Improving the Bounds of the Online Dynamic Power Management Problem

Ya-Chun Liang
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Kazuo Iwama
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Chung-Shou Liao
Department of Industrial Engineering and Engineering Management, National Tsing Hua University, Hsinchu, Taiwan

Abstract

We investigate the *power-down mechanism* which decides when a machine transitions between states such that the total energy consumption, characterized by execution cost, idle cost and switching cost, is minimized. In contrast to most of the previous studies on the offline model, we focus on the online model in which a sequence of jobs with their release time, execution time and deadline, arrive in an online fashion. More precisely, we exploit a different switching on and off strategy and present an upper bound of 3, and further show a lower bound of 2.1, in a dual-machine model, introduced by Chen et al. in 2014 [STACS 2014: 226-238], both of which beat the currently best result.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Online algorithm, Energy scheduling, Dynamic power management

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.28

Funding *Kazuo Iwama*: Supported by MOST Taiwan under Grants 110-2223-E-007-001, 110-2811-E-007-507 and KAKENHI Japan under Grant 16H02782.
Chung-Shou Liao: Supported by MOST Taiwan under Grants 110-2221-E-007-106-MY3 and 110-2622-8-007-017.

1 Introduction

Machines consume energy and many of them have the following property: namely machines have two states, ON and OFF, and change between the two states quite often. Furthermore, it requires a relatively large amount of energy to change from OFF to ON. For instance, a laptop goes to sleep or shutdown (OFF) if we do not touch it for, e.g., one hour, and comes back to ON if we press some key, but it is actually energy consuming. Copy machines also turn off automatically if there is no job for, e.g., ten minutes. Similarly, it consumes relatively large energy for heating them up when a new job arrives. There are many other similar examples, and it is obviously important to design online algorithms to minimize the energy consumption for this type of machines. Fortunately most of those problems are essentially equivalent to the well-known *ski-rental* problem and the optimal solution has been known for a long time both in deterministic and randomized cases [12, 16].

In this study, we investigate the online model of the dynamic power management problem, discussed by Irani et al. [14, 15] and Chen et al. [7], which aims at determining when to transition a machine between the states: *busy* or *idle* (ON) and *sleep* (OFF) to finish all input jobs such that the energy consumption is minimized. Suppose a machine M requires E units of energy to change its state from OFF to ON and needs ψ_σ units per unit of time to keep it ON, i.e. the *idle* cost. The goal is to finish all input jobs, arriving in an online fashion, while minimizing the total energy consumption. Let us consider a simpler model first, where jobs arriving in an online fashion are required to be immediately performed without any delay. One can observe that the optimal deterministic online algorithm of the ski rental problem can be applied to solve this model, with the competitive ratio of 2. The intuition is quite straightforward, by turning off the machine M E/ψ_σ units of time after its last job is finished. Note that the performance of an online algorithm is typically evaluated by competitive analysis. More precisely, the quality of an online algorithm is measured by the worst case ratio, called *competitive ratio*, which is defined to be the fraction of the cost of the online algorithm over that of the offline optimal algorithm, where the offline algorithm is aware of all jobs in advance.

However, when considering a more complicated model, in which each job j is given as (a_j, d_j, c_j), where a_j denotes the arrival time, d_j the deadline and c_j the execution time, a dramatic change does happen. In other words, the execution of each job is allowed to be postponed, which obviously increases the problem hardness. There have been actually not many studies in the literature for the online model [12]. Irani et al. [14,15] initiated the study of combining the above power-down mechanism with *dynamic speed scaling*, where the latter technique has been widely explored in the past decades [5,8,9,15,19]. The basic concept of dynamic speed scaling is that a machine's processing speed can be adjusted dynamically, and the power consumption rate is usually represented by a convex function of the processing speed in terms of time expense. They proposed the first online algorithm with a constant competitive ratio of $\max\{c_1c_2 + c_1 + 2, 4\}$, where c_1 and c_2 are some constant parameters in a given convex power function [15]. For example, if the power function is quadratic, the upper bound of competitive ratio is 8. Chen et al. [7] considered a similar model but without using dynamic speed scaling, where an online algorithm can use two machines M_1 and M_2 instead, under the same assumption that all input jobs must be finished by the offline optimal scheduler using a single machine. This assumption is actually the so-called *single machine schedulability* condition [10], which is characterized by the following lemma.

▶ **Lemma 1.** (Chetto et al. [10]) *For any set of jobs $\mathcal{J}$, they can be optimally scheduled on one machine using the* earliest-deadline-first *(EDF) principle. That is, the job with the earliest deadline is always selected for execution at any moment, if and only if the following condition holds:*

$$\text{For any time interval } (\ell, r)\text{, we have } \sum_{j: j\in\mathcal{J}, \ell\le a_j, d_j\le r} c_j \le r - \ell. \tag{1}$$

The condition is equivalent to the *earliest-deadline-first (EDF)* schedulability [19]. Here, we also remark that a feasible solution can be a preemptive schedule but the jobs can only be executed without migration. Chen et al. [7] gave an upper bound of 4 and a lower bound of 2.06 for the competitive ratio of this dual-machine problem. It is a significant contribution for online dynamic power management problems, but unfortunately, the above gap is not very small, for which there has been no improvement up to now.

Our Contribution. This paper improves both upper and lower bounds to 3 and 2.1, respectively, for exactly the same dual-machine model. Namely the competitive ratio gap between lower and upper bounds is improved from 1.94 in [7] to 0.9. Our algorithm has the same basic structure as the one in [7], but two critical differences, which exactly contributes the improvement, should deserve being mentioned. First, we delay jobs but turn on a machine earlier than its due time by a margin instead of "*energy-efficient anchor*" introduced in [7].

Second, our idle time after the machine is finished with its execution is not set to $\mathcal{B} = E/\psi_\sigma$, i.e. the so-called *break-even time*, but set to twice that value. Note that the break-even time has always been used for the class of similar problems including the famous ski-rental problem, as mentioned earlier. We hope this escape from the common tradition will help on several different occasions in the future.

Moreover, we use a standard math induction for the analysis, making our analysis significantly simpler than that of [7, 15], which will also contribute to further improvement of the bounds hopefully. In [7, 15], they made their competitive analysis by introducing what they call a "sleep interval" where the optimal offline schedule is OFF and an "awaken-interval" where their online algorithm is ON, and gave a key lemma saying a single sleep interval overlaps with at most two awaken intervals. This is clever since the analysis boils down to comparing a single (consecutively ON) interval of the online algorithm and that of the optimal offline schedule. However, a single such interval of the optimal offline schedule can still contribute to two such intervals of the online algorithm even if it can actually contribute to only one in many cases. Thus the analysis underestimates the cost of the optimal offline schedule. Our analysis splits the time line simply into "*phases*" that realizes the same intervals for an online algorithm and the optimal offline schedule, which makes it possible to use the standard math induction. Of course an online algorithm and the optimal offline schedule can execute different jobs in each phase, but that can be managed by considering "assets" and "liabilities" of jobs between phases. More details of the basic idea will be given in Sections 2 and 3.

