Journal of Scheduling (2018) 21:633–646
https://doi.org/10.1007/s10951-018-0565-1

Speed scaling problems with memory/cache consideration

Weiwei Wu[1] · Minming Li[2] · Kai Wang[2] · He Huang[3] · Enhong Chen[4]

Published online: 6 June 2018

Abstract
Speed scaling problems consider energy-efficient job scheduling in processors by adjusting the speed to reduce energy consumption, where power consumption is a convex function of speed (usually, $P(s) = s^{\alpha}$, $\alpha = 2, 3$). In this work, we study speed scaling problems considering memory/cache. Each job needs some time for memory operation when it is fetched from memory,, and needs less time if fetched from the cache. The objective is to minimize energy consumption while satisfying the time constraints of the jobs. Two models are investigated, the non-cache model and the with-cache model. The non-cache model is a variant of the ideal model, where each job i needs a fixed c_i time for its memory operation; the with-cache model further considers the cache, a memory device with much faster access time but limited space. The uniform with-cache model is a special case of the with-cache model in which all c_i values are the same. We provide an $O(n^3)$ time algorithm and an improved $O(n^2 \log n)$ time algorithm to compute the optimal solution in the non-cache model. For the with-cache model, we prove that it is NP-complete to compute the optimal solution. For the uniform with-cache model with agreeable jobs (later-released jobs do not have earlier deadlines), we derive an $O(n^4)$ time algorithm to compute the optimal schedule, while for the general case we propose a $(2\alpha\frac{g}{g-1})^{\alpha}/2$-approximation algorithm in a resource augmentation setting in which the memory operation time can accelerate by at most g times.

Keywords Speed scaling · Energy efficiency · Scheduling · Memory operation time · DVS · Algorithm design

1 Introduction

Advances in processor, memory and communication technologies in recent years have given rise to a tremendous proliferation of portable electronic devices. Since such devices are often powered by batteries, the energy-efficient execution of jobs in order to prolong battery life is increasingly important. Processors capable of operating at a range of voltages and frequencies are already available (e.g., Intel's SpeedStep technology and AMD's PowerNow technology). The ability of a processor to adjust voltages is often referred to in the literature as dynamic voltage scaling (DVS). Since energy consumption is at least a quadratic function of the supply voltage (hence CPU speed), it saves energy to run the processor at the lowest possible constant speed, while still meeting all the timing constraints, rather than running at full speed and then switching to idle.

One of the earliest theoretical models for speed scaling problems based on DVS was introduced by Yao et al. (1995). The power consumption function $P(s)$ in the processor is convex and is usually assumed to be $P(s) = s^{\alpha}$, where s is the speed and α is a constant larger than 1. Each job has a release time r_i, deadline d_i and workload w_i. The schedule should decide which job to execute and at what speed at time t. Energy consumption is the integral of the power function over all time t. The goal is to minimize energy usage while

✉ Weiwei Wu
wweiwei2@gmail.com

Minming Li
minmli@cs.cityu.edu.hk

Kai Wang
kai.wang@my.cityu.edu.hk

He Huang
huangh@suda.edu.cn

Enhong Chen
cheneh@ustc.edu.cn

[1] School of Computer Science and Engineering, Southeast University, Nanjing, China

[2] Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong

[3] School of Computer Science and Technology, Soochow University, Suzhou, China

[4] School of Computer Science, University of Science and Technology of China, Hefei, China

satisfying all the timing constraints of n jobs. This model is referred to as the *ideal model* by Yao et al. 1995. The authors proposed a polynomial time algorithm to compute the minimum-energy schedule (MES). Several online heuristics were also considered, including the average rate heuristic (AVR) and optimal available heuristic (OPA). The constant competitiveness of AVR and OPA are derived in Yao et al. (1995) and Bansal et al. (2004), respectively, which verifies the effectiveness of the online algorithms.

Additional algorithms for energy-efficient scheduling were subsequently proposed under various related models and assumptions. In the case of a CPU that is only able to change speed gradually instead of instantaneously, Hong et al. (1998) discussed some special cases that could be solved optimally in polynomial time. Wu et al. (2009) studied an acceleration model where the rate of speed change is bounded by a constant K. The authors derived an efficient algorithm to compute the optimal solution for agreeable jobs, i.e., earlier-released jobs have earlier deadlines. Aside from these DVS models, there are also other extensions. For example, Irani et al. (2007), Albers and Antoniadis (2014) and Antoniadis et al. (2015) investigated approximation algorithms for an extended scenario where the processor can be put into a low-power sleep state when idle, and a certain amount of energy is needed when the processor changes from the sleep state to the active state. Albers et al. (2015) developed an optimal offline algorithm and online heuristic algorithms for multi-processor speed scaling problem with migration. Other related works can be found in survey papers (Albers 2010; Bambagini et al. 2016).

Although numerous studies had been performed using different assumptions regarding CPU variable speed capacity, it was not until 2003 that a more realistic model incorporating the effect of memory operations was proposed by Seth et al. (2003). The authors noted that memory access time is dependent on the front-side bus (FSB), which is a fixed value, making memory operations difficult to accelerate. Since the execution time of memory operations is not affected by the processor frequency, the workload of each job is now divided into two parts: a computational component whose execution timescales are inversely proportional to the CPU speed, and a memory component whose execution time is fixed. In addition, if cache-locking techniques (which allow placing jobs in the cache) are supported, then we need to further consider which jobs to put in the cache to reduce the total energy consumption, because jobs placed in the cache require less memory execution time. Other, more recent studies have investigated the use of this model for periodic jobs that have periodic releases and processing requirements. For example, Bini et al. (2005) extended the model by allowing periodic jobs to have deadlines earlier than the end of the period and enforcing a discrete number of processor speeds. Aydin et al. (2006) assumed different power dissipation and on-chip/off-chip workload characteristics for different tasks. Yang et al. (2007) introduced a preemption control technique which significantly reduces the number of preemptions while also minimizing energy consumption. For additional works involving this model, please refer to Choi et al. (2005) and Hsu and Feng (2004).

In this paper, we theoretically study two models which consider memory operation and non-periodic jobs. The *non-cache model* is a generalization of the ideal model, where each job i requires a fixed c_i time to finish the memory operation, in addition to its computational workload w_i. The *with-cache model* additionally allows the jobs to be placed into the cache (which has much faster access speed than normal memory) to further save time in memory operations and prolong the execution time of the jobs. The number of jobs to be stored in the cache (limited space) is bounded by a constant N. We say a schedule has k evictions if it allocates k jobs to memory. The algorithm should determine the allocation and speed of jobs to achieve minimum energy consumption. The *uniform with-cache model* is a special case where every job has the same c_i value.

Our contributions are summarized as follows. For the non-cache model, we provide an $O(n^3)$ time algorithm and an improved $O(n^2 \log n)$ time algorithm to compute the optimal solution. For the with-cache model, we prove that optimizing energy for general jobs is NP-complete. We then study the agreeable jobs in the uniform with-cache model. We derive an $O(n^4)$ time dynamic programming algorithm to compute the optimal solution. Note that these results rely only on the assumption that $P(s)$ is convex and nonnegative, and $P(0) = 0$. We then consider the general jobs in the with-cache model with the stricter but common assumption that $P(s) = s^\alpha$ ($\alpha \geq 1$). We study the resource augmentation setting of this problem where the job's eviction time (memory operation time) can accelerate s times, which will be referred to as the *g-speed with-cache model*. In the resource augmentation setting, an algorithm is a c-approximation if it always outputs a solution (given the g-speed augmented resource) that is at most c times that of the optimal solution for the 1-speed with-cache model. We propose a $(2\alpha \frac{g}{g-1})^\alpha/2$-approximation for the g-speed with-cache model.

The remainder of this paper is organized as follows. In Sect. 2, we review models of speed scaling problems. In Sect. 3, we develop optimal algorithms for the non-cache model under continuous speed and discrete speed settings. In Sect. 4, we turn to the with-cache model, and present the NP-completeness of the optimization problem for the model. Section 4.1 derives a dynamic programming algorithm for the uniform with-cache model, and Sect. 4.2 investigates the g-speed resource augmentation setting of the with-cache model. Concluding remarks are presented in Sect. 5.

2 Formulation

The input $\mathcal{J}$ is composed of n jobs $J_i = (r_i, d_i, w_i, c_i)$, where $1 \leq i \leq n$. Each job i (or J_i) has a release time r_i, deadline d_i, workload w_i and memory operation time c_i.

In the *non-cache model*, a schedule S should specify three functions: the speed $s(t)$ for time t, the function $\delta(t, i)$ that indicates whether time t is used for the computation operation for job i, and the function $\rho(t, i)$ that indicates whether time t is used for the memory operation for job i. When time t is used for the computation operation (memory operation) for job i, we say that job i is executed (or evicted) at time t. Hence, $\delta(t, i)$ (or $\rho(t, i)$) equals 1 if job i is executed (or evicted) at time t. At most one job is executed or evicted at any time t, i.e., $\sum_{i\in\mathcal{J}}(\delta(t, i) + \rho(t, i)) \leq 1$. The condition $\sum_{i\in\mathcal{J}} \rho(t, i) = 1$ implies that $s(t) = 0$, because no job is executed at time t. The goal is to find a *feasible* schedule to minimize energy consumption $E = \int_0^\infty s^\alpha(t)\mathrm{d}t$. A schedule is feasible if it satisfies both the *workload constraint* $\int_{r_i}^{d_i} s(t)\delta(t, i)\mathrm{d}t \geq w_i$ and the *eviction time constraint* $\int_{r_i}^{d_i} \rho(t, i)\mathrm{d}t \geq c_i$. We assume a *preemptive* setting where the unfinished workload of a job that is suspended can be resumed later, without any penalty. For the memory operation when *eviction preemption* is allowed, the job's eviction time can be allocated to several separate intervals (jobs would later resume at that break). When *eviction preemption* is not allowed, the job's eviction time should be allocated to a contiguous interval. In this work, we focus on the case in which eviction preemption is allowed.

In the *with-cache model*, with the support of cache-locking techniques, we need to decide which jobs to put in the cache to reduce the energy consumed. Job i has eviction time c_i if it is allocated to memory, and eviction time 0 if it is allocated to the cache. $b_i = 1$ indicates that job i is allocated to the cache; otherwise $b_i = 0$. The *eviction time constraint* is $\int_{r_i}^{d_i} \rho(t, i)\mathrm{d}t \geq 0$ if $b_i = 1$ and $\int_{r_i}^{d_i} \rho(t, i)\mathrm{d}t \geq c_i$ if $b_i = 0$. We assume that each job occupies one slot of the memory/cache. The cache is usually much smaller than the memory. Assuming that the cache has N slots, while the memory has unbounded size, to reduce energy consumption, a schedule will allocate as many jobs as possible to the cache (which reduces the memory operation time of the jobs), i.e., $\sum_{1\leq i\leq n} b_i = N$. We say a schedule has k *evictions* if k jobs are stored in memory. Set $K = n - N$. Clearly, the optimal solution has K evictions. The *uniform with-cache model* is a special case of the with-cache model where every job has the same length of eviction time c if it is allocated to memory (i.e., $J_i = (r_i, d_i, w_i, c)$).

