as IdleLonger because IdleLonger only transits to idle when there are no active jobs and all its inactive flow during $[t_1, t_2]$ is due to new jobs. By definition of a wake-up transition in IdleLonger, the inactive flow equals $\omega_j + \ell\sigma_j$.

Since $\text{IS}_\text{w}[j]$-intervals are all disjoint, the flow and idling (static) energy charged to OPT by Cases 1, 2 and 3 come from different parts of C^*. For Case 1, each of OPT's wake-up from a sleep state is charged at most twice. Thus, $W + E_{\text{IW}} \le C^* + 2W^*$, completing the proof of the theorem. □

Remarks on bounded speed model. In Section 5, we are going to adapt IdleLonger to the bounded speed model. We add one more wake-up condition to wake up the processor to the working state if the number of active jobs exceeds a threshold. The motivation of this additional rule is explained there. We note that Lemma 13 and Property 10(i) give upper bounds on the inactive flow of IdleLonger. Since the adaptation in the bounded speed model only wakes up the processor earlier and this does not increase the inactive flow of any $\text{IS}_\text{w}[j]$-interval, both of these results continue to hold. For a similar reason, Property 10(ii) also holds since E_{IS} is not affected either. On the other hand, in the charging scheme to prove Theorem 14, Case 3 works well in $\text{IS}_\text{w}[j]$-intervals where IdleLonger wakes up at the end due to excessive inactive flow but not if it is due to large amount of active jobs. In Section 5.2, we describe how to handle this case and show that having to deal with this case leads to slightly worse bound on W plus E_{IW} in Theorem 22(i).

4.3 Competitiveness of IdleLonger(AJC)

The above inactive cost of IdleLonger, together with the working cost of AJC in Section 3, give the following competitiveness of IdleLonger(AJC). Let $\beta = 2/(1 - \frac{\alpha-1}{\alpha^{\alpha/(\alpha-1)}})$.

Theorem 15. *In the general speed scaling model with $m \geq 1$ sleep states, the total cost of* IdleLonger(AJC) *is at most $(2\beta + 2)$ times of the total cost of* OPT*.*

Proof. Consider the coupling of IdleLonger and AJC. By Property 10, Theorems 11 and 14, we have $F_{\mathrm{I}} \leq C^* + 2W^*$. Furthermore, by Corollary 12, the inactive cost is at most $3C^* + 6W^*$. In Section 3, we proved Theorem 2 that the working cost is $G_{\mathrm{W}} \leq \beta C^* + (\beta - 2)F_{\mathrm{I}}$. Therefore, the total cost of IdleLonger(AJC), comprised of the inactive cost and the working cost, is at most $(2\beta + 1)C^* + (2\beta + 2)W^*$, which is at most $(2\beta + 2)$ times of OPT's total cost. □

5 Bounded Speed Model

This section extends the speed scaling algorithm AJC and the sleep management algorithm IdleLonger to the setting where the processor speed is upper bounded by a constant $T > 0$. We analyze the working cost of AJC and the inactive cost of IdleLonger under the general speed scaling model (which assumes $m \geq 1$ levels of sleep states), and show that the total cost of IdleLonger(AJC) is $O(1)$ times of the optimal offline algorithm OPT.

Adaptation. Recall that AJC runs at the speed $(n(t) + \sigma)^{1/\alpha}$, where $n(t)$ is the number of active jobs at time t. To extend AJC to the bounded speed model, we simply cap the speed at T. That is, at any time t, the processor runs at the speed $\min\{(n(t) + \sigma)^{1/\alpha}, T\}$.

In the bounded speed model, IdleLonger (see Section 4) still works and the inactive cost is $O(1)$ times of OPT's total cost. However, IdleLonger often allows a long sleep, then a speed scaling algorithm, without the capability to speed up arbitrarily, cannot always catch up with the progress of OPT and may have unbounded working cost. Thus, we adapt IdleLonger to wake up earlier, especially when many new jobs have arrived. We add one more wake-up condition to IdleLonger. Recall that $\sigma (= \sigma_0)$ is the static power in the working state.

> In the sleep-j state, where $0 \leq j \leq m$, if the number of active jobs exceeds σ, the processor wakes up to the working state.