Other Studies. In comparison with a few amount of research on *pure* dynamic power management, there have been relatively more studies which incorporate speed scaling into the power-down mechanism. Albers et al. [1] investigated the offline setting of speed scaling with a sleep state and presented a $\frac{4}{3}$-factor approximation algorithm. Antoniadis et al. [4] further improved the result to a fully-polynomial time approximation scheme. Considering the multi-machine systems, Demaine et al. [11] developed a polynomial-time algorithm based on dynamic programming. Albers et al. [2] then considered dynamic speed scaling with job migration. Very recently, Antoniadis et al. [3] presented a pseudo-polynomial time algorithm for a single machine based on a linear programming relaxation and a constant-factor approximation algorithm for the case of multiple machines. Readers may refer to the survey works [6, 13, 18] for more details.

2 Upper Bound

We formally introduce the dual-machine model proposed by Chen et al. [7]. Each machine requires a constant amount of energy to switch its state from *sleep* to either *busy* or *idle*, denoted by E. Suppose every machine uses a constant speed to execute jobs; that is, it consumes a constant amount of energy per unit of time, denoted by ψ_b when it is busy and ψ_σ when it is idle. The break-even time, as mentioned earlier, denoted by $\mathcal{B}$, is defined to be E/ψ_σ, which represents that when a machine is idle for $\mathcal{B}$ units of time, the energy consumption is equal to E, i.e. the energy consumption for switching on a machine from sleep to the other states. The standard assumption is that $\psi_\sigma \le \psi_b$. Another assumption in this study is that $E = \psi_b = 1$ (and $\psi_\sigma \le 1$). If E is a positive integer k, we can use our model by changing a unit time from 1 to $1/k$ (and changing the job and idle length accordingly). Thus our model does not lose any generality and this setting contributes to significantly simplified expositions. Recall that our input always satisfies Condition (1) in this model, where a feasible schedule allows job preemption but no migration.

Before introducing our online algorithm, we have some more notation. The dual-machine model has two machines: one of them is called M_P (a *primary machine*) and the other M_S (a *secondary machine*). We basically use the primary machine as far as possible and turn on the secondary machine if necessary. That is, we set M_P to be the main machine, and M_S to be the backup machine. When a job arrives, we always turn on M_P first and decide to turn on M_S only if it finds out that M_P is not able to finish all the pending jobs before their deadlines; i.e., M_P is overloaded. In other words, M_P and M_S are logical names and the two (physical) machines may swap their names in our algorithm. Let Q_{M_P} and Q_{M_S} denote the job queues for M_P and M_S, respectively, which are already scheduled using EDF; namely, the jobs in Q_{M_P} and Q_{M_S} are being executed continuously on M_P and M_S, respectively. We use Q to denote a queue for jobs not yet scheduled. If a job in Q_{M_P} or Q_{M_S} is finished, it is removed from the queue. Also, let $c'_j(t)$ denote the remaining execution time of job j at time t, and $W(t, t^\dagger) = \sum_{j \in Q(t,t^\dagger)} c'_j(t)$ denote the total remaining execution time in time interval $(t, t^\dagger)$, where $Q(t, t^\dagger)$ denotes the subset of jobs that have not yet finished their execution up to time t while their deadlines are not larger than $t^\dagger$, where $t^\dagger > t$. When a new job j', given by $(a_{j'}, d_{j'}, c_{j'})$, is arriving, there are two cases. One is that the current schedule for Q_{M_P} can accommodate its execution. If so, j' is inserted into Q_{M_P}, which is rescheduled by EDF (we say M_P is available). The other case is that there is no room for j' in Q_{M_P} (if jobs in Q_{M_P} would not have been delayed, this cannot happen because of the single machine schedulability assumption). We call such a j' *urgent*. If an urgent job comes, then M_S is turned on and j' is executed on M_S immediately (where M_P continues executing the already assigned jobs). We then swap M_P and M_S at this point of time, which means the original M_S can act as a new primary machine. Table 1 shows our algorithm, denoted by $\mathcal{A}$. Note that in the following, we use ALG to denote an online algorithm, OPT an offline optimal scheduler and CR a competitive ratio for simplicity.

Table 1 Algorithm $\mathcal{A}$.

```
Initially assign $M_P$ and $M_S$ to two machines arbitrarily.
The input satisfies Condition 1 of Lemma 1.
At any time $t$, $\mathcal{A}$ proceeds as follows:

1. Execution of jobs:
   a. No machines are on.
      If there exist $t^\dagger$ and $t^*$ such that $W(t^\dagger, t^*) \geq t^* - t^\dagger$ and $t \geq t^\dagger - u$, turn on one
      machine ($M_P$) and move all jobs in $Q$ to $Q_{M_P}$.
   b. Only $M_P$ is on and a new job $j$ comes.
      If there is no $t^*$ such that $W(t, t^*) > t^* - t$ for jobs in $Q_{M_P} \cup \{j\}$ (i.e., $M_P$ is
      available), add $j$ to $Q_{M_P}$ and reschedule it.
      Otherwise, turn on the other machine ($M_S$) and add $j$ to $Q_{M_S}$ that is empty.
      After that switch $M_P$ and $M_S$.
   c. $M_S$ is on and a new job $j$ comes.
      If $M_S$ is available, add $j$ into $Q_{M_S}$ and reschedule it.
      Otherwise move it to $Q_{M_P}$ (it is guaranteed that $M_P$ is available).
2. Idle state:
   a. $M_S$ never has an idle state.
      That is, we immediately turn off $M_S$ once $Q_{M_S}$ becomes empty.
   b. $M_P$ becomes idle once $Q_{M_P}$ becomes empty.
      We turn off $M_P$ when its total idle time becomes $2/\psi_\sigma$.
```

Algorithm $\mathcal{A}$ has two key gadgets. First, we present a new notion of "*margin*", as a positive constant u, for delaying a job. That is, we start a job earlier than its due time by u. Fig. 1 shows examples to illustrate the idea. Here, we use a solid line segment from time a_j to d_j for each job $j = (a_j, d_j, c_j)$, and a gray box to represent its execution time c_j. Below those line segments and boxes, we illustrate how those jobs are executed by ALG and OPT. Dashed line segments show the idle time of ALG and dotted line segments the idle time of OPT (if any).

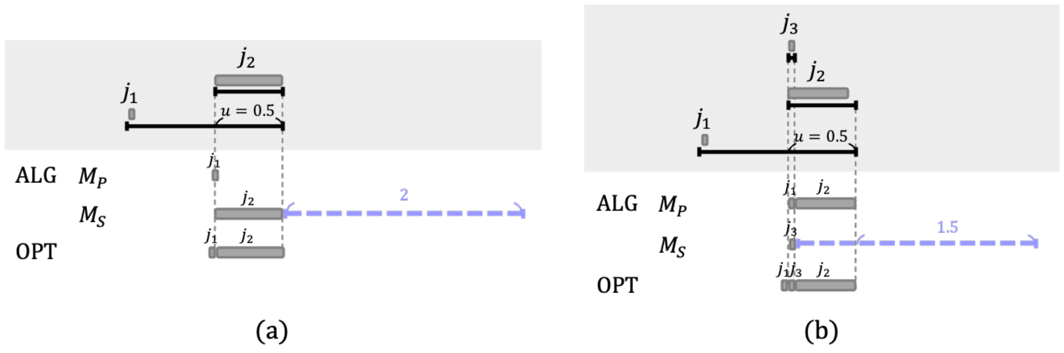

Figure 1 Two examples for illustrating the notion of margin.