In the resource augmentation setting of the with-cache model, the job's eviction time can accelerate s times, which is referred to as the *g-speed with-cache model*. Given the g-speed augmented resource, an algorithm is a c-approximation if it always outputs a solution that is at most c times the optimal solution for the 1-speed with-cache model.

By abusing the notation, we use OPT to denote both the optimal schedule and the energy in the optimal schedule for a specified model (if the context is clear). We define OPT $= \infty$ when the input instance has no feasible schedule. We use set $\mathcal{R}$ (or $\mathcal{D}$) to denote the collection of different release times (or deadlines) for the jobs. We define $r_{\min} = \min_{i\in\mathcal{J}} r_i$ and $d_{\max} = \max_{i\in\mathcal{J}} d_i$. By $W_{[t_1,t_2]}(S)$, we denote the total workload that is executed in interval $[t_1, t_2]$ in schedule S. Given a schedule, the maximum interval that has the same speed is called a *block*. The *peak block* is the block with the maximum speed among all blocks of a schedule. The time t is said to be *tight in S* if it is used to execute some job i which has a deadline/release time at exactly t in S (we say t is a tight deadline/tight release time, respectively, in S).

3 Non-cache model

In this section, we develop the optimal scheduling algorithms for the non-cache model. First, we derive an $O(n^3)$ time algorithm to compute the optimal solution in the continuous speed setting. Next, we develop an $O(dn \log n)$ time algorithm to solve this problem under the discrete speed setting, with d discrete speeds available. Finally, we extend the approach in the discrete speed setting to obtain an algorithm for the continuous speed setting, improving the time complexity to $O(n^2 \log n)$.

3.1 Optimal continuous voltage schedule

We first propose an $O(n^3)$ time algorithm in the continuous speed setting, based on the idea of fixing the schedule block by block. The following fact states a fundamental property that will be used frequently.

Fact 1 *Assume that there exists a schedule to execute jobs j_1, j_2 in two separate intervals with length t_1, t_2 and speed s_1, s_2, respectively, where $s_1 > s_2$. Then, another feasible schedule which moves a partial workload of j_1 to be executed in j_2's interval with new speeds $s'_1 \geq s'_2$, where $s'_1 < s_1$ and $s'_2 > s_2$, reduces the total energy consumed by these two jobs.*

This can be easily proved using the convexity of the power function.

To simplify our discussion, we define the *virtual speed* of job j as the speed of its eviction interval. If S executes job j with speed s_j (one speed by the provable property (P1) in Lemma 1 below), we then set the virtual speed of j in its eviction interval to be s_j (which actually does not execute any workload). Thus, in the following, when we refer to block $[a, b]$ in S, we mean the maximum interval which has the same speed (including the virtual speed) in S. Consider the case that schedule S arranges the eviction time of job j in interval $[t_j, d_j]$. For unification, w.l.o.g., we assume that OPT executes *virtual workload* $s_j(d_j - t_j)$ of j in interval $[t_j, d_j]$. Thus, in this case we can also say that d_j is a tight deadline in S. We first prepare some key properties of the optimal solution, which are extended from the ideal model (Yao et al. 1995), with the proof moved to the "Appendix".

Lemma 1 *The optimal solution of the non-cache model has three properties:*

(P1) *There is an optimal schedule, with jobs executed in EDF order.*
(P2) *OPT always executes each job with a single speed. OPT is composed of blocks.*
(P3) *For a peak block $B = [a, b]$ in OPT, a is a tight arrival time and b is a tight deadline.*

Based on these three proven properties, we derive a key observation for the peak block of the optimal solution.

Lemma 2 *The speed in the peak block $B = [a, b]$ of OPT is* $\frac{\sum_{i:[r_i,d_i]\subseteq[a,b]} w_i}{b-a-\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i}$.

Proof Assume that OPT has speed s_1 in $[a, b]$, which is the largest among all blocks. We first argue that OPT only executes jobs with $[r_i, d_i] \subseteq [a, b]$ in interval $[a, b]$. Otherwise, there exists a job j with $d_j > b$ or $r_j < a$ that is executed in $[a, b]$ (assume that such a job is executed in $[a_0, b_0]$ with $a \le a_0 < b_0 \le b$). W.l.o.g., assume that $d_j > b$. If OPT executes some workload in interval $[a', b']$ with $b \le a' < b' \le d_j$ and speed $s_2 < s_1$, then moving a partial workload of j from interval $[a_0, a_0 + \epsilon]$ to $[a', a' + \epsilon]$ (resulting in the same speed in these two intervals) will reduce the energy by Fact 1. For the remaining case in which jobs executed in interval $[b, d_{j_1}]$ have speed s_1 (or the whole interval $[b, d_{j_1}]$ is used for eviction), we can apply the swapping procedure as defined in Lemma 1 to deduce a contradiction. Hence, OPT executes only jobs with $[r_i, d_i] \subseteq [a, b]$ in the interval $[a, b]$ in EDF order by Lemma 1. This block has one speed and total eviction time of at least $\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i$. In OPT, all time of $[a, b]$ excluding the eviction time should run the workload with speed of at least $\frac{\sum_{i:[r_i,d_i]\subseteq[a,b]} w_i}{b-a-\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i}$ to finish the jobs. We then remove the possibility of $s_1 > \frac{\sum_{i:[r_i,d_i]\subseteq[a,b]} w_i}{b-a-\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i}$. We will first show that OPT will not allocate the eviction time of a job j_1 with $d_{j_1} > b$ or $r_{j_1} < a$ in interval $[a, b]$. W.l.o.g., we only discuss the case $d_{j_1} > b$, since it holds symmetrically for the case $r_{j_1} < a$. We choose a small interval $[t, t + \epsilon]$ with $a \le t < t+\epsilon \le b$ in which OPT allocates j_1's eviction time. Similar to the proof of (P3) in Lemma 1, we assume that j_1's workload is executed with speed s_1 in interval $[a_0, a_0 + \epsilon']$ (the only difference now is that this small interval must be outside interval $[a, b]$). By discussing the same two cases of interval $[b, d_{j_1}]$ (whether or not there exists a job executed in $[b, d_{j_1}]$ with speed $s_2 < s_1$) as the proof of (P3) in Lemma 1, we can again deduce a contradiction. Thus, no job with $d_{j_1} > b$ or $r_{j_1} < a$ has its eviction time allocated in $[a, b]$ in OPT. By the assumption $s_1 > \frac{\sum_{i:[r_i,d_i]\subseteq[a,b]} w_i}{b-a-\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i}$, there exists at least one small interval in $[a, b]$ that is not used for either the execution of the workload or for eviction time. This contradicts the choice of "block" $[a, b]$. Therefore, OPT always has speed $\frac{\sum_{i:[r_i,d_i]\subseteq[a,b]} w_i}{b-a-\sum_{i:[r_i,d_i]\subseteq[a,b]} c_i}$ if $[a, b]$ is its peak block. □

Using this lemma, we derive Theorem 1 to compute the optimal solution. The general idea is as follows: identify the peak block that corresponds to the largest speed block in the optimal solution, and then re-scale the jobs to a new instance; iteratively find the new peak block in the new instance that can be verified to be the second largest speed block in the optimal solution.

Theorem 1 *The optimal solution for the non-cache model can be computed in $O(n^3)$ time.*

Proof By Lemma 1, the peak block of OPT can be computed by looking for an interval $[t_1, t_2]$ where $t_1, t_2 \in \mathcal{R} \cup \mathcal{D}$ that maximizes $\frac{\sum_{i:[r_i,d_i]\subseteq[t_1,t_2]} w_i}{t_2-t_1-\sum_{i:[r_i,d_i]\subseteq[t_2,t_1]} c_i}$. We only need to try all possible pairs of tight times in $\mathcal{R} \cup \mathcal{D}$ that could form such an interval. Note that the peak block $[a_1, b_1]$ executes jobs in (and only in) $\{i : [r_i, d_i] \subseteq [a_1, b_1]\}$. This allows us to handle the remaining jobs recursively to determine the block with the second highest speed in OPT. The method consists in computing the peak block $[a_2, b_2]$ for the remaining jobs $\mathcal{J} \setminus \{i : [r_i, d_i] \subseteq [a_1, b_1]\}$ as though there were no interval $[a_1, b_1]$ (Bansal et al. 2004). Applying the same proof in Lemma 1, we can exactly determine the jobs that should be executed in interval $[a_2, b_2]$. Therefore, we can recursively identify all the blocks of OPT and also its optimal schedule. There are at most n blocks, and computing each block uses $O(n^2)$ time, since enumerating all possible pairs of tight times requires $O(n^2)$ time. Thus, energy optimization of the non-cache model can be completed in $O(n^3)$ time. □

3.2 Optimal discrete voltage schedule

In this section, we consider the discrete speed setting in which the processor can run at only d given speed levels $s_1 > s_2 > \cdots > s_d > 0$. In the non-cache model, the execution time of a job consists of two parts: job processing time and memory operation time. In the analysis, we may consider the allocation time for one job. Once the union of time intervals is assigned to the job, it does not really matter which operation comes first (memory operation or job processing) as long as the timing constraints for both job processing and memory operation are satisfied.

It is natural to compute the optimal continuous voltage schedule and then adjust the speed to its neighboring speed among $\{s_1, s_2, \ldots, s_d, s_{d+1}\}$, where we let $s_{d+1} = 0$. For a single job, it has been proven in Ishihara and Yasuura (1998) that the minimum energy is achieved by using the immediate neighbors $s_l,\ s_{l+1}$ of the ideal speed s in appropriate proportions. It also works for this model with memory consideration, as rounding the speed to further neighbors will result in more energy consumption owing to Fact 1. This property is necessary, as in the following we show how to obtain the schedule of discrete speeds without computing the optimal continuous voltage schedule. For the ideal model without memory operation time, a method was introduced in Li and Yao (2005) to obtain an optimal schedule, which contains two main steps, partitioning and two-level scheduling. We show that this method can be extended to our model and gives an optimal algorithm with time complexity $O(dn \log n)$.