In the rest of this section, we will analyze the working cost of AJC and the inactive cost of IdleLonger. Throughout this section, we set the constant $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$.

5.1 Working cost of AJC

Following Section 3, we want to show that $G_{\mathrm{W}} = O(C^* + F_{\mathrm{I}})$, where G_{W} is the working cost of IdleLonger(AJC), C^* is the total cost of OPT minus its wake-up energy W^*, and F_{I} is the inactive flow of IdleLonger(AJC).

Adaptation of analysis. We adapt the potential analysis of AJC in Section 3.2. We use the same potential function $\Phi(t)$ except that β is set to a smaller value $2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$.

As shown in Section 3.2, it suffices to show at any time t that when no discrete event occurs, $\frac{\mathrm{d}G_{\mathrm{W}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} = O(\frac{\mathrm{d}C^*}{\mathrm{d}t} + \frac{\mathrm{d}F_{\mathrm{I}}}{\mathrm{d}t})$. To upper bound $\frac{\mathrm{d}\Phi}{\mathrm{d}t}$, we again divide $\frac{\mathrm{d}\Phi}{\mathrm{d}t}$ into two parts such that $\frac{\mathrm{d}\Phi}{\mathrm{d}t} = \frac{\mathrm{d}\Phi_1}{\mathrm{d}t} + \frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$, where $\mathrm{d}\Phi_1$ is the change of Φ due to execution of IdleLonger(AJC) and $\mathrm{d}\Phi_2$ due to OPT. It is useful to recall the following notations: At time t, n_{a} and n_{o} denote the number of active jobs in IdleLonger(AJC) and OPT, respectively; s_a and s_o are the current speed of IdleLonger(AJC) and OPT, respectively.

Note that Lemma 4 remains valid as long as we choose β greater than zero. That is, we still have the following upper bound of $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$.

$$\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_\mathrm{a} + \sigma - n_\mathrm{o})\frac{s_\mathrm{a}}{(n_\mathrm{a}+\sigma)^{1/\alpha}} \ . \tag{4}$$

Furthermore, as we set $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$, Corollary 7(iii) provides the following upper bound of $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t}$.

$$\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le \beta(1+1/\alpha)s_\mathrm{o}^\alpha + (\beta - 2)(n_\mathrm{a} + \sigma) \ .$$

As we discussed in Section 3.2, Corollary 5 does not hold when $s_\mathrm{a} = T$. In this case, we need to bound $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$ differently in Lemma 17 and relax the running condition of Lemma 8 to Lemma 18. Then we can bound the working cost in Theorem 20 (cf. Theorem 2). The details are as follows.

Details of analysis. If $s_\mathrm{a} < T$, then $s_\mathrm{a} = (n_\mathrm{a} + \sigma)^{1/\alpha}$ and Ineqaulity (4) becomes $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_\mathrm{a} + \sigma - n_\mathrm{o})$. That is, the upper bound of $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$ remains the same as in Corollary 5. Thus, the generalized running condition of Lemma 8, i.e., Inequality (3), still holds. In particular, for $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ and $\mu = (\alpha+1)^{-1/\alpha}$,

$$\frac{\mathrm{d}G_\mathrm{W}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le \max\{\beta, \frac{\beta}{\alpha\mu^\alpha}\}\frac{\mathrm{d}C^*}{\mathrm{d}t} + (\beta-2)\frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t} \le \beta(1+1/\alpha)\frac{\mathrm{d}C^*}{\mathrm{d}t} + (\beta-2)\frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t}.$$

The rest of this section is devoted to the non-trivial case when $s_\mathrm{a} = T$, i.e., IdleLonger(AJC) is working at the maximum speed T. In this case, Lemma 4 is no long sufficient to obtain a meaningful bound for $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$, as in Corollary 5. Intuitively, when n_a is big, the ideal speed $(n_\mathrm{a} + \sigma)^{1/\alpha}$ can be way higher than the actual speed T. This means a longer working period and hence an excess of flow time and static energy.