As shown in Fig. 1 (a), j_1, having a tiny execution time of ϵ, starts at $d_1 - u - \epsilon$. Here we let $u = 0.5$ to get close to the optimal setting although its perfect setting seems hard. (We will show more details about the margin setting in the next section.) Thus, it is necessary that job j_2 that provokes M_S's turn-on has an execution time of at least 0.5 (otherwise it can interrupt j_1 and be inserted into M_P). See Fig. 1 (b) for a more complicated scenario. Now j_2 is a bit smaller than the one in (a) and can be inserted into the (old) M_P's execution gap caused by the margin u. Then a tiny j_3, which is urgent, makes the secondary machine ON, but its idle time can start 0.5 earlier than before. Note that the idle time of a machine is typically set to $\mathcal{B}$ after the last job is finished, where $\mathcal{B} = E/\psi_\sigma = 1$ since we are temporarily assuming $\psi_\sigma = 1$. It is actually a popular setting for the idle time that has been proved optimal in similar situations including the ski-rental problem. In contrast, the second key gadget of Algorithm $\mathcal{A}$ is that we set this value to twice, giving rise to that OPT has an interval of 1.5 from the end of j_2 (i.e. d_2) to the end of the idle state of M_P. On the other hand, recall that OPT has an interval of 2 in the previous scenario (see Fig. 1 (a)). One can observe that OPT could use this interval to execute the pending jobs and thus the longer the better for this interval. However, we have to use 1.5 instead of 2 for the analysis in Section 3.

Tight Analysis of Algorithm S [7]. The two gadgets of Algorithm $\mathcal{A}$ reveal the improvement of the upper bound in some sense. Here we recall Chen et al.'s 4-competitive algorithm, called *Algorithm* ***S***, and conduct tight analysis of their algorithm to clarify the merit of our proposed gadgets. That is, we present a tight example to show its worst competitive performance.

Basically, the main structure of Algorithm **S** is similar, but they let every job j be associated with a parameter $h_j = \max\{a_j, d_j - \lambda\mathcal{B}\}$, i.e. its *energy-efficient anchor*, to determine when the main machine M_P should be switched on for a pending job j, where λ is a constant (setting to be one in [7]). Set $\psi_b = \psi_\sigma = 1$, and $E = \mathcal{B} = k$, where k is a sufficiently large value, for simplicity. As shown in Fig. 2, we let $c_1 = c_3 = \mathcal{B}$ and $c_2 = c_4 = \epsilon$. Algorithm **S** turns on M_P at h_1. Since j_2 arrives at the same time, i.e. $a_2 = h_1$, obviously,

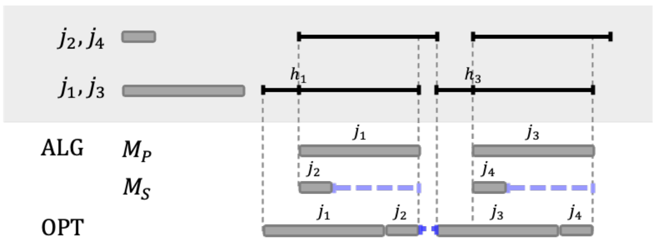

Figure 2 Tight example for Algorithm **S** [7].

M_P cannot finish j_2 before its deadline. Therefore, M_S is switched on to execute j_2. One can observe that the design of the margin in Algorithm $\mathcal{A}$ can provide the flexibility to escape from the scenario.

Next, the turn-off strategy of Algorithm **S** is as follows: if both machines are ON, M_P (new M_S) is turned off as soon as it becomes idle. Otherwise, if only one of the machines is ON, then when the machine becomes idle, it is switched off once the length between the current time and the moment when M_P was turned on has reached $\mathcal{B}$. In this example, M_S remains idle until M_P keeps ON for $\mathcal{B}$ units of time, resulting in M_P and M_S being switched off at the same time. One can observe that if the job executed on M_S is tiny, the malicious adversary can punish the turn-off strategy since it turns off M_S too early. We design the same worst-scenario for jobs j_3 and j_4 and consider the performance. As a result, the total energy cost of ALG is $4E + 4\mathcal{B} - 2\epsilon = (4k - \epsilon) \cdot 2$. By contrast, in the offline optimal solution, it turns on a single machine at a_1 and keep it ON until all the input jobs are finished. The minimum energy cost is $E + 2\mathcal{B} + 2\epsilon + \epsilon' = k + 2(k + \epsilon) + \epsilon'$.

It is not hard to see that, if we consider the above case of four jobs to be one round, and the energy cost of Algorithm **S** for $\frac{r}{2}$ rounds, where r is a multiple of 2, is $(4k - \epsilon) \cdot r$, while the energy cost of OPT is $k + r(k + \epsilon) + (r - 1) \cdot \epsilon'$. Letting the values of ϵ and ϵ' be very small, the ratio becomes

$$\frac{4kr}{k + kr} = \frac{k \cdot 4r}{k \cdot (r + 1)} = \frac{4r}{r + 1}.$$

When the value of r is sufficiently large, the CR approaches 4. The tight analysis reveals the advantages of our designed gadgets in Algorithm $\mathcal{A}$.

3 Analysis of the Algorithm

For analysis, we need to show that (i) $\mathcal{A}$ is correct, namely it can execute any sequence of jobs arriving under the single machine schedulability condition and (ii) its CR is at most 3. We mostly focus on (ii). (i) is not hard and the following observation should be enough to see every job is executed:

(1) Suppose when a new job j comes at time t, no machine is ON. Then if the condition of 1-a for $Q \cup j$ is not met, j just goes to queue Q. Once j enters Q, it must be executed eventually by 1-a. If the condition of 1-a is met, then all the jobs in Q go to Q_{M_P} and we go to 1-b. Here j is inserted to Q_{M_P} (if possible) or is executed on M_S that is turned on at t. Thus j is executed.

(2) Suppose only M_P is already ON but M_S is OFF when j comes. As (1), j is inserted to Q_{M_P} or executed on M_S that is turned on at t.

(3) Suppose M_S is ON when j comes. If M_S is available, j is executed on it. Otherwise, note that when M_S is turned on for the last time, all the jobs in Q go to M_S (recall M_P and M_S are swapped) and while M_S is on, all newly coming jobs are executed on M_P due to EDF. This execution is possible since if the whole input sequence satisfies Condition 1 of Lemma 1, its arbitrary suffix obviously does, too. j is one of them and must be executed.