For ease of presentation, we define three independent states of a processor: *idle* (no job to be executed), *mo* (performing memory operation for a job) and *processing* (performing calculation for a job). Only when a processor is in state *processing* does it consume energy.

3.2.1 Partition

The analysis in this section is based on the optimal continuous schedule of $\mathcal{J}$. From Lemma 1, the optimal continuous voltage schedule follows the EDF policy and has the following properties: the processing speed of one job is constant, and the processing speed (including virtual speed) in a block is also constant. In this optimal continuous schedule of $\mathcal{J}$, let $\mathcal{J}^{\geq s}$ (resp. $\mathcal{J}^{<s}$) denote the subset of $\mathcal{J}$ consisting of jobs whose processing speeds $\geq s$ (resp. $< s$), and let $T^{\geq s}$ (resp. $T^{<s}$) denote the unions of blocks in which the processing speed $\geq s$ (resp. $< s$).

In this section, we aim to compute $\mathcal{J}^{<s}$ and $T^{<s}$ with a constant s. Using the approach in Li and Yao (2005), we show in the following that the partition $\left\langle \mathcal{J}^{\geq s}, \mathcal{J}^{<s} \right\rangle$ can be computed without computing the optimal continuous schedule. An s-**schedule** is a schedule that processes jobs in EDF order at constant speed s and operates memory for one job immediately once the job is determined to be executed. In the optimal continuous schedule, for each job i during time interval $[r_i, d_i]$, the machine cannot be idle and the machine speed (including virtual speed) cannot be smaller than the processing speed of job i. Hence, any job that can be scheduled during $T^{<s}$ must belong to $\mathcal{J}^{<s}$. In other words, for each job $i \in \mathcal{J}^{\geq s}$, the interval $[r_i, d_i]$ is outside $T^{<s}$. As jobs $\mathcal{J}^{\geq s}$ can never be executed during $T^{<s}$, the union of time intervals $T^{<s}$ is more than enough to finish jobs $\mathcal{J}^{<s}$ under constant speed s. As a result, by running s-schedule, an idle interval (we call it "gap") must exist and must only exist in $T^{<s}$, and all jobs of $\mathcal{J}^{<s}$ must be finished during $T^{<s}$. Moreover, some jobs of $\mathcal{J}^{\geq s}$ must be unfinished (or finished exactly at its deadline), and we call this deadline a critical deadline in the s-schedule. By running the s-schedule in a reverse manner, we can get the critical release times as well. As a matter of fact, the critical deadline and release times come from jobs $\mathcal{J}^{\geq s}$ and hence cannot be inside $T^{<s}$. Also, the execution intervals of unfinished jobs (and jobs that are finished exactly at their deadline) in s-**schedule** will cover all time intervals outside $T^{<s}$. Therefore, the boundary of each interval in $T^{<s}$ must be a critical deadline or critical release time. By expanding each gap to the right (resp. left) until reaching any critical release time (resp. critical deadline), we are able to identify $T^{<s}$ and furthermore to obtain $\mathcal{J}^{<s}$, and hence $\mathcal{J}^{\geq s}$, $T^{\geq s}$.

When the machine speed is fixed to constant s, for each job the required processing time is also fixed, i.e., there is no difference between processing time and memory operation time with respect to the job. Therefore, the following theorem is cited from Li and Yao (2005).

Theorem 2 *For a given speed s, the partition $\left\langle \mathcal{J}^{\geq s}, \mathcal{J}^{<s} \right\rangle$ of job set $\mathcal{J}$ can be computed in $O(n\ log\ n)$ time.*

We keep partitioning $\mathcal{J}$ by the given speeds. For the input of d given speeds, job set $\mathcal{J}$ could be partitioned into d disjoint groups $\mathcal{J}^1, \mathcal{J}^2, \ldots, \mathcal{J}^d$, where $\mathcal{J}^l$ consists of all jobs whose execution speeds in the continuous optimal schedule lie between s_l and s_{l+1}.

This process can be completed by applying the second fastest speed s_2 first, and obtains $T^{\geq s_2}$ and $\mathcal{J}^{\geq s_2}$ (which is $\mathcal{J}^1$), according the mentioned partition method. Then, by eliminating the occupied time intervals $T^{\geq s_2}$ for the remaining jobs, we apply the speed s_3, and so on. Based on Theorem 2, the partition $\{\mathcal{J}^1, \mathcal{J}^2, \ldots, \mathcal{J}^d\}$ takes $O(dn\ log\ n)$ time.

3.2.2 Two-level schedule

As argued earlier, there exists an optimal schedule that executes jobs in group $\mathcal{J}^l$ at two speeds s_l and s_{l+1}.

Definition 1 For a job set $\mathcal{J}$, a two-level schedule (or (s_l, s_{l+1})-schedule for short) with speeds s_l and s_{l+1}, $s_l > s_{l+1}$, is a feasible schedule $s(t)$ for $\mathcal{J}$ if for any time t, $s(t) \in \{s_l, s_{l+1}\}$ or some job is evicted at time t (memory operation).

Theorem 3 *For each job group $\mathcal{J}^l$, an (s_l, s_{l+1})-schedule exists if and only if*

(1) *the s_l-schedule for $\mathcal{J}^l$ is a feasible schedule, and*

(2) *the s_{l+1}-schedule for $\mathcal{J}^l$ contains no idle processor state.*

Proof Note that $s_l > s_{l+1}$. If (s_l, s_{l+1})-schedule exists, then both (1) and (2) hold by definition. Condition (1) shows that speed s_l is sufficient to finish all jobs, while speed s_{l+1} is not. Therefore, by slowing the speed in s_{l+1}-schedule to eliminate all the idle periods, one can obtain an (s_l, s_{l+1})-schedule. □

Theorem 4 *Any (s_l, s_{l+1})-schedule for job group $\mathcal{J}^l$ is optimal.*

Proof We prove the theorem by showing that any feasible schedule of $\mathcal{J}^l$ using two speeds s_l and s_{l+1}, without idle time, consumes the same amount of energy, which results in the fact that any such feasible (s_l, s_{l+1})-schedule is optimal.

Given a feasible schedule of $\mathcal{J}^l$, let α and β denote the total amount of time the processor runs at speed s_l and s_{l+1}, respectively. Let t_{all} denote the total available execution time for all jobs of $\mathcal{J}^l$. We then have

$$\begin{cases} \alpha s_l + \beta s_{l+1} = \sum_{j\in\mathcal{J}^l} w_j, \\ \alpha + \beta + \sum_{j\in\mathcal{J}^l} c_j = t_{\text{all}}. \end{cases}$$

t_{all} and total workload of job set $\mathcal{J}^l$ are fixed values; thus α and β are the same in all (s_l, s_{l+1})-schedule of job set $\mathcal{J}^l$, and the energy consumption is the same. There is no idle state of the processor in any (s_l, s_{l+1})-schedule, and t_{all} contains all applicable time intervals. Thus, any (s_l, s_{l+1})-schedule is optimal. □

We give an algorithm extended from Li and Yao (2005) in the "Appendix" (Algorithm 5) to find the (s_l, s_{l+1})-schedule of job group $\mathcal{J}^l$ in the "Appendix". Finally, we obtain the optimal schedule of $\mathcal{J}$ by first computing the (s_1, s_2)-schedule for jobs in $\mathcal{J}^1$, eliminating the time intervals occupied by jobs in $\mathcal{J}^l$ for the remaining jobs, and we then compute the (s_2, s_3)-schedule for jobs in $\mathcal{J}^2$, and so on.

3.3 Improved optimal continuous voltage schedule

We now use the partition method in the above-referenced discrete speed setting to improve the time complexity of computing the optimal solution in the continuous speed setting. We will show that the continuous optimal voltage schedule can be obtained in $O(n^2 \log n)$ time.

Define T as the union of all job intervals of job set $\mathcal{J}$. Define the average rate $avr(\mathcal{J}) = \frac{\sum_{j\in\mathcal{J}} w_j}{|T| - \sum_{j\in\mathcal{J}} c_j}$, where $|T|$ is the length of available intervals in T. By executing s-schedule of job set $\mathcal{J}$ over T with speed $s = avr(\mathcal{J})$, we obtain a partition $\langle J^{\geq s}, J^{<s}\rangle$.

Lemma 3 *Let $s = avr(\mathcal{J})$, if $T^{<s} = \emptyset$; then s-schedule is an optimal schedule for job set $\mathcal{J}$.*

Proof Suppose there is an optimal schedule with speed function S_{opt}. It is easy to see that $S_{\text{opt}} \geq s$ in interval $T^{\geq s}$, and $S_{\text{opt}} < s$ in interval $T^{<s}$. Since $\int_T S_{\text{opt}} \mathrm{d}t = \int_T s\, \mathrm{d}t = \sum w_k$, we have $\int_{T^{\geq s}} (S_{\text{opt}} - s)\mathrm{d}t = \int_{T^{<s}} (s - S_{\text{opt}})\mathrm{d}t$. If $T^{<s} = \emptyset$, then $S_{\text{opt}} = s$ over $T^{\geq s}$ and $T^{\geq s} = T$. □

Based on Lemma 3, the optimal schedule can be constructed by executing partition $\langle \mathcal{J}^{\geq s}, \mathcal{J}^{<s}\rangle$ of job set $\mathcal{J}$, recursively, where we describe the process in Algorithm 1.

Algorithm 1 Partitioned Optimal Voltage Schedule (POVS)

```
Input: job set $\mathcal{J}$, available interval $T$ of $\mathcal{J}$
Output: continuous optimal voltage schedule for job set $\mathcal{J}$

if $\mathcal{J} = \emptyset$ then
    return
end if
$s \leftarrow avr(\mathcal{J})$
$\langle \mathcal{J}^{\geq s}, \mathcal{J}^{<s}\rangle \leftarrow Partition(\mathcal{J}, s)$

if $T^{<s} = \emptyset$ then
    return $s$-schedule over $T$
else
    return the union of schedules $POVS(\mathcal{J}^{\geq s}, T^{\geq s})$ and $POVS(\mathcal{J}^{<s}, T^{<s})$
end if
```

In Algorithm 1, each partitioning for job set $\mathcal{J}$ will cost $O(n \log n)$ time and at most $O(n)$ recursion nodes. In total, Algorithm 1 computes a continuous optimal schedule for a job set $\mathcal{J}$ in $O(n^2 \log n)$ time.

4 With-cache model

In this section, we investigate the with-cache model, which further allows cache operation with faster memory access time. We will first prove that computing the optimal solution in the general setting is NP-complete. We will then study the uniform with-cache model with agreeable jobs and give a polynomial time algorithm to compute the optimal solution. Finally, we will propose an approximation algorithm for the general setting with resource augmentation.