To upper bound such excess of flow time and static energy, we need two techniques. First, we observe that the modified algorithms IdleLonger and AJC can indeed guarantee that IdleLonger(AJC) keeps up with OPT on the number of active jobs. Lemma 16 below shows that $n_\mathrm{a} - n_\mathrm{o}$ is bounded by T^α and σ. Together with Lemma 4, we are able to upper bound $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t}$ as $O(n_\mathrm{o} - n_\mathrm{a})$ (see Lemma 17). Second, we relax the running condition with a new notion E_WS, the *total working static energy*, which is the static power σ times the total length of working intervals of IdleLonger(AJC). When $s_\mathrm{a} > 0$, $\frac{\mathrm{d}E_\mathrm{WS}}{\mathrm{d}t} = \sigma$. A detailed analysis shows that $\frac{\mathrm{d}G_\mathrm{W}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} = O(\frac{\mathrm{d}C^*}{\mathrm{d}t} + \frac{\mathrm{d}F_\mathrm{I}}{\mathrm{d}t} + \frac{\mathrm{d}E_\mathrm{WS}}{\mathrm{d}t})$ (see Lemma 18). Since E_WS changes continuously over time, together with the boundary and discrete events conditions, $G_\mathrm{W} = O(C^* + F_\mathrm{I} + E_\mathrm{WS})$. Finally, we prove that $E_\mathrm{WS} \le C^*$ (see Lemma 19). Then it follows that $G_\mathrm{W} = O(C^* + F_\mathrm{I})$.

Lemma 16. *At any time t, $n_\mathrm{a} - n_\mathrm{o} \le \max\{T^\alpha, \sigma\}$.*

Proof. The non-trivial case is when $n_\mathrm{a} > \max\{T^\alpha, \sigma\}$. Let $t_0 < t$ be the last time when $n_\mathrm{a} \le \max\{T^\alpha, \sigma\}$. By definition, IdleLonger(AJC) is working at maximum speed T using SRPT during $[t_0, t]$. Suppose that h new jobs are released during $[t_0, t]$ and OPT can complete ℓ of them by time t. Note that all these h jobs (and some other active jobs at t_0) are available for IdleLonger(AJC) to process and it is possible to complete ℓ of them (as OPT can). Since SRPT maximizes the number of jobs completed by any time [21], the number of jobs completed by IdleLonger(AJC) during $[t_0, t]$ is at least ℓ (among the h new jobs and the active jobs at t_0). Thus, $n_\mathrm{a} \le \max\{T^\alpha, \sigma\} + h - \ell \le \max\{T^\alpha, \sigma\} + n_\mathrm{o}$. $\square$

Lemma 17. *Assume that* $s_{\mathrm{a}} = T$. **(i)** *If* $n_{\mathrm{a}} > n_{\mathrm{o}} + \sigma$, *then* $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} - (1+1/\alpha)n_{\mathrm{o}})$. **(ii)** *If* $n_{\mathrm{o}} - \sigma < n_{\mathrm{a}} \le n_{\mathrm{o}} + \sigma$, *then* $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le 0$. **(iii)** *If* $n_{\mathrm{a}} \le n_{\mathrm{o}} - \sigma$, *then* $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}})$.

Proof. Assume that $s_{\mathrm{a}} = T$. **(i)** Consider $n_{\mathrm{a}} - n_{\mathrm{o}} > \sigma$. Lemma 16 shows that n_{a} cannot be arbitrarily larger than n_{o}, and hence $\sigma < n_{\mathrm{a}} - n_{\mathrm{o}} \le T^{\alpha}$. Note that for any non-negative x and y, if $x \le y$, then $1 + \sigma/x \ge 1 + \sigma/y \ge (1+\sigma/y)^{1/\alpha}$, and thus $(x+\sigma)/(y+\sigma)^{1/\alpha} \ge x/y^{1/\alpha}$. By setting $x = n_{\mathrm{a}} - n_{\mathrm{o}}$ and $y = n_{\mathrm{a}}$, Inequality (4) becomes $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} - n_{\mathrm{o}}){n_{\mathrm{a}}}^{-1/\alpha}T$. Since $T \ge (n_{\mathrm{a}} - n_{\mathrm{o}})^{1/\alpha}$, we have

$$\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le \frac{-\beta(n_{\mathrm{a}}-n_{\mathrm{o}})^{1+1/\alpha}}{n_{\mathrm{a}}^{1/\alpha}} \le \frac{-\beta({n_{\mathrm{a}}}^{1+1/\alpha} - (1+1/\alpha)n_{\mathrm{a}}^{1/\alpha}n_{\mathrm{o}})}{n_{\mathrm{a}}^{1/\alpha}} = -\beta(n_{\mathrm{a}} - (1+1/\alpha)n_{\mathrm{o}}) \ .$$

Note that the second inequality is due to the fact that for any $x \ge 1$ and a, b, $(a+b)^x \ge a^x + xa^{x-1}b$.