To prove the CR, we first define a *phase*. Suppose the system changes its state from both machines OFF to at least one machine ON at t_1 and returns to the state of both machines OFF at t_e. Also let $a_1, a_2, \ldots, a_k$ be the arrival time of the jobs executed during t_1 to t_e and let t_0 be $\min\{a_1, \ldots, a_k\}$. Then the time slot from t_0 to t_e is called a phase. Note that an entire execution of $\mathcal{A}$ consists of some phases. Let $P(i)$ be the i'th phase.

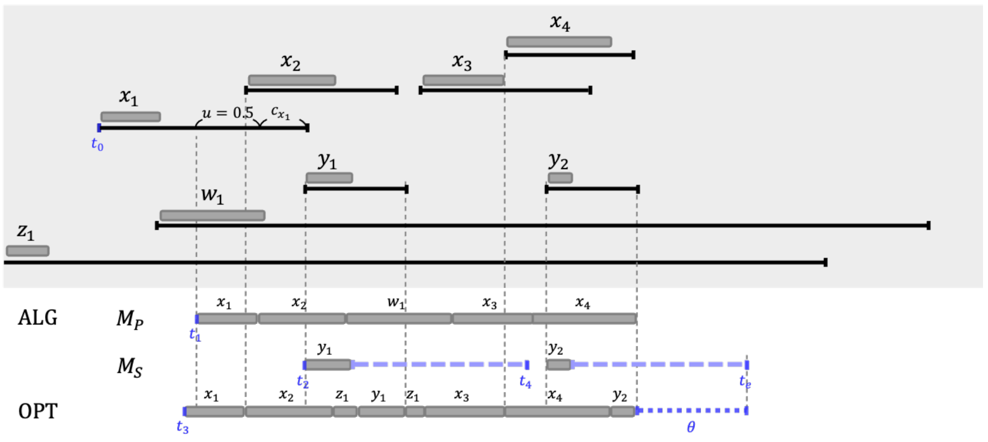

Figure 3 A single phase. Now a period of idle time is $2/\psi_\sigma$. M_P and M_S are swapped at a_{y_1}.

Fig. 3 illustrates how a single phase looks like. x_1 and x_2 are executed on M_P, then an urgent y_1 comes and executed on the new M_P (so w_1, x_3 and x_4 are executed on the new M_S). w_1 is a job moved from Q to Q_{M_P} when x_1 starts (more precisely, w_1 is scheduled after x_1, but x_2 is inserted later due to the EDF principle). x_3 and x_4 are added to Q_{M_S}. y_2 is a job that cannot be inserted to Q_{M_S} because of the existence of x_4; it goes to M_P. Thus M_P can be OFF for some period of time during a single phase (but M_S should be ON during that period by definition). The earliest arrival time of the jobs $\mathcal{A}$ executes in this phase is that of x_1, which is the start time, t_0, of this phase. Note that t_e at which the phase ends is the moment when the idle time after y_2 expires. An example of the OPT's execution sequence is given at the bottom of the figure. It must execute x_1, x_2, x_3, x_4, y_1 and y_2 since each of them has its arrival time and deadline within the phase. w_1 is not executed by OPT in this example. By contrast, z_1 which is performed by OPT is not executed by $\mathcal{A}$ (can be executed in the previous phase), so its arrival time is not counted to determine the beginning of the phase. (Note that OPT and $\mathcal{A}$ as well, may execute each job separately, so we have two z_1's in the figure.) Fig. 3 is also used for the later analysis of the competitive ratio. A phase is called a single-machine phase if only one machine (M_P) is ON in that phase and a dual-machine phase otherwise.

Our proof of the CR uses a math induction. Since $\mathcal{A}$ is deterministic, the set of jobs that are executed by $\mathcal{A}$ in $P(i)$ is uniquely determined once the input is given, which we denote by $J_A(i)$. For the jobs executed by OPT in $P(i)$, the situation is less clear; jobs whose arrival time and deadline are both within $P(i)$ must be executed, but jobs such that only one of them is within $P(i)$ may or may not be executed even partially. Furthermore, OPT may execute some jobs outside phases, namely while no machine is ON. Fix an arbitrary execution sequence $S(i)$ of OPT in this phase. Then we define the following parameters used in the induction.

(1) $\alpha(i)$ ($\lambda(i)$, and $\delta(i)$, resp.) = the total execution time of the jobs executed by both $\mathcal{A}$ and OPT (only by $\mathcal{A}$ and only by OPT possibly partially, resp.)

(2) $A(i)$ is the cost of $\mathcal{A}$ in $P(i)$ that includes $\alpha(i)$, $\lambda(i)$, turn-on costs and idle costs.

(3) $O_f(i)$ is a lower bound for the cost of OPT, namely it includes $\alpha(i)$, $\delta(i)$ and idle costs if any. Note that it does not include the turn-on cost when $S(i)$ starts since OPT may not need it depending on its state at the end of the previous phase, but does include one(s) if $S(i)$ includes turn-off and turn-on in the middle of the phase. $O_n(i)$ is similar but we impose the condition that OPT is ON at the end of $P(i)$.

For instance, consider $P(i)$ whose execution sequence looks like Fig. 1 (a). Then $A(i) = 4.5$, $O_f(i) = 0.5$ and $O_n(i) = 2.5$, where 0.5 is for the execution of jobs and $2 = (2/\psi_\sigma)\psi_\sigma$ is the idle cost to keep it ON until the end of the phase.

▶ **Theorem 2.** *$\mathcal{A}$ is correct and its CR is at most 3 for $u = 0.5$.*

Proof. We omit the first part (see the previous observation). For the CR, we fix an arbitrary execution sequence S ($S(i)$ is its subsequence associated with $P(i)$) for the entire execution sequence of OPT and prove two lemmas.

▶ **Lemma 3.** *The following (2) and (3) hold for each phase for $r = 3$.*

$$rO_f(i) - A(i) \geq \delta(i) - \lambda(i) - r. \tag{2}$$
$$rO_n(i) - A(i) \geq \delta(i) - \lambda(i). \tag{3}$$

The proof will be given later. Let $O(i)$ be the (real) cost of OPT under the sequence $S(i)$ in $P(i)$. Also let m be an integer less than or equal to the number of phases.