We start by presenting the complexity of the general with-cache model in the following theorem.

Theorem 5 *Computing the optimal solution for the speed scaling problem in the with-cache model is NP-complete.*

Proof Given any instance of the Partition problem (with $|U|$ items $u_1, u_2, \ldots, u_{|U|}$ and $\sum_{j=1}^{|U|} u_j = 2B$ where each $u_j > 0$), the goal of the Partition problem is to find a subset with total value B. We construct a corresponding instance of the with-cache model with $2|U| + 1$ jobs. Let the number of evictions be $K \in [1, |U|]$, and the corresponding settings are as follows:

$$P(s) = s^2;\ u_{\max} = \max_{1\le i\le |U|} u_i;$$
$$y_i = \sqrt{\frac{11u_{\max}}{4u_i} - \frac{1}{2}}$$
$$w_0 = 3B;$$
$$T = \sum_{i=1}^{|U|} 4y_i^2 u_i - (n-K)\frac{11u_{\max}}{2} + 4B;$$

We construct the following $2|U|+1$ jobs.

$$J_0 = \left(0, 4\sum_{0\le j\le |U|} u_j, 3B, 0\right);$$
$$J_{2i-1} = \left(4\sum_{0\le j\le i-1} u_j, 4\sum_{0\le j\le i-1} u_j + 2u_i, 2y_i u_i, 0\right),$$
$$1 \le i \le |U|;$$
$$J_{2i} = \left(4\sum_{0\le j\le i-1} u_j + u_i, 4\sum_{0\le j\le i-1} u_j + 3u_i, 0, 2u_i\right),$$
$$1 \le i \le |U|;$$

In the jobs constructed, each job has a time interval of length $2u_i$. Note that every job J_{2i-1} has eviction time 0, while job J_{2i} has eviction time $2u_i$. Each job J_{2i} has zero workload; thus, it has eviction time $2u_i$ when its speed is fixed to be zero. The speed of job J_{2i-1} would be affected by cases of whether J_{2i} is allocated to the cache.

We will show that the constructed instance has energy $E \le T$ if and only if the Partition problem has a subset with sum B and the total number of elements in that set is exactly K.

We start by analyzing the possible schedules that optimize the energy. First we have $y_i = \sqrt{\frac{11u_{\max}}{4u_i} - \frac{1}{2}} \ge \frac{3}{2}$. Thus, job J_{2i-1} should be executed with speed of at least $\frac{3}{2}$ in OPT. Let $b_j = 0$ if OPT allocates job J_j to memory for eviction (incurs eviction time c_j for this job) and $b_j = 1$ if OPT allocates job j to the cache for eviction (incurs eviction time 0 for this job). Note that job J_{2i} needs eviction time of exactly $2u_i$. If $b_{2i} = 0$, then the eviction interval of J_{2i} is exactly $\left[4\sum_{0\le j\le i-1} u_j + u_i, 4\sum_{0\le j\le i-1} u_j + 3u_i\right]$. A simple extension of Lemma 1 can show that job J_0 is executed at a single speed in OPT. Note that the total workload w_0 is equal to $3B$. Even if these workloads are executed only in interval $\cup_{1\le i\le |U|}\left[4\sum_{0\le j\le i-1} u_j + 3u_i, 4\sum_{0\le j\le i-1} u_j + 4u_i\right]$ (that can only be used to execute J_0), the speed is at most $\frac{3}{2}$, which is smaller than the speed of job J_{2i-1}. Therefore, J_0 would not be executed in interval $[r_{2i-1}, d_{2i-1}]$ by Fact 1.

Based on the above-stated properties, J_{2i-1} runs until time $r_{2i}+u_i$ with a time interval of length $2u_i$ if J_{2i} is in the cache, and runs until time r_{2i} with a time interval of length u_i if J_{2i} is not in the cache. Moreover, J_0 is free to use the remaining intervals. Therefore, the energy consumed by J_{2i-1} is $2y_i^2 u_i$ when $b_{2i} = 1$ and is $(2y_i)^2 u_i$ when $b_{2i} = 0$. Job J_0 is not executed in interval $[r_{2i}+u_i, d_{2i}]$ when $b_{2i} = 0$, and is allowed to execute there when $b_{2i} = 1$. Thus, the total time that job J_0 is executed in OPT can be written as $\sum_{i=1}^{|U|} u_i + \sum_{i=1}^{|U|} b_{2i}u_i$. The energy incurred by J_0 is thus $\frac{w_0^2}{\sum_{i=1}^{|U|}(u_i + b_{2i}u_i)} = \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}$. The total energy is

$$E = \sum_{i=1}^{|U|} 4y_i^2 u_i + \sum_{i=1}^{|U|}\left(2y_i^2 u_i - 4y_i^2 u_i\right) b_{2i} + \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}$$
$$= \sum_{i=1}^{|U|} 4y_i^2 u_i + \sum_{i=1}^{|U|}\left(-2y_i^2 u_i\right) b_{2i} + \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}.$$

Since $y_i = \sqrt{\frac{11u_{\max}}{4u_i} - \frac{1}{2}}$, we have

$$E = \sum_{i=1}^{|U|} 4y_i^2 u_i - \sum_{i=1}^{|U|} b_{2i}\left(\frac{11u_{\max}}{2} - u_i\right) + \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}$$
$$= \sum_{i=1}^{|U|} 4y_i^2 u_i - \sum_{i=1}^{|U|} b_{2i}\frac{11u_{\max}}{2} + \sum_{i=1}^{|U|} b_{2i}u_i + \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}$$
$$= \sum_{i=1}^{|U|} 4y_i^2 u_i - (n-K)\frac{11u_{\max}}{2} + \sum_{i=1}^{|U|} b_{2i}u_i + \frac{w_0^2}{2B + \sum_{i=1}^{|U|} b_{2i}u_i}.$$

The last step is taken because there should be K evictions where $K = n - \sum_{i=1}^{|U|} b_{2i}$.

Let $x = \sum_{1\le i\le |U|} b_{2i}\cdot u_i$. Note that we set $T = \sum_{i=1}^{|U|} 4y_i^2 u_i - (n-K)\frac{11u_{\max}}{2} + 4B$. E is minimized at $\sum_{i=1}^{|U|} 4y_i^2 u_i - (n-K)\frac{11u_{\max}}{2} - 2B + 2w_0 = T$ if and only if $2B + x = w_0 \wedge \sum_{i=1}^{|U|} b_{2i} = K$. Since $w_0 = 3B$, E is minimized if and only if $\sum_{i=1}^{|U|} b_{2i}u_i = B \wedge \sum_{i=1}^{|U|} b_{2i} = K$. The Partition problem says that looking for any subset with sum B is NP-complete. Thus, the problem that looks for a subset with sum B where the total number of elements is K, is still NP-complete. Because otherwise we can search in $O(|U|)$ time to find a solution in polynomial time for the Partition problem.

Hence, the decision version of the with-cache model is NP-complete. This finishes the proof. □

4.1 Uniform with-cache model with agreeable jobs

In this section, we design an $O(n^4)$ time algorithm for the uniform with-cache model with agreeable jobs (where earlier-released jobs have earlier deadlines).

Before discussing the details, we present an example to demonstrate a difference in the properties of the optimal solution structure between the non-cache model and the with-cache model. The optimal algorithm in Sect. 3 implies that the optimal solution for the non-cache model with jobs executed in EDF order is unique (this can be verified by observing the uniqueness of the schedule in the peak block of the optimal solution). However, this property does not hold in the with-cache model. We construct an instance for illustration. The instance is composed of three jobs, $J_1 = (0, 2, 4, 1)$, $J_2 = (0, 7, 3, 1)$, $J_3 = (5, 7, 4, 1)$. Set $K = 1, \alpha = 2$. There are three cases (corresponding to selecting one job to store in the cache) among the feasible schedules. Minimum energy consumption can only be achieved by allocating job 1 or job 3 to the cache. Moreover, the peak block is interval [0, 2] with speed 4 when we choose job 3, while the peak block is interval [5, 7] with speed 4 when we choose job 1. Both of these schedules achieve the minimum cost of 32. Clearly, the optimal schedule (restricted to EDF order) for the with-cache model is not unique. Thus, the notion of a greedy-like algorithm for the non-cache model is difficult to extend to the with-cache model.

For ease of understanding, in the following discussion we will focus on the case where an arrival time would not be a deadline, i.e., any time t is either an arrival time or a deadline. On the other hand, for the general case in which a time, say t_1, can be both an arrival time and a deadline, our results can be easily extended by discussing the types of t_1.

Iterative function

Given an interval $[t_1, t_2]$ where $t_1, t_2 \in \mathcal{R} \cup \mathcal{D}$, we denote by $J(t_1, t_2)$ the job set that would be assigned to this interval. We define the assignment of jobs $J(t_1, t_2)$ as follows:

1. If $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{D}$, then assign all jobs with $[r_i, d_i] \subseteq [t_1, t_2]$ to $J(t_1, t_2)$.
2. If $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{R}$, then assign all jobs with $t_1 \leq r_i < t_2$ to $J(t_1, t_2)$.
3. If $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{D}$, then assign all jobs with $t_1 < d_i \leq t_2$ to $J(t_1, t_2)$.
4. If $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{R}$, then assign all jobs with $[r_i, d_i] \cap (t_1, t_2) \neq \emptyset$ to $J(t_1, t_2)$.

We denote by $E(t_1, t_2, k)$ the minimum cost (energy) among all feasible schedules that finish jobs $J(t_1, t_2)$ with exactly k evictions. Correspondingly, we use $E(t_1, t_2)$ to denote the minimum cost among all feasible schedules that finish the jobs $J(t_1, t_2)$ in the ideal model without considering memory operation. By abusing the notation, we also use $J(t_1, t_2)$ to denote the total workload of these jobs if the context is clear. Let $s_0(t_1, t_2) = \frac{J(t_1,t_2)}{t_2-t_1-kc}$ for every pair $t_1, t_2 \in \mathcal{R} \cup \mathcal{D}$. In the initialization iteration, set $E(t_1, t_2, k) = s_0(t_1, t_2)^\alpha (t_2 - t_1 - kc)$ if executing all jobs $J(t_1, t_2)$ with speed $s_0(t_1, t_2)$ in EDF order in interval $[t_1, t_2]$ is feasible. Otherwise, set $E(t_1, t_2, k) = \infty$. We will prove the following iterative function as stated in Lemma 4, which is crucial to our polynomial time algorithm.

Lemma 4 *If t_1, t_2 are tight times in OPT, then*
$$E(t_1, t_2, k) = \min_{t: t_1 < t < t_2, t \in \mathcal{R} \cup \mathcal{D}} \min_{0 \leq i \leq k} \{E(t_1, t, k-i) + E(t, t_2, i)\}$$
where $t_1, t_2 \in \mathcal{R} \cup \mathcal{D}$.