(ii) If $-\sigma < n_{\mathrm{a}} - n_{\mathrm{o}} \le \sigma$, then Inequality (4) simply implies that $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le 0$.

(iii) Note that $T \le (n_{\mathrm{a}} + \sigma)^{1/\alpha}$. If $n_{\mathrm{a}} \le n_{\mathrm{o}} - \sigma$, then $-\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}}) \ge 0$. By Inequality (4),

$$\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}})\frac{T}{(n_{\mathrm{a}}+\sigma)^{1/\alpha}} \le -\beta(n_{\mathrm{a}} + \sigma - n_{\mathrm{o}}) \ . \qquad \square$$

We are now ready to prove the relaxed running condition which uses E_{ws} to bound the excess static energy.

Lemma 18. *Assume that* $s_{\mathrm{a}} = T$. *At any time when no discrete events occur,* $\frac{\mathrm{d}G_{\mathrm{w}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le \beta(1+\frac{1}{\alpha})\frac{\mathrm{d}C^*}{\mathrm{d}t} + (\beta - 2)\frac{\mathrm{d}F_{\mathrm{I}}}{\mathrm{d}t} + \zeta\frac{\mathrm{d}E_{\mathrm{ws}}}{\mathrm{d}t}$, *where* $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ *and* $\zeta = \max\{4, \beta(1-1/\alpha)\}$.

Proof. We give a case analysis depending on whether OPT is working and the relative values of n_{a} and n_{o}. Notice that when $s_{\mathrm{a}} = T$, we have $T \le (n_{\mathrm{a}} + \sigma)^{1/\alpha}$, and $\frac{\mathrm{d}G_{\mathrm{w}}}{\mathrm{d}t} \le 2(n_{\mathrm{a}} + \sigma)$. Furthermore, $\frac{\mathrm{d}F_{\mathrm{I}}}{\mathrm{d}t} = 0$, and $\frac{\mathrm{d}E_{\mathrm{ws}}}{\mathrm{d}t} = \sigma$.

Case A. $\boldsymbol{n_{\mathrm{a}} > n_{\mathrm{o}} + \sigma}$ **and** $\boldsymbol{s_{\mathrm{o}} = 0}$**:** In this case, $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le 0$, and $\frac{\mathrm{d}C^*}{\mathrm{d}t} \ge n_{\mathrm{o}}$. By Lemma 17(i), $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le -\beta(n_{\mathrm{a}} - (1+1/\alpha)n_{\mathrm{o}})$. Thus,

$$\begin{aligned}\frac{\mathrm{d}G_{\mathrm{w}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} &\le 2(n_{\mathrm{a}} + \sigma) - \beta(n_{\mathrm{a}} - (1+1/\alpha)n_{\mathrm{o}}) = \beta(1+1/\alpha)n_{\mathrm{o}} + 2\sigma - (\beta-2)n_{\mathrm{a}} \\ &\le \beta(1+1/\alpha)\frac{\mathrm{d}C^*}{\mathrm{d}t} + \zeta\frac{\mathrm{d}E_{\mathrm{ws}}}{\mathrm{d}t} \ .\end{aligned}$$

The last inequality follows from the fact that $\beta \ge 2$ and $\zeta \ge 4$.