▶ **Lemma 4.** *For $r = 3$, we have*

$$\sum_{i=1}^{m}(rO(i) - A(i)) \geq \begin{cases} \sum_{i=1}^{m}(\delta(i) - \lambda(i)) & \text{if OPT is OFF at the end of } P(m). \quad (6) \\ \sum_{i=1}^{m}(\delta(i) - \lambda(i)) + r & \text{if OPT is ON at the end of } P(m). \quad (7) \end{cases}$$

Proof. Suppose OPT is OFF at the end of $P(i)$. Then $O(i)$ should be at least $O_f(i)$ since the latter is a lower bound and similarly for $O_n(i)$ if OPT is ON at the end of the phase. For $m = 1$, note that OPT must spend the turn-on cost of 1 that is not included in either $O_f(i)$ or $O_n(i)$. Therefore if OPT is OFF at the end of the phase, we have

$$rO(1) - A(1) \geq rO_f(1) + r - A(1) \geq \delta(1) - \lambda(1)$$

by Lemma 3. Otherwise, if OPT is ON at the end of the phase, we have

$$rO(1) - A(1) \geq rO_n(1) + r - A(1) \geq \delta(1) - \lambda(1) + r$$

similarly. Now suppose the lemma is true for $m' = m - 1$. Then to prove that the lemma also holds for $m' = m$, we consider four cases and define $C \to D$ as the states of OPT at the end of $P(m-1)$ and $P(m)$ respectively: (i) OFF $\to$ OFF, (ii) OFF $\to$ ON, (iii) ON $\to$ OFF and (iv) ON $\to$ ON. For case (i), OPT must pay the turn-on cost in $P(m)$, so

$$\begin{aligned}\sum_{i=1}^{m}(rO(i)-A(i)) &= \sum_{i=1}^{m-1}(rO(i)-A(i)) + rO(m) - A(m)\\ &\geq \sum_{i=1}^{m-1}(\delta(i)-\lambda(i)) + 0\cdot r + rO_f(m) - A(m) + 1\cdot r\\ &\geq \sum_{i=1}^{m-1}(\delta(i)-\lambda(i)) + \delta(m) - \lambda(m)\\ &= \sum_{i=1}^{m}(\delta(i)-\lambda(i))\end{aligned}$$

by the induction hypothesis and Lemma 3. Here 0 and 1 before r are for handling the four cases. Since the current case is OFF $\to$ OFF, the first 0 means formula (4) does not have r on the right-hand side and the second 1 means that OPT must turn on in $P(m)$. The other cases are similar (just 0 and 1 before r change) and may be omitted. ◀

If there are m phases in total, it must be that $\sum_{i=1}^{m}(\delta(i)-\lambda(i)) = 0$ and thus the theorem is proved. What remains is to prove Lemma 3.

Proof. (Proof of Lemma 3) Suppose $x_1 = (a_1, d_1, c_1)$ is the first job executed in some phase P starting from t_0 and ending at t_e. Then since x_1 is executed in P, $t_0 \leq a_1$ (there may be another job executed in P and having an earlier arrival time). Also, since the ending time of x_1's execution is to be $d_1 - 0.5$ due to the delay (or not delayed at all if this amount of delay is impossible) and we have a mandatory idle time, $2/\psi_\sigma \geq 2$, after its (or a later job's) execution, it must be that $d_1 \leq t_k$. This means the period (a_1, d_1) is included in P, meaning OPT also executes x_1 in P. Thus, in each phase, both $\mathcal{A}$ and OPT execute at least one job. Also it turns out, by definition, that $\mathcal{A}$ never executes a job outside phases.

Note, however, that OPT may execute some job, say x, outside phases (this happens, e.g., if u_1 in Fig. 4 is executed by OPT after this phase and before the next phase). If that happens, we consider that x is executed in a "special" phase, $P(i')$, by extending the definition of a phase. Note that this phase has $A(i) = \lambda(i) = 0$ and both $O_f(i)$ and $O_n(i)$ are at least $\delta(i)$, so (2) and (3) obviously hold. A special phase may continue to/from a neighboring (normal) phase.

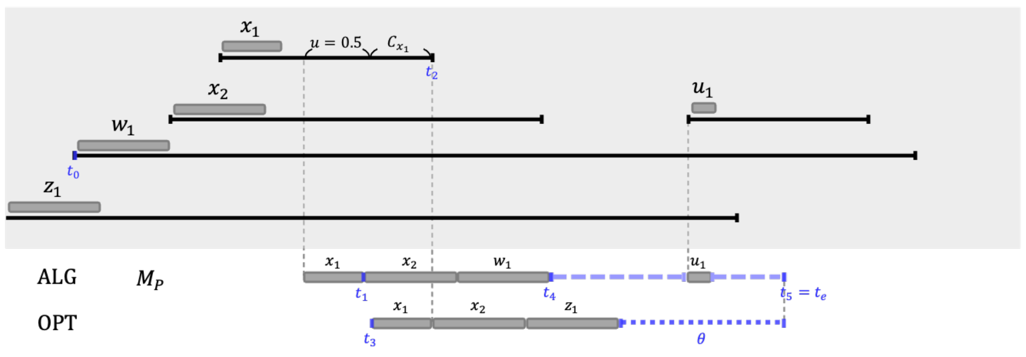

Figure 4 A single-machine phase. Recall a period of idle time of $\mathcal{A}$ is $2/\psi_\sigma$.

We first prove the lemma for a single-machine phase $P(i)$. See Fig. 4 which illustrates the execution sequences of $\mathcal{A}$ and OPT as with Fig. 3. In this figure, the phase starts at the arrival time of w_1 and ends when ALG's machine (M_P) turns off. x_1 is the first job executed in this phase, and x_2 is a job whose arrival time and deadline are both within this phase (so must be executed by both $\mathcal{A}$ and OPT in this phase). z_1 has only its deadline, and u_1 and w_1 have only their arrival times within the phase. Thus there are several different types of jobs, but what is important is whether or not each job is executed in this phase. In the example of this figure, x_1 and x_2, called *type-AO* jobs, are executed by both $\mathcal{A}$ and OPT, w_1 and u_1, *type-A*, by only $\mathcal{A}$, and z_1, *type-O*, only by OPT. Each type can include an arbitrary number of jobs, but as will be seen in a moment, those numbers are not important. So we will progress our analysis by using these five jobs, x_1, x_2, w_1, u_1, z_1, for a while and will mention the generalization after this analysis. It should also be noted that a single job may be executed by OPT in two or more phases. If this happens, we divide that job into two or more parts and allow each part can have a different job type, if necessary.

Set the following moments (see the figure) of time: t_1 and t_2: the ending time of execution and the deadline of x_1, respectively. t_3: the time when OPT becomes ON after time t_0 to execute x_1 or maybe another job. t_4 and t_5: the start and ending times of the idle state of $\mathcal{A}$, respectively. Thus one can observe that neither the number of jobs in each type nor their execution sequence is important to define these times, except x_1 that is first executed. For the five jobs, it turns out that $A(i) = c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 1 + 2$ (1 is the turn-on cost and 2 is the idle cost; recall once $\mathcal{A}$ enters an idle state, it continues until its total time becomes $2/\psi_\sigma$, i.e., until its total idle cost becomes $(2/\psi_\sigma)\psi_\sigma = 2$), and $O_f(i) \geq (c_{x_1} + c_{x_2} + c_{z_1})$. OPT's execution of some job may exceed the border of the phase. If that happens, as mentioned before, we can partition that job into two parts, the first one ends at the end of the phase and the second one is to be the remaining part, which is executed in the following special phase.