For simplicity, we will first explain it in the ideal model, and then extend the reasoning to the uniform with-cache model.

Dynamic programming algorithm in the ideal model

Note that Wu et al. (2009) derived an improved $O(n^2)$ time algorithm to compute the optimal solution for agreeable jobs in the ideal model. We restudy this problem and present an extendable dynamic programming algorithm to compute such a solution by losing an $O(n)$ factor of time. Since the ideal model is a special case of the non-cache model (where all $c_i = 0$), the lemmas proved in Sect. 3 still hold for the ideal model. Let the peak block in OPT be $[a, b]$; then a, b are tight. Denote by $E(t_1, t_2)$ the minimum cost among all feasible schedules that finish jobs $J(t_1, t_2)$. We derive the following execution and operation rules.

Lemma 5 ***Execution rule*** *If $[t_1, t_2]$ is a block in OPT, then OPT executes jobs $J(t_1, t_2)$ exactly in interval $[t_1, t_2]$. The corresponding speed in this interval is $\frac{J(t_1,t_2)}{t_2-t_1}$.*

Operation rule *If $[t_1, t]$, $[t, t_2]$ are two adjacent blocks in OPT, then $J(t_1, t) \cap J(t, t_2) = \emptyset$ and $J(t_1, t_2) = J(t_1, t) \cup J(t, t_2)$ for all $t_1 < t < t_2$, $t \in \mathcal{R} \cup \mathcal{D}$.*

Proof We first prove the execution rule. First, if $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{D}$, then t_1 is a tight arrival time and t_2 is a tight deadline by Lemma 1. For the agreeable jobs, OPT executes the jobs with no preemptions in EDF order by Lemma 1. Thus, all the jobs executed in $[t_1, t_2]$ have $r_1 \leq r_i < d_i \leq t_2$. The jobs $J(t_1, t_2)$ are $\{i : [r_i, d_i] \subseteq [t_1, t_2]\}$ in this case. Next, if $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{R}$, then this implies that t_1, t_2 are tight arrival times. Since OPT executes the jobs with no preemptions in EDF order, all the jobs with $t_1 \leq r_i < t_2$, and only these jobs, should be executed in $[t_1, t_2]$ in OPT. The case in which $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{D}$ can be similarly discussed. If $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{R}$, then t_1 is a tight deadline and t_2 is a tight arrival time. Exactly those jobs with $[r_i, d_i] \cap [t_1, t_2] \neq \emptyset$ should be executed in $[t_1, t_2]$. For all cases, OPT executes $J(t_1, t_2)$ in $[t_1, t_2]$. Obviously, the block $[t_1, t_2]$ has speed $\frac{J(t_1,t_2)}{t_2-t_1}$. Therefore, the execution rule is proved.

Next we focus on the operation rule. By Lemma 1 and its extension in the proof of Theorem 1, t_1, t, t_2 are tight. We will discuss this by the types of t_1, t_2.

First, if $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{D}$, then $J(t_1, t_2) = \{i : [r_i, d_i] \subseteq [t_1, t_2]\}$. If $t \in \mathcal{R}$, then $J(t_1, t) = \{i : t_1 \leq r_i < t\}$ and $J(t, t_2) = \{i : [r_i, d_i] \subseteq [t, t_2]\}$. Obviously, $J(t_1, t) \cap J(t, t_2) = \emptyset$ and $J(t_1, t_2) \subseteq J(t_1, t) \cup J(t, t_2)$. It is sufficient to prove that $\{i : t_1 \le r_i < t\} \cup \{i : [r_i, d_i] \subseteq [t, t_2]\} \subseteq \{i : [r_i, d_i] \subseteq [t_1, t_2]\}$. Note that t is a tight arrival time and t_2 is a tight deadline by Lemma 1. OPT has speed in $[t, t_2]$ greater than that in $[t_1, t]$. For the agreeable jobs, OPT executes the jobs with no preemptions in EDF order by Lemma 1. Therefore, there exists at least one job j with $t \le r_j < d_j \le t_2$. Thus, all the jobs with $t_1 \le r_i < t$ have $d_i \le t_2$, because the input is agreeable jobs. Hence, we have $J(t_1, t) \cup J(t, t_2) \subseteq J(t_1, t_2)$. If $t \in \mathcal{D}$, the case is similar by symmetrical handling of the arrival time and deadline.

Second, we consider the case $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{R}$. Here, $J(t_1, t_2) = \{i : t_1 \le r_i < t_2\}$. If $t \in \mathcal{R}$, then $J(t_1, t) = \{i : t_1 \le r_i < t\}$ and $J(t, t_2) = \{i : t \le r_i < t_2\}$. Obviously, $J(t_1, t_2) = \{i : t_1 \le r_i < t_2\} = \{i : t_1 \le r_i < t\} \cup \{t \le r_i < t_2\}$ and $J(t_1, t) \cap J(t, t_2) = \emptyset$. If $t \in \mathcal{D}$, then $J(t_1, t) = \{i : [r_i, d_i] \subseteq [t_1, t]\}$ and $J(t, t_2) = \{i : [r_i, d_i] \cap (t, t_2) \ne \emptyset\}$. Obviously, $J(t_1, t) \cap J(t, t_2) = \emptyset$ and $J(t_1, t_2) \subseteq J(t_1, t) \cup J(t, t_2)$. Since t_1 is a tight arrival time and t is a tight deadline, there exists a job with $t_1 \le r_{j'} < d_{j'} \le t$. We claim that all jobs $j \in J(t, t_2)$ have $r_j \ge t_1$, because otherwise $r_j < t_1 < t < d_j$, and the existence of a job with $t_1 \le r_{j'} < d_{j'} \le t$ contradicts "the input is agreeable jobs". Therefore, we have $J(t_1, t) \cup J(t, t_2) \subseteq J(t_1, t_2)$. Thus, $J(t_1, t) \cup J(t, t_2) = J(t_1, t_2)$ in this case.

Third, $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{D}$ can be handled symmetrically, similar to $t_1 \in \mathcal{R} \wedge t_2 \in \mathcal{R}$.

Lastly, we discuss the case $t_1 \in \mathcal{D} \wedge t_2 \in \mathcal{R}$. In this case, we have $J(t_1, t_2) = \{i : [r_i, d_i] \cap (t_1, t_2) \ne \emptyset\}$. If $t \in \mathcal{D}$, then $J(t_1, t) = \{i : t_1 < d_i \le t\}$ and $J(t, t_2) = \{i : [r_i, d_i] \cap (t, t_2) \ne \emptyset\}$. Clearly, $\{i : [r_i, d_i] \cap (t_1, t] \ne \emptyset\} \cup \{i : [r_i, d_i] \cap (t, t_2) \ne \emptyset\} \subseteq \{i : [r_i, d_i] \cap (t_1, t_2) \ne \emptyset\}$. We have $J(t_1, t) \cap J(t, t_2) = \emptyset$ and $J(t_1, t) \cup J(t, t_2) \subseteq \{i : [r_i, d_i] \cap (t_1, t] \ne \emptyset\} \cup \{i : [r_i, d_i] \cap (t, t_2) \ne \emptyset\} \subseteq J(t_1, t_2)$. It remains to be shown that $J(t_1, t_2) \subseteq J(t_1, t) \cup J(t, t_2)$. This is true, since a job in $J(t_1, t_2)$ with $t_1 < d_i \le t$ (or $d_i > t$) belongs to $J(t_1, t)$ (or $J(t, t_2)$). If $t \in \mathcal{R}$, this can be discussed similarly to the case $t \in \mathcal{D}$. Therefore, the lemma is true. □

We are now ready to present the iterative function for the ideal model in Lemma 6. Note that the iterative function computes $E(t_1, t_2)$ by summing two values $E(t_1, t)$, $E(t, t_2)$, which implicitly computes a schedule/cost for jobs $J(t_1, t) \cup J(t, t_2)$. When computing $E(r_{\min}, d_{\max})$, the final iteration computes a schedule for jobs $J(r_{\min}, t) \cup J(t, d_{\min})$. An implicit corollary by Lemma 5 is that jobs $J(r_{\min}, t) \cup J(t, d_{\min})$ in the final iteration are exactly equal to the input jobs $\mathcal{J}$.

Lemma 6 *If t_1, t_2 are tight times in OPT, then* $E(t_1, t_2) = \min_{t: t_1 < t < t_2, t \in \mathcal{R} \cup \mathcal{D}} E(t_1, t) + E(t, t_2)$.

Proof We prove this lemma by induction on the number of jobs in $\mathcal{J}$. When $|\mathcal{J}| = 1$, OPT is composed of one block. OPT can be computed in the initialization step by computing $E(r_{\min}, d_{\max})$. When $|\mathcal{J}| = 2$, OPT is composed of at most two blocks. These two blocks are separated at time $t \in \mathcal{R} \cup \mathcal{D}$. W.l.o.g., assume that t is a tight deadline in OPT, since the argument on the release time is symmetrical. Moreover, OPT executes jobs $J(r_{\min}, t)$ in interval $[r_{\min}, t]$ and jobs $J(t, d_{\max})$ in interval $[t, d_{\max}]$. The value $E(t, d_{\max})$ (or $E(r_{\min}, t)$) is computed by calculating $\frac{(J(t, d_{\max}))^\alpha}{(d_{\max} - t)^{\alpha - 1}}$ (or $\frac{(J(r_{\min}, t))^\alpha}{(t - r_{\min})^{\alpha - 1}}$) in the initialization step. The total value of these two blocks $E(r_{\min}, d_{\max})$ can be obtained in the second iteration when computing $\min_{t: r_{\min} < t < d_{\max}, t \in \mathcal{R} \cup \mathcal{D}} E(r_{\min}, t) + E(t, d_{\max})$. For the case in which there are k jobs, we assume the induction basis wherein the iterative function can compute the optimal schedule in $[t, d_{\max}]$ (or $[r_{\min}, t]$) when there are i jobs ($i \in \{1, \ldots, k-1\}$) being executed there. W.l.o.g., assume that t is a tight deadline in OPT. We have $t \in \mathcal{D}$ in this case. Assuming that $|J(r_{\min}, t)| = k_1$ where $1 \le k_1 < k$, and $|J(t, d_{\max})| = k_2$ where $k_2 = k - k_1$, OPT executes jobs $J(r_{\min}, t)$ in interval $[r_{\min}, t]$ and jobs $J(t, d_{\max})$ in interval $[t, d_{\max}]$. The value $E(t, d_{\max})$ (or $E(r_{\min}, t)$) can be computed by the induction basis since $1 \le k_1, k_2 \le k - 1$. By enumerating all possible times $t \in \mathcal{D}$, the minimum value of $E(r_{\min}, t) + E(t, d_{\max})$ among all $r_{\min} < t < d_{\max}$ is $E(r_{\min}, d_{\max})$, exactly the cost of OPT.