Case B. $\boldsymbol{n_{\mathrm{a}} > n_{\mathrm{o}} + \sigma}$ **and** $\boldsymbol{s_{\mathrm{o}} > 0}$**:** In this case, $\frac{\mathrm{d}C^*}{\mathrm{d}t} = n_{\mathrm{o}} + s_{\mathrm{o}}^{\alpha} + \sigma$. By Lemma 17(i) and Corollary 7(iii),

$$\begin{aligned}\frac{\mathrm{d}G_{\mathrm{w}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} &\le 2(n_{\mathrm{a}} + \sigma) - \beta(n_{\mathrm{a}} - (1+1/\alpha)n_{\mathrm{o}}) + \beta(1+1/\alpha)s_{\mathrm{o}}^{\alpha} + (\beta-2)(n_{\mathrm{a}} + \sigma) \\ &\le \beta(1+1/\alpha)n_{\mathrm{o}} + \beta(1+1/\alpha)s_{\mathrm{o}}^{\alpha} + \beta\sigma \\ &\le \beta(1+1/\alpha)\frac{\mathrm{d}C^*}{\mathrm{d}t} \ .\end{aligned}$$

Case C. $\boldsymbol{n_{\mathrm{o}} - \sigma < n_{\mathrm{a}} \le n_{\mathrm{o}} + \sigma}$ **and** $\boldsymbol{s_{\mathrm{o}} = 0}$**:** In this case, $\frac{\mathrm{d}\Phi_2}{\mathrm{d}t} \le 0$, and $\frac{\mathrm{d}C^*}{\mathrm{d}t} \ge n_{\mathrm{o}}$. By Lemma 17(ii), $\frac{\mathrm{d}\Phi_1}{\mathrm{d}t} \le 0$. Thus

$$\frac{\mathrm{d}G_{\mathrm{w}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 2(n_{\mathrm{a}} + \sigma) \le 2((n_{\mathrm{o}} + \sigma) + \sigma) \le \beta(1+1/\alpha)\frac{\mathrm{d}C^*}{\mathrm{d}t} + \zeta\frac{\mathrm{d}E_{\mathrm{ws}}}{\mathrm{d}t} \ .$$

The last inequality follows from the fact that $\beta \geq 2$ and $\zeta \geq 4$.

Case D. $n_o - \sigma < n_a \leq n_o + \sigma$ and $s_o > 0$: In this case, $\frac{dC^*}{dt} = n_o + s_o^\alpha + \sigma$. By Lemma 17(ii), $\frac{d\Phi_1}{dt} \leq 0$. Together with Corollary 7(iii), we can show that

$$\begin{aligned}
\tfrac{dG_w}{dt} + \tfrac{d\Phi}{dt} &\leq 2(n_a + \sigma) + \beta(1 + 1/\alpha)s_o^\alpha + (\beta - 2)(n_a + \sigma) \\
&= \beta(1 + 1/\alpha)s_o^\alpha + \beta(n_a + \sigma) \\
&\leq \beta(1 + 1/\alpha)s_o^\alpha + \beta(n_o + \sigma + \sigma) \\
&\leq \beta(1 + 1/\alpha)(s_o^\alpha + n_o + \sigma) + (\beta + \beta - \beta(1 + 1/\alpha))\sigma \\
&= \beta(1 + 1/\alpha)\tfrac{dC^*}{dt} + \beta(1 - 1/\alpha)\tfrac{dE_{ws}}{dt} \\
&\leq \beta(1 + 1/\alpha)\tfrac{dC^*}{dt} + \zeta\tfrac{dE_{ws}}{dt}.
\end{aligned}$$

Case E. $n_a \leq n_o - \sigma$. This is a relatively trivial case. Lemma 17(iii) gives an upper bound of $\frac{d\Phi_1}{dt}$ same as Corollary 5, i.e., $\frac{d\Phi_1}{dt} \leq -\beta(n_a + \sigma - n_o)$. Thus the generalized running condition of Lemma 8, i.e., Inequality (3), holds again: For $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$, $\frac{dG_w}{dt} + \frac{d\Phi}{dt} \leq \beta(1 + 1/\alpha)\frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt}$.

In conclusion, in any one of the five cases, $\frac{dG_w}{dt} + \frac{d\Phi}{dt} \leq \beta(1 + 1/\alpha)\frac{dC^*}{dt} + (\beta - 2)\frac{dF_I}{dt} + \zeta\frac{dE_{ws}}{dt}$. □

Next we derive an upper bound of E_{ws} in terms of C^*, which would allow us to turn the running condition in Lemma 18 into our main result that $G_w = O(C^* + F_I)$ (see Theorem 20).