Now we have

$$\begin{aligned} rO_f(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{z_1}) - (c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 3) \\ &= 2(c_{x_1} + c_{x_2} + c_{z_1}) + c_{z_1} - (c_{w_1} + c_{u_1}) - 3 \\ &\geq c_{z_1} - (c_{w_1} + c_{u_1}) - 3. \end{aligned}$$

Thus (2) holds for any nonnegative values of c_{x_1} through c_{u_1}. Similarly, letting θ be the idle time of OPT (if any) that keeps the OPT's machine ON until t_5 (OPT can turn off once and turn on at or before t_5, but OPT would then need an extra turn-on cost and our proof becomes easier), we have $O_n(i) \geq c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta$ and hence

$$\begin{aligned} rO_n(i) - A(i) &\geq 3(c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta) - (c_{x_1} + c_{x_2} + c_{w_1} + c_{u_1} + 3) \\ &\geq 2(c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta) + c_{z_1} - (c_{w_1} + c_{u_1}) - 3. \end{aligned}$$

Here we have the following claim.

▷ Claim 5.

$$c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta \geq 1.5$$

Proof. We have a sequence of time relations:

(i) $t_2 - t_1 \leq 0.5$ due to the margin,
(ii) $t_3 \leq t_2$ (x_1 must be executed before its deadline),
(iii) $t_1 \leq t_4$ (obviously),

(iv) $t_3 - t_4 \le t_3 - t_1$ (by (iii)) $\le t_2 - t_1$ (by (ii)) ≤ 0.5 (by (i)),
(v) $t_5 - t_4 \ge 2/\psi_\sigma$ (the total idle time),
(vi) $t_4 - t_5 \le -2/\psi_\sigma$ (inversion of (v)), and
(vii) $c_{x_1} + c_{x_2} + c_{z_1} + \theta = t_5 - t_3 = -(t_3 - t_5) \ge 2/\psi_\sigma - 0.5$ (by (iv)+(vi)).

We may need a bit more explanation for (i). Recall that if x_1 is delayed and nothing happens, $t_2 - t_1$ is exactly 0.5. However, some (small) job that comes after x_1's execution has started may interrupt x_1 and be inserted. It should also be noted that if $d_{x_1} - (a_{x_1} + c_{x_1}) < 0.5$, then x_1 is not delayed in the first place. Now we have from (vii) (note $\psi_\sigma \le 1$)

$$c_{x_1} + c_{x_2} + c_{z_1} + \psi_\sigma\theta \ge \psi_\sigma(c_{x_1} + c_{x_2} + c_{z_1} + \theta) \ge 2 - 0.5\psi_\sigma \ge 1.5. \qquad \triangleleft$$

Thus we have

$$rO_n(i) - A(i) \ge 2 \times 1.5 + c_{z_1} - (c_{w_1} + c_{u_1}) - 3 = c_{z_1} - (c_{w_1} + c_{u_1}),$$

meaning (3) is also true. In general, we have more (or less) jobs in each job type. However, one can see that the definitions of t_1 through t_5 are not affected by that (only the first job, x_1 is important). Also we can simply replace c_{x_1} by the sum of execution times of type-AO jobs and similarly for type-A and type-O jobs. Thus the extension to the general case is straightforward and details may be omitted.

We next consider a dual-machine phase as shown in Fig. 3, where there are two executions, y_1 and y_2 on M_P while M_S is busy. (Recall M_P and M_S are swapped when M_S turns on. Now M_S is busy and it may not be available for urgent jobs.) Let the first execution be E_1 and the second one E_2. We first consider the case that E_2 does not exist. Namely, there is no x_3, x_4 or y_2 and OPT has an idle time after having executed z_1. Thus the phase would be ending at the end of the idle state following the execution of y_1 and we prove the lemma for this case first. As before, we use specific examples for jobs to be executed in this phase, x_1, x_2 and y_1 for type-AO, w_1 for type-A and z_1 for type-O. Since the M_S's turn-on is provoked, there must be a set S of jobs such that their execution on M_P is impossible during the period from some t'_1 to some t'_2. We assume that $S = \{x_1, x_2, y_1\}$, where $t'_1 = t_1$ and t'_2 is the deadline of y_1. What we do for the generalization is the same as before, namely we replace $\{x_1, x_2, y_1\}$ by the real jobs in S, maybe more jobs for type-AO, replace w_1 by real type-A jobs, and z_1 by real type-O jobs.

Now we start with definitions of time moments (see the figure) as before. t_1, t_2 and t_3: the time when M_P and M_S become on and the time OPT becomes busy, respectively. t_4: the time when the idle state of M_P is ended and this is the end of the phase, too. For those five jobs, we have $A(i) = c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 2 + 2$ (we now need to turn on both M_P and M_S), $O_f(i) \ge (c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1})$, and $O_n(i) \ge (c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \psi_\sigma\theta)$, where θ is the idle time (if any) to keep the OPT's machine on until t_4.

To prove formulas (2), we have (recall $\psi_\sigma \le 1$)

$$\begin{aligned} rO_f(i) - A(i) &\ge 3(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) - (c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 4.0) \\ &= 2(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) + c_{z_1} - c_{w_1} - 4.0. \end{aligned}$$

Here we can claim that $c_{x_1} + c_{x_2} + c_{y_1} \ge 0.5$. The reason is that there is a margin of 0.5 between the end of the execution of x_1 and its deadline (recall again if realizing this margin is impossible, M_S would not have been turned on). So if $c_{x_1} + c_{x_2} + c_{y_1} < 0.5$, then it follows of course $c_{x_2} + c_{y_1} < 0.5$, which means that x_2 and y_1 could have been executed using this margin time (they can interrupt the execution of x_1) on M_P, resulting in a contradiction. Thus $rO_f(i) - A(i) \ge 2 \times 0.5 + c_{z_1} - c_{w_1} - 4.0 = c_{z_1} - c_{w_1} - 3$ and we are done for (2).

For formula (3), we have ($\psi_\sigma \le 1$)

$$\begin{aligned} rO_n(i) - A(i) &\ge 3(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \psi_\sigma\theta) - (c_{x_1} + c_{x_2} + c_{y_1} + c_{w_1} + 4.0) \\ &= 2((c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1}) + 3\psi_\sigma\theta + c_{z_1} - c_{w_1} - 4.0 \\ &\ge 2\psi_\sigma(c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \theta) + c_{z_1} - c_{w_1} - 4.0. \end{aligned}$$

We again have the following time relations:

(i) $t_1 < t_2$ (M_S never turns on before M_P),

(ii) $t_3 < t_1$ (see below),

(iii) $t_4 - t_3 \ge t_4 - t_1$ (by (ii)) $\ge t_4 - t_2$ (by (i)) $\ge 2/\psi_\sigma$.

For (ii) recall that M_P cannot execute x_1, x_2 and y_1 from time t_1. Since OPT does execute those jobs by a single machine, it should have started their execution before t_1, meaning $t_3 < t_1$. Since $c_{x_1} + c_{x_2} + c_{y_1} + c_{z_1} + \theta = t_4 - t_3$, we finally have

$$rO_n(i) - A(i) \ge 2\psi_\sigma(2/\psi_\sigma) + c_{z_1} - c_{w_1} - 4.0 = c_{z_1} - c_{w_1},$$

and (3) is proved. The generalization to arbitrary number of jobs is the same as before and may be omitted.