Now we extend the setting to the case in which t_1, t_2 are the input in function $E(\cdot, \cdot)$ (instead of $r_{\min}, d_{\max}$). If t is a tight time in OPT, then OPT executes jobs $J(t_1, t)$ in interval $[t_1, t]$ and jobs $J(t, t_2)$ in interval $[t, t_2]$. The same proof as stated above can be used to show $E(t_1, t_2) = \min_{t: t_1 < t < t_2, t \in \mathcal{R} \cup \mathcal{D}} E(t_1, t) + E(t, t_2)$. □

With the iterative function in hand, we are now ready to present the proposed Algorithm 2. Theorem 6 concludes the results.

Theorem 6 *For the ideal model with agreeable jobs, OPT can be computed by the dynamic programming algorithm in $O(n^3)$ time.*

Proof As shown in Algorithm 2, in the initialization iteration, we set $E(t_1, t_2) = s_0(t_1, t_2)^\alpha (t_2 - t_1)$ if executing all jobs $J(t_1, t_2)$ with speed $s_0(t_1, t_2)$ in EDF order in interval $[t_1, t_2]$ is feasible. Otherwise, we set $E(t_1, t_2) = \infty$. Note that the optimal solution is composed of blocks where the starting and finishing time of the block are tight. Function $E(t_1, t_2)$ computes the minimum possible energy of an interval $[t_1, t_2]$ with the implicit assumption that t_1, t_2 are tight. To find the optimal solution, we need only compute $E(r_{\min}, d_{\max})$.

Assume that $t_1, t_2, \ldots, t_{2n}$ are the times in $\mathcal{R} \cup \mathcal{D}$ with non-decreasing order. The energy of the optimal schedule can be computed by solving $E(r_{\min}, d_{\max}) = E(t_1, t_{2n})$. In iteration i, the algorithm computes a schedule with minimal cost for an interval that is crossing i consecutive times

Algorithm 2 Dynamic Programming Algorithm

```
Assume that $t_1, t_2, \dots, t_{2n}$ are the times in $\mathcal{R} \cup \mathcal{D}$ with non-decreasing order. Denote by $C(t_u, t_{u+i})$ $(u, u+i \in [1, 2n])$ the minimal cost to execute jobs $J(t_u, t_{u+i})$ in interval $[t_u, t_{u+i}]$.
Define sub-routine $F(t_u, t_{u+i})$ to be value $\infty$ if it is infeasible to execute $J(t_u, t_{u+i})$ with speed $\frac{J(t_u,t_{u+i})}{t_{u+i}-t_u}$ in interval $[t_u, t_{u+i}]$ or $\frac{J(t_u,t_{u+i})^\alpha}{(t_{u+i}-t_u)^{\alpha-1}}$ if feasible.
for iteration $i = 1, 2, \dots, 2n$ do
    for each value $u = 1, \dots, 2n-i$ do
        Compute value $F(t_u, t_{u+i})$.
        Update $C(t_u, t_{u+i}) = \min_{1 \le j \le i} \{C(t_u, t_{u+j}) + C(t_{u+j}, t_{u+i}), F(t_u, t_{u+i})\}$.
    end for
end for
Return $C(t_1, t_{2n})$.
```

in $\{t_1, t_2, \dots, t_{2n}\}$. We first consider the case in which $t_1 < t_2 < \cdots < t_{2n}$, i.e., every two jobs have different arrival times or deadlines. Denote by $C(t_u, t_{u+i})$ $(u, u+i \in [1, 2n])$ the minimal cost to execute jobs $J(t_u, t_{u+i})$ in interval $[t_u, t_{u+i}]$. Computing each value $F(\cdot, \cdot)$ needs to enumerate the inner index j, which uses $O(n)$ time. There are $O(n^2)$ items of $F(\cdot, \cdot)$. Therefore, the complexity of the algorithm is $O(n^3)$. The optimal schedule can be constructed by keeping the information of every iteration.

For the general case in which $t_1 \le t_2 \le \cdots \le t_{2n}$, we need to distinguish the function between $C(t_a, t_b)$ and $C(t_{a'}, t_b)$ (or between $C(t_b, t_a)$ and $C(t_b, t_{a'})$) if the two times t_a, t'_a have equal value, but one is an arrival time and the other is a deadline. For example, given interval $[t_a, t_b]$ with $t_a < t_b$, job i_1 has a deadline at t_a and job i_2 has an arrival time at t_a. There are two cases. One is that OPT has a tight deadline at time t_a, and the other is that OPT has a tight arrival time at time t_a. For these two cases, the operation rule is different for function $J(t_a, t_b)$, and hence also for function $C(t_a, t_b)$. Thus, we would distinguish the two kinds of operations by indexing the type of t_a. This allows the algorithm to return the optimal solution. This finishes the proof. □

Remark Note that in the implementing Algorithm 2, it needs to index the type of time t (deadline or arrival time) to distinguish the operations over it when $t_1 \le t_2 \le \cdots \le t_{2n}$. The other, simpler method is to replace the notation $C(t_u, t_{u+i})$, $F(t_u, t_{u+i})$ with $C(u, u+i)$, $F(u, u+i)$ (using a different index to distinguish the type of the time) in the definition and further set $C(u, u+i) = F(u, u+i) = \infty$ if $t_u = t_{u+i}$.

Dynamic programming algorithm in the uniform with-cache model:

Finally, we extend the results to the uniform with-cache model. With the same proofs, it is easy to show that the properties in Lemmas 1 and 5 still hold for the optimal schedule in the uniform with-cache model.

Based on these results, the extension is then simple, where we mainly update the iterative function for the ideal model to be the form in Lemma 4. The running time needs another $O(n)$ factor compared to Theorem 6, since there is one more inner loop in the updated iterative function.

Theorem 7 *For agreeable jobs in the uniform with-cache model, OPT can be computed in $O(n^4)$ time.*

Proof Lemma 5 can be directly extended to the uniform with-cache model. If $[t_1, t_2]$ is a block in OPT, then OPT executes the jobs $J(t_1, t_2)$ exactly in interval $[t_1, t_2]$. This follows Lemma 5 directly. The workload executed in this block has one speed throughout the whole interval of $[t_1, t_2]$, excluding the eviction time. When the number of evictions (let it be k) in $[t_1, t_2]$ is known, since no preemption is generated when the agreeable jobs are executed in EDF order, the total length of eviction time in this interval is exactly kc. Thus, the speed in this interval is $\frac{J(t_1,t_2)}{t_2-t_1-kc}$.

Therefore, we should set $E(t_1, t_2, i) = s_0(t_1, t_2)^\alpha(t_2-t_1-ic)$ in the initialization where $0 \le i \le K$ if executing all jobs $J(t_1, t_2)$ with speed $s_0(t_1, t_2)$ and i evictions in EDF order in interval $[t_1, t_2]$ is feasible. Otherwise, we set $E(t_1, t_2, i) = \infty$. There are $O(n^2)$ pairs of time t_1, t_2.

Based on such revisions, we can extend the recursive function stated in Lemma 6 to obtain the recursive function for the uniform with-cache model (Lemma 4). The proof paradigm is almost the same, the only difference being that we need to further enumerate the possible number of evictions in the inner loop.

With the recursive function in Lemma 4, we can implement a dynamic programming algorithm to compute the minimum energy for the uniform with-cache model. The energy of the optimal schedule can be computed by solving $E(r_{\min}, d_{\max}, K)$. The final dynamic programming algorithm has complexity of $O(n^4)$, since it spends another factor $O(n)$ time compared to $O(n^3)$ time in Algorithm 2 to enumerate the eviction times in the loops. The optimal schedule can be constructed by keeping the information of every iteration. □

4.2 With-cache model: approximation algorithm for general jobs with resource augmentation

Theorem 5 shows that optimizing the energy for the with-cache model is NP-complete. In this section, we study the approximation algorithms in the resource augmentation setting, i.e., the g-speed with-cache model.

Note that the definition of approximation algorithms for the resource augmentation setting implicitly indicates that the 1-speed with-cache model has a feasible solution. That is, given the input jobs and K, there is a feasible schedule which completes all the jobs in time and uses only K evictions. We will design an algorithm that either shows that the input is infeasible for the 1-speed with-cache model or outputs a feasible solution for the g-speed with-cache model that incurs energy of at most c times that of the optimal solution in the 1-speed with-cache model.

Define $\mathcal{V} = \{[t_1, t_1 + l_1), [t_2, t_2 + l_2), \ldots, [t_k, t_k + l_k)\}$ to be the configuration of the eviction intervals where $[t_i, t_i + l_i)$ is used for job eviction and $t_i + l_i \le t_{i+1}$. Given $\mathcal{V}$ and jobs $\mathcal{J}$, a schedule $S^{\mathcal{J}}_{\mathcal{V}}$ should not execute any workload of $\mathcal{J}$ in the eviction intervals defined by $\mathcal{V}$. Denote by $h(i) = \frac{w_i}{d_i - r_i}$ the intensity of job i. In the with-cache model, we define $h_{\mathcal{V}}(i)$ as the intensity of job i excluding the eviction intervals $\mathcal{V}$. That is, $h_{\mathcal{V}}(i) = \frac{w_i}{d_i - r_i - \int_{t=r_i}^{d_i} \rho(\mathcal{V}, t) \mathrm{d}t}$, where indication function $\rho(\mathcal{V}, t)$ denotes whether or not t belongs to the eviction intervals.