Lemma 19. *With respect to* IdleLonger(AJC)*,* $E_{ws} \leq C^*$*.*

Proof. Let x be the total size of all jobs. First, consider $T \geq \sigma^{1/\alpha}$. When IdleLonger(AJC) is working, its speed is at least $\sigma^{1/\alpha}$ and thus $E_{ws} \leq \sigma \cdot (x/\sigma^{1/\alpha}) = x\sigma^{1-1/\alpha}$. Define the critical speed $s_{crit} = (\sigma/(\alpha-1))^{1/\alpha}$. Note that running a job at the critical speed s_{crit} minimizes the energy usage of the job. To see why this is the case, suppose a job J with $p(J)$ units of work is processed to completion using speed s. The energy usage is $P(s)p(J)/s$, where $P(s) = s^\alpha + \sigma$. This energy usage is minimized if $P(s) = s \times P'(s)$, i.e., $s = (\sigma/(\alpha-1))^{1/\alpha}$. Therefore, the energy usage of any schedule and hence C^* is at least $(x/s_{crit}) \cdot (s_{crit}^\alpha + \sigma) \geq (\alpha/(\alpha-1)^{1-1/\alpha}) \cdot (x\sigma^{1-1/\alpha}) \geq (\alpha/(\alpha-1)^{1-1/\alpha})E_{ws}$. For any $\alpha > 1$, $(\alpha/(\alpha-1)^{1-1/\alpha}) \geq 1$, and hence $C^* \geq E_{ws}$.

Now consider the case that $T < \sigma^{1/\alpha}$. When IdleLonger(AJC) is working, its speed is always T and thus $E_{ws} \leq \sigma \cdot (x/T)$. If $s_{crit} \leq T$, then the energy usage of any schedule and hence C^* is at least $(x/s_{crit}) \cdot (s_{crit}^\alpha + \sigma) \geq \sigma \cdot (x/s_{crit}) \geq \sigma \cdot (x/T) = E_{ws}$. Otherwise, if $s_{crit} > T$, then the optimal schedule would always run a job at speed T. It is because when running a job below s_{crit}, the slower the speed, the more energy as well as flow time are incurred. Thus, $C^* \geq (x/T) \cdot (T^\alpha + \sigma) \geq \sigma \cdot (x/T) = E_{ws}$. □

Theorem 20. *With respect to* IdleLonger(AJC)*,* $G_w \leq (\beta(1 + 1/\alpha) + \zeta)C^* + (\beta - 2)F_I$*, where* $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ *and* $\zeta = \max\{4, \beta(1 - 1/\alpha)\}$*.*

Remark on the simple speed scaling model. Similar to Section 3.4, the above result on the working cost of AJC already implies that AJC itself is competitive in the simple speed scaling model where there is no sleep states and the static power is zero. In this case, we use a trivial sleep management algorithm Slp_o that always keeps the processor working whenever there are active jobs. Then both the inactive flow F_I and the inactive cost are zero, and hence the total cost of Slp_o(AJC) is equal to the working cost. Furthermore, the working static energy E_{ws} is also zero (cf. Lemma 19). Therefore, the working cost is at most $\beta(1+1/\alpha)$ times the total cost of OPT (of the simple speed scaling model). The result is summarized below.

Corollary 21. *When there is no sleep state and the power function is in the form of s^α (i.e., $\sigma = 0$),* AJC *is $\beta(1+1/\alpha)$-competitive for flow plus energy, where $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$.*

5.2 Inactive cost of IdleLonger

It remains to analyze the inactive cost of IdleLonger. The rigid structure of sleep intervals remains the same as before, and the inactive cost is still at most $3W + 2E_{\text{IW}}$, where W is the wake-up energy and E_{IW} is the idling energy incurred in those idling or intermediate sleep intervals that end with a wake-up transition (see Section 4 for details). I.e., Property 10 and Lemma 13 remain valid. However, due to the additional wake-up rule, IdleLonger has a slightly worse bound on W plus E_{IW}. Our main result is stated in Theorem 22. Again, W^* denotes the wake-up energy of OPT, and C^* is the total cost of OPT minus W^*.

Theorem 22. (i) $W + E_{\text{IW}} \leq C^* + 3W^*$. (ii) *The inactive cost of* IdleLonger *is at most* $3C^* + 9W^*$.