Finally we consider the case that E_2 (or even more) exists, which can be simply done by considering that a new virtual phase, just an interval starting from t_4 and ending at t_e where we do the similar calculation as above. Note that $\mathcal{A}$ needs only one turn-on cost and so the energy consumption of the new virtual phase is the job execution costs +3 instead of +4 above. Therefore we do not need to lower bound the cost for executing type-AO jobs (as we did for $c_{x_1} + c_{x_2} + c_{y_1}$ above). Also, since the new virtual phase obviously includes the whole idle time of M_S, the proof for formula (3) is also straightforward. Details may be omitted.

Thus Lemma 3 is proved. ◀

And the proof of Theorem 2 is also concluded. ◀

4 Lower Bound

In this section we give our second result, a lower bound of 2.1, which improves 2.06 obtained in [7]. Throughout this section we set $\psi_\sigma = 1$ (which seems the worst case for online algorithms).

▶ **Theorem 6.** *The CR of any online algorithm for the online DPM job scheduling problem is at least 2.1.*

Proof. Our strategy is quite simple and standard. The adversary, Adv, gives requests, one by one, so that each request blames the last action of the algorithm. Fix an arbitrary algorithm ALG and a target CR lower bound, α, we want to prove. Before the formal proof, we briefly look at the basic strategy of Adv. Recall that $\psi_\sigma = 1$ in this proof.

The first request by Adv is $j_1 = (0, d_1, c_1)$, where its execution time c_1 is tiny and d_1 should not be too small. ALG must execute j_1 at some time before d_1, say at $d_1 - x_1$ on one of the two machines, say M_1 (Fig. 5). Here we have two cases.

Suppose that x_1 is relatively large. Then Adv sees how long ALG stays in the idle state after the execution of j_1. We can assume without loss of generality that this idle state continues at least until d_1 and may be more for additional y_1 as shown in Fig. 5. (If the idle state ends before d_1, then Adv immediately gives another tiny request with the same deadline d_1. This situation is similar to that ALG postpones its execution of j_1 up to this

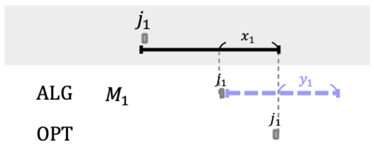

Figure 5 Illustration of the incoming job j_1.

time. Since ALG has already spent a turn-on cost of 1, it is not hard to show that Adv's job becomes easier. We omit details.) From the OPT side, it suffices to execute a tiny c_1 at d_1. Here ALG cannot have a long y_1 since the CR at this moment is $\frac{1+c_1+x_1+y_1}{1+c_1}$ (both ALG and OPT need a turn-on cost of 1, since this is the beginning of the game), which may exceed α and the game would end. So y_1 is relatively small, for which Adv gives a similar request right after the idle time expires. As shown in Fig. 6, OPT can manage these two requests by being ON from d_1 to a_2, thus blaming the small value of y_1.

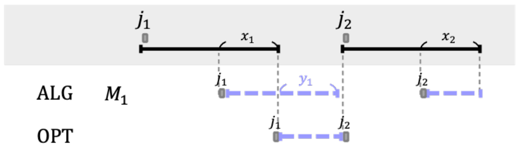

Figure 6 Illustration of the incoming job j_2.

What if x_1 is relatively small? Then Adv gives a request j_2 as shown in Fig. 7 immediately after ALG has started the execution of j_1. Note that j_2 has the same deadline as j_1 (i.e. $d_1 = d_2$) and its execution time is x_1 (i.e. $c_2 = x_1$). Thus ALG cannot execute j_2 on M_1 and it turns on M_2 meaning ALG has to pay a new turn-on cost. OPT can manage this by being on from slightly before $d_1 - x_1$ to d_1, thus blaming the shortness of x_1.

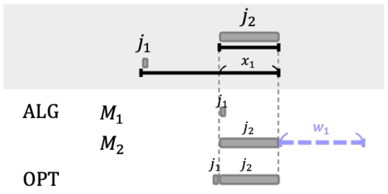

Figure 7 Case A with j_1 and j_2.

Now we start our formal proof. As mentioned above, Adv has basically two different strategies depending on the first response of ALG, namely $x_1 \leq \beta$ (Case A) and $x_1 > \beta$ (Case B), where β is some constant to be optimized later. Let us look at Case A first. As shown above, it proceeds to the situation illustrated in Fig. 7 and now ALG selects its new idle time w_1 on M_2. Of course M_1 can also have some idle time. However, as seen in a moment, our Adv always gives a next request after both machines become OFF. Therefore without loss of generality, we can assume only one machine which is busy until later than the other enters an idle state and the other turns off immediately when it finishes all the assigned requests. At the moment of Fig. 7, the CR is $\frac{2+c_1+x_1+w_1}{1+c_1+x_1}$ (recall $\psi_\sigma = 1$). Here we set $c_1 = 0$ for the exposition (and will do the same for a tiny execution time in the remaining part, too). This does not lose much sense since we can make c_1 arbitrarily small and it always appears as a sum with far greater values. Thus our current CR is $\frac{2+x_1+w_1}{1+x_1}$. If this value is greater than α, Adv has achieved its goal and the game ends. For the game to continue it must be $\frac{2+x_1+w_1}{1+x_1} \leq \alpha$. This implies

$$w_1 \leq \alpha(1+x_1) - (2+x_1) \leq \alpha + (\alpha-1)x_1 - 2 \leq \alpha + (\alpha-1)\beta - 2 \ (= f_1)$$

since $\alpha \geq 1$ and $x_1 \leq \beta$. Let f_1 be the value of the right-hand side.

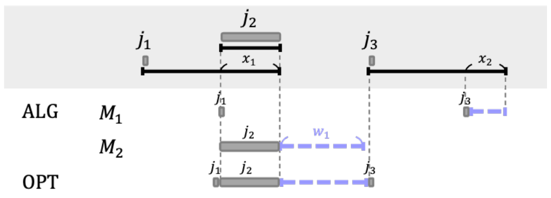

Figure 8 Case A with j_1, j_2 and j_3.

Next, the adversary releases another tiny request j_3 after the idle period of M_2. See Fig. 8. At this moment, both M_1 and M_2 are OFF, so we can use M_1 without loss of generality for j_3. Similarly as before, ALG executes j_3 at $d_3 - x_2$ and has an idle time of x_2. OPT executes j_3 immediately when it comes by keeping its idle state for w_1 assuming that $w_1 \leq 1$, which can be verified after we eventually fix all parameter values (indeed, $w_1 \leq 0.63197$ for our final setting of the parameters). Note that ALG needs three turn-ons and OPT one, so the current CR is $\frac{3+x_1+w_1+x_2}{1+x_1+w_1}$. For the game to be continued, it must be

$$x_2 \leq \alpha(1+x_1+w_1) - (3+x_1+w_1) \leq \alpha - 3 + (\alpha-1)\beta + (\alpha-1)w_1 \leq f_2 + (\alpha-1)w_1$$

by letting $\alpha - 3 + (\alpha-1)\beta = f_2$.