Algorithm 3 Compute the configuration $\mathcal{V}$

```
Compute that maximum throughput $m$ for problem $\mathcal{P}$ using the dynamic programming algorithm in Baptiste (1999). Denote the computed schedule to be $\bar{S}$ and the intervals (for the execution of $m$ jobs in $\mathcal{H}$) generated in the schedule to be $\bar{\mathcal{V}}$.
if $m < K$ then
    return that the 1-speed with-cache model is infeasible.
else
    Choose $K$ jobs with the shortest eviction time. Denote by $\mathcal{V}$ the interval configuration induced by $\bar{\mathcal{V}}$ which is only used for eviction for the selected $K$ jobs.
end if
Return $\mathcal{V}$.
```

We first show an algorithm which determines whether the instance for the 1-speed with-cache model is infeasible. If not, it further returns a configuration of the eviction intervals for future use. We use $\mathcal{P}$ to denote the following problem. Given the input instance $\mathcal{J}$ for the with-cache model, let $\mathcal{H} = \{J_i = (r_i, d_i, c_i), i \in \mathcal{J}\}$ be the induced job instance where $w_i = 0$ for all i. For job J_i in $\mathcal{H}$, we need to allocate c_i units of time in its alive interval $[r_i, d_i]$, and each unit of time is assigned to at most one job. The objective is to maximize the number of jobs that are allocated without conflict. We observe that this problem is in fact $1|r_j, pmtn| \sum U_j$ in the scheduling literature. Baptiste (1999) shows that the optimal solution of $\mathcal{P}$ can be computed by dynamic programming in $O(n^4)$ time. Our algorithm adopts their result. A feasible schedule S (for the 1-speed with-cache model) which allocates K evictions for $\mathcal{J}$ implies the existence of a feasible schedule $\bar{S}$ which completes K jobs in $\mathcal{H}$. If the problem $\mathcal{P}$ has maximum number of jobs m with $m < K$, then this shows that there is no feasible schedule for the 1-speed with-cache model with K evictions. We return a configuration $\mathcal{V}$ for the remaining case $K \le m$. Note that $K \le m$ does not necessarily indicate the feasibility of the 1-speed with-cache model. However, we will prove that there exists an $O(1)$-approximation algorithm with g-speed resource augmentation. Now we present our algorithm that returns a schedule $AVR^{\mathcal{J}}_{\mathcal{V}'}$, and the performance of the algorithm is proved in Theorem 8.

Algorithm 4 Schedule with performance guarantee

```
1. Let $\mathcal{V}$ be the configuration returned by Algorithm 3.
2. Let $\mathcal{V}'$ be the configuration induced by $\mathcal{V}$ where every job's eviction time can accelerate by $s$ times.
3. Compute $h_{\mathcal{V}'}(i)$ for each job $i$. Schedule each job $i$ with speed $h_{\mathcal{V}'}(i)$ in the non-eviction interval of $\mathcal{V}'$ in EDF order. Denote the resulting schedule as $AVR^{\mathcal{J}}_{\mathcal{V}'}$.
```

Theorem 8 *Algorithm 4 is $(2\alpha \frac{g}{g-1})^{\alpha}/2$-approximation for the g-speed with-cache model.*

Proof Algorithm AVR was first proposed in Yao et al. (1995) for the ideal model. It executes the jobs in EDF order at speed $s(t) = \sum_i h(i) \cdot alive(i, t)$ where indication function $alive(i, t)$ equals 1 if $t \in [r_i, d_i)$, and 0 otherwise. Clearly, AVR is feasible for jobs in the ideal model. We adopt the idea for algorithm AVR, and our schedule $AVR^{\mathcal{J}}_{\mathcal{V}}$ executes the jobs in EDF order at speed $s(t) = \sum_i h_{\mathcal{V}}(i) \cdot alive(i, t) \cdot \overline{\rho(\mathcal{V}, t)}$. Obviously, the speed is zero for the time that belongs to the eviction intervals. We observe two useful properties of algorithm $AVR^{\mathcal{J}}_{\mathcal{V}}$ below.

First, assuming that $\mathcal{V}, \mathcal{V}'$ are two configurations with $\mathcal{V} \subseteq \mathcal{V}'$, and job i in $\mathcal{J}$ has $h_{\mathcal{V}'}(i) \le r \cdot h_{\mathcal{V}}(i)$, then we have $AVR^{\mathcal{J}}_{\mathcal{V}'} \le r^{\alpha} \cdot AVR^{\mathcal{J}}_{\mathcal{V}}$. This is because $AVR^{\mathcal{J}}_{\mathcal{V}'} = \int_{t=0}^{\infty} (\sum_i h_{\mathcal{V}'}(i) \cdot alive(i, t) \cdot \overline{\rho(\mathcal{V}', t)})^{\alpha} \mathrm{d}t \le \int_{t=0}^{\infty} (\sum_i r \cdot h_{\mathcal{V}}(i) \cdot alive(i, t) \cdot \overline{\rho(\mathcal{V}', t)})^{\alpha} \mathrm{d}t \le \int_{t=0}^{\infty} (\sum_i r \cdot h_{\mathcal{V}}(i) \cdot alive(i, t) \cdot \overline{\rho(\mathcal{V}, t)})^{\alpha} \mathrm{d}t \le r^{\alpha} \int_{t=0}^{\infty} (\sum_i h_{\mathcal{V}}(i) \cdot alive(i, t) \cdot \overline{\rho(\mathcal{V}, t)})^{\alpha} \mathrm{d}t = r^{\alpha} AVR^{\mathcal{J}}_{\mathcal{V}}$, where the first step follows from the assumption that $h_{\mathcal{V}'}(i) \le r \cdot h_{\mathcal{V}}(i)$ and the second step holds by $\overline{\rho(\mathcal{V}', t)} \le \overline{\rho(\mathcal{V}, t)}$, since $\mathcal{V} \subseteq \mathcal{V}'$. Second, if $\mathcal{V}'$ is the configuration induced by $\mathcal{V}$ (returned by Algorithm 3), where every job's eviction time can accelerate by g times, then we have $h_{\mathcal{V}'}(i) \le \frac{g}{g-1} \cdot h(i)$. We show the reasoning below. Note that the configuration $\mathcal{V}$ computed in Algorithm 3 uses K evictions. Moreover, each unit of time is only used by at most one job for eviction. Thus, for each job i, the total length of eviction time in interval $[r_i, d_i]$ induced by $\mathcal{V}$ is at most $d_i - r_i$. In the g-speed augmentation setting, the total length of eviction time in interval $[r_i, d_i]$ induced by $\mathcal{V}'$ is at most $\frac{d_i - r_i}{g}$. Thus, the interval that can be used for job execution has a length of at least $d_i - r_i - \frac{d_i - r_i}{g}$. We have $h_{\mathcal{V}'}(i) \le \frac{w_i g}{(d_i - r_i)(g-1)} \le \frac{g}{g-1} h(i)$ where $h(i) = \frac{w_i}{d_i - r_i}$.

Now we prove that $AVR^{\mathcal{J}}_{\mathcal{V}'}$ is a $(2\alpha \frac{g}{g-1})^{\alpha}/2$-approximation. Let $\emptyset$ be the configuration with total eviction time of length 0. Each job i has $h_{\emptyset}(i) = h(i)$ in $AVR^{\mathcal{J}}_{\emptyset}$. Clearly, $\emptyset \subseteq \mathcal{V}'$.

We have $AVR_{\mathcal{V}'}^{\mathcal{J}} \leq (\frac{g}{g-1})^{\alpha} AVR_{\emptyset}^{\mathcal{J}}$ by combining the two properties above. Observe that $AVR_{\emptyset}^{\mathcal{J}} \leq \frac{(2\alpha)^{\alpha}}{2}\text{OPT}_{\emptyset}^{\mathcal{J}}$ by Yao et al. (1995) and Bansal et al. (2008). Thus, $AVR_{\mathcal{V}'}^{\mathcal{J}} \leq (2\alpha\frac{g}{g-1})^{\alpha}/2 \cdot \text{OPT}_{\emptyset}^{\mathcal{J}}$. Finally, letting $\mathcal{V}_{\text{opt}}$ be the optimal configuration of eviction time in the optimal schedule for the with-cache model, we have $AVR_{\mathcal{V}'}^{\mathcal{J}} \leq (2\alpha\frac{g}{g-1})^{\alpha}/2 \cdot \text{OPT}_{\mathcal{V}_{\text{opt}}}^{\mathcal{J}}$. This is because the optimal energy consumption without eviction time is strictly less than that for the optimal configuration $\mathcal{V}_{\text{opt}}$. Hence, the theorem is proved. □

5 Conclusion

We have studied the DVS-based energy minimization problem in additional practical models involving memory operation time, and we have presented several optimal algorithms for the non-cache and with-cache models. Minimizing the energy consumption in the general with-cache model is proved NP-complete, and our approximation algorithm relies on a g-speed resource augmentation; thus, one possible future direction is studying the approximation algorithm without such resource augmentation.

Acknowledgements This work is supported in part by the National Natural Science Foundation of China under Grants No. 61727809, No. 61572342, No. 61672154, Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU11268616), Natural Science Foundation of Jiangsu Province under Grant No. BK20151240.

Appendix A: Proof of Lemma 1

Proof Property (P1) can be proved by standard swapping techniques. For example, for two jobs with deadlines $d_{j_1} \leq d_{j_2}$, suppose on the contrary that in a schedule these two jobs finish at time t_{j_2}, t_{j_1} with $t_{j_2} < t_{j_1}$. Clearly, $t_{j_2} < t_{j_1} \leq d_{j_1} \leq d_{j_2}$ by the feasibility of the schedule. We can then obtain a new schedule by swapping the execution order of j_1, j_2 while keeping the speed function unchanged. The new schedule would make j_1 finish earlier than j_2 and j_2 finish at time t_{j_1} with $t_{j_1} \leq d_{j_2}$. Thus, the deadline constraints of these two jobs are not violated. Moreover, the energy consumption stays the same, since the speed function is not changed. Thus, applying such transformations, we can transform any feasible schedule into EDF order.

We now prove (P2) by contradiction. Suppose on the contrary that job i is executed with more than two speeds in the optimal schedule. We choose two small intervals (with the same length $\epsilon \to 0$) that are used to execute job i with speeds $s_1 > s_2 > 0$. We could move workload $\frac{s_1-s_2}{2}\epsilon$ of i from the interval with speed s_1 to the interval with speed s_2 so that the two intervals will have the same speed $\frac{s_1+s_2}{2}$. According to the convexity of the power function, this deceases the energy consumption and does not violate the feasibility of the schedule, which leads to a contradiction. Therefore, in the optimal solution, every job has a unique speed in the optimal solution. Accordingly, the optimal solution is composed of blocks.

Now we focus on property (P3). Assume that OPT has speed s_1 in $[a, b]$ which is the maximum speed in OPT. Let the job that is executed (or virtually executed) at the end of block $[a, b]$ be j_1. Assuming on the contrary that b is not a tight deadline, we have $d_{j_1} > b$. We will select a small interval $[a_0, a_0 + \epsilon']$ with length $\epsilon' \to 0$ as defined below. If the end of block $[a, b]$ is executing a virtual workload, then we set $[a_0, a_0 + \epsilon']$ to be the latest interval that executes j_1's workload (not virtual) with speed s_1 (this small interval exists because j_1 has virtual speed s_1). Otherwise, we set $a_0 + \epsilon' = b$.