Theorem 22(ii) follows directly from Theorem 22(i) and Property 10. To prove Theorem 22(i), we extend the charging scheme in Section 4.2 to show that for each $\text{IS}_{\text{w}}[j]$-interval, OPT can be charged with a cost at least $\omega_j + \ell\sigma_j$, where ℓ is the length of the deepest sleep subinterval (recall that $\omega_0 = 0$, $\sigma_0 = \sigma$ and $\sigma_m = 0$). The three cases of the old charging scheme remain the same, except that Case 3 is restricted to $\text{IS}_{\text{w}}[j]$-intervals where IdleLonger wakes up at the end due to excessive inactive flow. We supplement Case 3 with a new scheme (Case 4) to handle $\text{IS}_{\text{w}}[j]$-intervals with wake-ups due to more than σ active jobs.

Charging scheme – Case 4. If OPT is sleeping (at any level except zero) throughout an $\text{IS}_{\text{w}}[j]$-interval $[t_1, t_2]$, and IdleLonger has accumulated more than σ active jobs at t_2, we consider two scenarios to charge OPT, depending on $n_{\text{o}}(t_1)$, the number of active jobs in OPT at t_1.
(a) Suppose $n_{\text{o}}(t_1) \geq \sigma_0$. We charge OPT the inactive flow of these jobs over $[t_1, t_2]$, which is at least $(t_2 - t_1)\sigma_0$. By Lemma 13, this charge is at least $\omega_j + \ell\sigma_j$.
(b) Suppose $n_{\text{o}}(t_1) < \sigma_0$. Note that OPT stays in a sleep-k state, for some $k \geq 1$, in the entire interval and uses static energy $(t_2 - t_1)\sigma_k$ during $[t_1, t_2]$. We charge OPT the sum of $(t_2 - t_1)\sigma_k$ and ω_k (in view of OPT's first wake-up after t_2, which must exist because new jobs have arrived within $[t_1, t_2]$). By Lemma 13, this charge is at least $\omega_j + \ell\sigma_j$.

In conclusion, we are able to charge OPT, for each $\text{IS}_{\text{w}}[j]$-interval, a cost at least $\omega_j + \ell\sigma_j$. Therefore, the sum of the charges to all $\text{IS}_{\text{w}}[j]$-intervals is at least $W + E_{\text{IW}}$. On the other hand, since $\text{IS}_{\text{w}}[j]$-intervals are all disjoint, the flow and idling (static) energy charged to OPT by Cases 1, 2, 3, 4(a) and 4(b) come from different parts of C^*. Recall that for Case 1, each OPT's wake-up is charged at most twice. Below, we argue that for Case 4(b), each OPT's wake-up is charged at most once (Lemma 23). Then we have $W + E_{\text{IW}} \leq C^* + 3W^*$.

Lemma 23. *With respect to the above charging scheme,* OPT *is charged by Case* 4(b) *with a cost of at most W.*

Proof. To prove that the total charge due to Case 4(b) is at most W^*, it suffices to show that each wake-up of OPT is charged at most once by an $\text{IS}_{\text{w}}[j]$-interval in Case 4(b). Suppose, for the sake of contradiction, there are two $\text{IS}_{\text{w}}[j]$-intervals $[t_1, t_2]$ and $[r_1, r_2]$ (with possibly different j), where $t_2 < r_1$, both charging to the same wake-up transition in Case 4(b). This implies OPT is sleeping during $[t_1, r_2]$. At time t_2, IdleLonger, as well as OPT, have at least $\sigma = \sigma_0$ active jobs. As OPT is sleeping during $[t_1, r_2]$, the number of active jobs of OPT at time r_1 is also at least σ_0 and Case 4(a) should have been applied for $[r_1, r_2]$, which is a contradiction. □

5.3 Summary

Summing up the inactive cost of IdleLonger and the working cost of AJC in Section 5.1, we can show that IdleLonger(AJC) is $O(1)$-competitive for flow plus energy. The exact competitive ratio is stated below. Recall that $\beta = 2/(1 - \frac{1-1/\alpha}{(\alpha+1)^{1/(\alpha-1)}})$ and $\zeta = \max\{4, \beta(1 - 1/\alpha)\}$.