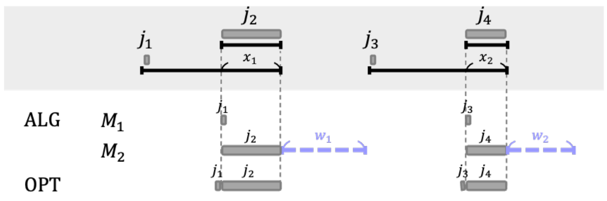

Figure 9 Case A with j_1 to j_4.

The adversary then releases j_4 exactly as it did for j_2 as shown in Fig. 9. Here, it is better for OPT to turn off at d_2 and turn on at a_4 since Adv selects a large $d_3 - a_3$. The CR is $\frac{4+x_1+w_1+x_2+w_2}{2+x_1+x_2}$. For the game to continue, we must have

$$\begin{aligned}
w_2 &\leq \alpha(2+x_1+x_2) - (4+x_1+w_1+x_2) \\
&= 2\alpha - 4 + (\alpha-1)x_1 + (\alpha-1)x_2 - w_1 \\
&\leq 2\alpha - 4 + (\alpha-1)\beta + (\alpha-1)(f_2 + (\alpha-1)w_1) - w_1 \\
&= 2\alpha - 4 + (\alpha-1)\beta + (\alpha-1)f_2 + (\alpha-1)^2 w_1 - w_1 \\
&= 2\alpha - 4 + (\alpha-1)(\beta + f_2) + (\alpha^2 - 2\alpha)w_1 \\
&= f_3 + (\alpha^2 - 2\alpha)w_1
\end{aligned}$$

by letting $f_3 = 2\alpha - 4 + (\alpha-1)(\beta + f_2)$.

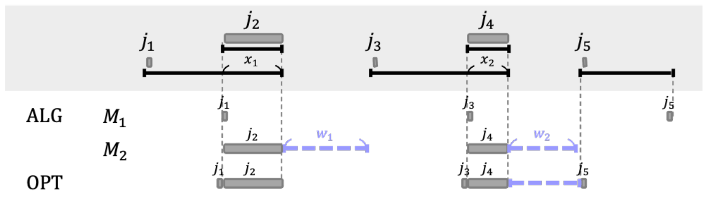

Figure 10 Case A ending with j_5.

Finally, the adversary releases a tiny request j_5 as shown in Fig. 10. For ALG, we impose only a turn-on cost which ALG at least spends. OPT can manage this request by continuing its idle state as before. Now the CR is at least

$$\frac{5+x_1+w_1+x_2+w_2}{2+x_1+x_2+w_2} \geq \frac{5+\beta+w_1+f_2+(\alpha-1)w_1+f_3+(\alpha^2-2\alpha)w_1}{2+\beta+f_2+(\alpha-1)w_1+f_3+(\alpha^2-2\alpha)w_1}$$
$$= \frac{5+\beta+f_2+f_3+(\alpha^2-\alpha)w_1}{2+\beta+f_2+f_3+(\alpha^2-\alpha-1)w_1}$$
$$\geq \frac{5+\beta+f_2+f_3+(\alpha^2-\alpha)f_1}{2+\beta+f_2+f_3+(\alpha^2-\alpha-1)f_1} = C_A.$$

The first inequality holds for the following reason: x_1, x_2 and w_2 appear both in the numerator and the denominator, the fraction becomes minimum when all x_1, x_2 and w_2 are maximum. For the second inequality, recall that our target CR, α, is greater than 2. So $\frac{(\alpha^2-\alpha)}{\alpha^2-\alpha-1} \leq 2$, which means the fraction becomes minimum when w_1 is maximum. Thus the CR is at least C_A for Case A.

Case B is similar and we can prove that the CR is at least

$$C_B = \frac{(\alpha^2-\alpha)g_1+4+\beta+g_2+g_3}{(\alpha^2-\alpha-1)g_1+2+g_2+g_3},$$

where $g_1 = \alpha-1-\beta$, $g_2 = \alpha-2-\beta$, and $g_3 = (\alpha-1)g_2+2\alpha-3-\beta$ (see [17] for more details).

For $\beta = 0.4745$ and $\alpha = 2.1068$, our numerical calculation shows that $C_A = 2.107447$ and $C_B = 2.106989$, and the theorem is proved. ◀

5 Concluding Remarks

Obvious future work is to narrow the gap. For the lower bound, one can easily notice that increasing the number of stages (currently, it is three) might help. This is correct and in fact one more stage can give us a strictly better bound. Unfortunately the degree of improvement is already pretty small and getting even smaller in further stages. For the upper bound, it is obviously important to consider more flexible structures for delaying requests. Our present algorithm executes all the pending requests when one request comes to its due time. It would be even more important to make our CR a function in ψ_σ. Our open question is that a smaller ψ_σ very likely implies a better CR.

References

1 Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms (TALG)*, 10(2):9, 2014.

2 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration. *Journal of Computer and System Sciences*, 81(7):1194–1209, 2015.

3 Antonios Antoniadis, Naveen Garg, Gunjan Kumar, and Nikhil Kumar. Parallel machine scheduling to minimize energy consumption. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2758–2769. SIAM, 2020.

4 Antonios Antoniadis, Chien-Chung Huang, and Sebastian Ott. A fully polynomial-time approximation scheme for speed scaling with sleep state. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1102–1113. SIAM, 2015.

5 Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):3, 2007.

6 Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE transactions on very large scale integration (VLSI) systems*, 8(3):299–316, 2000.

7 Jian-Jia Chen, Mong-Jen Kao, DT Lee, Ignaz Rutter, and Dorothea Wagner. Online dynamic power management with hard real-time guarantees. *Theoretical Computer Science*, 595:46–64, 2015.

8 Jian-Jia Chen and Tei-Wei Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. *ACM SIGPLAN Notices*, 41(7):153–162, 2006.

9 Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 289–294. IEEE, 2007.

10 Houssine Chetto and Maryline Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Information Processing Letters*, 30(4):177–184, 1989.

11 Erik D Demaine, Mohammad Ghodsi, MohammadTaghi Hajiaghayi, Amin S Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. *Journal of Scheduling*, 16(2):151–160, 2013.

12 Sandy Irani and Anna R. Karlin. *Online Computation*, pages 521–564. PWS Publishing Co., USA, 1996.

13 Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM Sigact News*, 36(2):63–76, 2005.

14 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Transactions on Embedded Computing Systems (TECS)*, 2(3):325–346, 2003.

15 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41, 2007.

16 Anna R Karlin, Mark S Manasse, Larry Rudolph, and Daniel D Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

17 Ya-Chun Liang, Kazuo Iwama, and Chung-Shou Liao. Improving the bounds of the online dynamic power management problem, 2022. `arXiv:2209.12021`.

18 Adam Wierman, Lachlan Leicester Henry Andrew, and Minghong Lin. *Speed scaling: An algorithmic perspective*, pages 385–406. CRC Press, 2012.

19 Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 374–382. IEEE, 1995.