Theorem 24. *In the bounded speed model with single or multiple sleep states, the total cost of* IdleLonger(AJC) *is at most* $\max\{\beta(2 + 1/\alpha) + \zeta + 1, 3\beta + 3\}$ *times of the total cost of* OPT.

Proof. By Theorem 22(ii), the inactive cost is at most $3C^* + 9W^*$. In addition, by Theorem 20, the working cost of IdleLonger(AJC) is $G_{\mathrm{W}} \leq (\beta(1 + 1/\alpha) + \zeta)C^* + (\beta - 2)F_{\mathrm{I}}$. By Property 10 and Theorem 22(i), $F_{\mathrm{I}} \leq W + E_{\mathrm{IW}} \leq C^* + 3W^*$. Therefore, the total cost of IdleLonger(AJC), comprised of the inactive cost and the working cost, is at most $(\beta(2+1/\alpha)+\zeta+1)C^* + (3\beta+3)W^*$. Therefore, the total cost of IdleLonger(AJC) is at most $\max\{\beta(2 + 1/\alpha) + \zeta + 1, 3\beta + 3\}$ times the total cost of OPT. □

6 Conclusion

In this paper we have initiated the study of flow-energy scheduling on a processor that allows speed scaling and multiple sleep states. We introduce a speed scaling algorithm AJC and a sleep management algorithm IdleLonger. AJC sets its speed based on the number of active jobs, and changes its speed only at job arrival or completion. This is in contrast to previous work [4,6], which changes the speed continuously over time. Under the simple speed scaling model (i.e., without sleep state), AJC itself gives an improvement in the competitive ratio for minimizing flow plus energy, without using extra speed (precisely, the competitive ratio is improved from $O((\frac{\alpha}{\ln \alpha})^2)$ to $O(\frac{\alpha}{\ln \alpha})$). In our analysis, we introduce a new form of potential functions based on the integral number of active jobs. Such a potential function allows us to reason directly on total flow. Very recently, Bansal et al. [5] adapt the analysis of such a potential function and show that AJC can be 3-competitive; the ratio is independent of α. They also generalize the result to work for arbitrary power functions.

The sleep management algorithm, called IdleLonger, works for a processor with one or multiple levels of sleep states. AJC together with IdleLonger are shown to be $O(\frac{\alpha}{\ln \alpha})$-competitive for flow plus energy under the general speed scaling model. Apparently, a sleep management algorithm and a speed scaling algorithm would affect each other, and analyzing their relationship and their total cost could be a complicated task. Nevertheless, the results of this paper stem from the fact that we can isolate the analysis of these algorithms.

References

[1] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49, 2007.

[2] J. Augustine, S. Irani, and C. Swamy. Optimal power-down strategies. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 530–539, 2004.

[3] K. R. Baker. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[4] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 409–420, 2008.

[5] N. Bansal, H. L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 693–701, 2009.

[6] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 805–813, 2007.

[7] L. Benini, A. Bogliolo, and G. de Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on VLSI Systems*, 8(3):299–316, 2000.

[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] H. L. Chan, W. T. Chan, T. W. Lam, L. K. Lee, K. S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 795–804, 2007.

[10] H. L. Chan, J. Edmonds, T. W. Lam, L. K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *Proceedings of International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 255–264, 2009.

[11] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[12] G. Greiner, T. Nonner, and A. Souza. The bell is ringing in speed-scaled multiprocessor scheduling. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 11–18, 2009.

[13] G. H. Hardy, J. E. Littlewood, and G. Polya. *Inequalities.* Cambridge University Press, 1952.

[14] S. Irani and K. Pruhs. Algorithmic problems in power management. *ACM SIGACT News*, 32(2):63–76, 2005.

[15] S. Irani, S. Shukla, and R. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems*, 2(3):325–346, 2003.

[16] S. Irani, S. Shukla, and R. K. Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4):41, 2007.

[17] A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 301–309, 1990.

[18] T. W. Lam, L. K. Lee, I. K. K. To, and P. W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 256–264, 2008.

[19] T. W. Lam, L. K. Lee, I. K. K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 647–659, 2008.

[20] T. Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[21] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[22] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.