We will discuss two cases of interval $[b, d_{j_1}]$. If OPT has a small interval $[a', b']$ with $b \leq a' < b' \leq d_{j_1}$ that executes some workload with speed $s_2 < s_1$, then moving a partial workload of j_1 in $[a_0, a_0 + \epsilon']$ to $[a', a' + \epsilon']$ (resulting in the same speed in these two blocks) reduces the energy by Fact 1. The resulting schedule is feasible, which contradicts the optimality of OPT. Consider the remaining case that the whole interval $[b, d_{j_1}]$ is used for eviction or jobs executed in interval $[b, d_{j_1}]$ have speed s_1. Then OPT must allocate an eviction interval $[b, v]$ with virtual speed $s_2 < s_1$. Such $[b, v]$ exists, because otherwise the peak block is $[a, v]$ instead of $[a, b]$. Suppose that OPT allocates job j_2's eviction time at interval $[v - \epsilon, v]$ with $\epsilon < \epsilon'$. According to the definition of virtual speed, j_2's workload is executed at speed s_2 in some interval which should be outside $[a, d_{j_1}]$, since we have shown that there are no other jobs that are executed in interval $[a, d_{j_1}]$ at a speed lower than s_1. W.l.o.g., assume that interval $[a', b']$ with $d_{j_1} \leq a' < b' \leq d_{j_2}$ is used to execute j_2's workload. We will prove by contradiction that if b is not a tight deadline, we can obtain another transformed schedule with less energy consumption. The transformation attempts to change the original two chosen intervals with speeds s_1, s_2 to three intervals with speed s_3, s_3, s_4, which follows $s_1 > s_3 > s_4 > s_2$, so that the energy consumption is reduced by the convexity of the power function. The details are as follows. We focus on one virtual interval $[v - \epsilon, v]$ and two intervals $[a_0, a_0+\epsilon']$ and $[a', b']$. We then apply the following transformation procedure:

1. Move all workloads in $[a', a' + \epsilon]$ to $[a' + \epsilon, b']$.
2. Swap job $j_2's$ eviction interval $[v - \epsilon, v]$ with the vacant interval $[a', a' + \epsilon]$.
3. Move partial workload of $[a_0, a_0+\epsilon']$ to the vacant interval $[v - \epsilon, v]$ so that these two blocks have the same speed (let the speed be s_3). Ensure that the speed s_3 in $[a_0, a_0 + \epsilon']$, $[v - \epsilon, v]$ is larger than the speed (s_4) in $[a', a' + \epsilon]$ by selecting a small ϵ.

After the transformation, we obtain one virtual interval $[a', a'+\epsilon]$ and the following three intervals: $[a_0, a_0+\epsilon']$ with speed s_3, $[v-\epsilon, v]$ with speed s_3, and $[a'+\epsilon, b']$ with speed s_4. The total energy in these intervals after the transformation is $s_3^\alpha \cdot (\epsilon' + \epsilon) + s_4^\alpha (b' - a' - \epsilon)$. Obviously, the resulting schedule is still feasible for all jobs after transformation. The total energy in intervals $[a_0, a_0 + \epsilon']$ and $[a', b']$ before the transformation is $s_1^\alpha \cdot \epsilon' + s_2^\alpha (b' - a')$. Through such a chain-like transformation procedure, we can see that the energy is reduced by the property $s_1 > s_3 > s_4 > s_2$ and the convexity of the power function. Thus, all cases lead to a contradiction. Therefore, b is a tight deadline. Symmetrically, we can prove that a is a tight arrival time using similar deduction. This proves property (P3). □

Appendix B: Algorithm of optimal discrete voltage schedule in Section 3.2

Algorithm 5 Two-Level Schedule

```
Input: speed $s_1$,$s_2$ ($s_1 > s_2$) and job set $\mathcal{J}$
Output: $(s_1, s_2)$-schedule for $\mathcal{J}$.
Variables:
Committed: the list of allocated time intervals.
Committed(i): the time intervals allocated to job $J_i$.
$I_k^{s_1}$: the union of time intervals which are assigned to job $J_k$ for job
processing or memory operation in $s_1$-schedule.
 Run $s_1$-schedule for $\mathcal{J}$ to obtain $I_k^{s_1}$ for $k = 1, \ldots, n$.
 Run $s_2$-schedule for $\mathcal{J}$ to obtain $I_k^{s_2}$ for $k = 1, \ldots, n$.
 $Committed \leftarrow \emptyset$
for $i = n$ downto 1 do
    1. $I = I_i^{s_2} - Committed$
    2. Take $I' \subseteq I_i^{s_1}$ of appropriate length (possibly 0) from the right
       end of $I_i^{s_1}$ to obtain an $(s_1, s_2)$-schedule for $J_i$ over $I \cup I'$
    3. Assign memory time of job $J_i$ at the beginning of $I \cup I'$
    4. $Committed(i) = I \cup I'$
    5. $Committed \leftarrow Committed \cup Committed(i)$
end for
```

Theorem 9 *In Algorithm 5, the algorithm maintains the following properties at the beginning of iteration i:*
(1) $Committed(i+1) \subseteq I_{i+1}^{s_1} \cup I_{i+1}^{s_2}$
(2) $\bigcup_{k=i+1}^n I_k^{s_2} \subseteq Committed \subseteq (\bigcup_{k=i+1}^n I_k^{s_1}) \cup (\bigcup_{k=i+1}^n I_k^{s_2})$
(3) $Committed \cap (\bigcup_{k=1}^i I_k^{s_1}) = \emptyset$
(4) $Committed \cap I_i^{s_1} = \emptyset$.

Proof In iteration i, time intervals assigned to job J_i include all time intervals of $I_i^{s_2}$ and some time intervals (explained in more detail later) from $I_i^{s_1}$; therefore, (1) and (2) are correct. For (3), first, $(\bigcup_{k=1}^i I_k^{s_1})$ is disjoint with $(\bigcup_{k=i+1}^n I_k^{s_1})$, and $(\bigcup_{k=1}^i I_k^{s_2})$ is disjoint with $(\bigcup_{k=i+1}^n I_k^{s_2})$. Second, $(\bigcup_{k=1}^i I_k^{s_1})$ is contained in $(\bigcup_{k=1}^i I_k^{s_2})$ (which can be proved by induction; one may refer to Li and Yao (2005) for additional detail), which results in $\bigcup_{k=1}^i I_k^{s_1} \cap (\bigcup_{k=i+1}^n I_k^{s_1} \cup \bigcup_{k=i+1}^n I_k^{s_2}) = \emptyset$. Combined with (2), (3) is correct. (4) is obvious from (3). □

Correctness of Algorithm 5 The high-level idea is to assign job J_i the remaining intervals from its s_2-schedule first. If the allocated time is sufficient to finish J_i at speed s_1 ad s_2, then we move on to the next iteration. Otherwise (the time is not sufficient even if we run the job at high speed s_1 constantly), we extract some time from J_i's s_1-schedule.

In iteration i, time intervals assigned to job J_i are from $(I_i^{s_2} \cup I_i^{s_1}) - Committed$. We first explain Step 2, and we discuss two extreme cases. In the first case, we take $Committed(i) = I$ and use constant speed s_2 for job J_i. Apparently, this schedule may not be feasible, but it contains no idle time. In the second case, we take $Committed(i) = I \cup I_i^{s_1}$ and use constant speed s_1 for job J_i. This schedule must be feasible for job J_i by property (4) of Theorem 9. Combined with these two cases, an (s_1, s_2)-schedule must exist. Once $Committed(i)$ is fixed, an (s_1, s_2)-schedule could be obtained by solving the linear equations in Theorem 4 for job J_i. We set $Committed(i) = I$ if $s_1 \cdot (|Committed(i)| - c_i) \geq w_i$; otherwise we set $Committed(i) = I \cup I'$ such that $s_1 \cdot (|Committed(i)| - c_i) = w_i$, where $I' \subseteq I_i^{s_1}$ is taken from the right side of $I_i^{s_1}$. Consequently, no idle time interval is produced by the algorithm, and we obtain a feasible (s_1, s_2)-schedule for job J_i.

References

Albers, S. (2010). Energy-efficient algorithms. *Communications of the ACM*, *53*(5), 86–96.

Albers, S., & Antoniadis, A. (2014). Race to idle: New algorithms for speed scaling with a sleep state. *ACM Transactions on Algorithms (TALG)*, *10*(2), 9.

Albers, S., Antoniadis, A., & Greiner, G. (2015). On multi-processor speed scaling with migration. *Journal of Computer and System Sciences*, *81*(7), 1194–1209.

Antoniadis, A., Huang, C. C., & Ott, S. (2015). 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* (pp. 1102–1113).

Aydin, H., Devadas, V., & Zhu, D. (2006). System-level energy management for periodic real-time tasks. In *Proceedings of the 27th IEEE real-time systems symposium* (pp. 313–322).

Bambagini, M., Marinoni, M., Aydin, H., & Buttazzo, G. (2016). Energy-aware scheduling for real-time systems: A survey. *ACM Transactions on Embedded Computing Systems (TECS)*, *15*(1), 7.

Bansal, N., Bunde, D. P., Chan, H. L., & Pruhs, K. (2008). Average rate speed scaling. In *Proceedings of the 8th Latin American theoretical informatics symposium, volume 4957 of LNCS* (pp. 240–251).

Bansal, N., Kimbrel, T., & Pruhs, K. (2004). Dynamic speed scaling to manage energy and temperature. In *Proceedings of the 45th annual symposium on foundations of computer science* (pp. 520–529).

Baptiste, P. (1999). An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, *24*(4), 175–180.

Bini, E., Buttazzo, G., & Lipari, G. (2005). Speed modulation in energy-aware real-time systems. In *IEEE proceedings of the 17th Euromicro conference on real-time systems* (pp. 3–10).

Choi, K., Soma, R., & Pedram, M. (2005). Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, *24*(1), 18–28.

Hong, I., Qu, G., Potkonjak, M., & Srivastavas, M. B. (1998). Synthesis techniques for low-power hard real-time systems on variable voltage processors. In *Proceedings of the IEEE real-time systems symposium* (pp. 178–187).

Hsu, C. H., & Feng, W. C. (2004). Effective dynamic voltage scaling through CPU-boundedness detection. *In the 4th IEEE/ACM workshop on power-aware computing systems* (pp. 135–149).

Irani, S., Shukla, S., & Gupta, R. K. (2007). Algorithms for power savings. *Journal ACM Transactions on Algorithms*, *3*(4), 41.

Ishihara, T., & Yasuura, H. (1998). *Voltage scheduling problem for dynamically variable voltage processors*. In *Proceedings. 1998 International Symposium on low power electronics and design, 1998*. IEEE.

Li, M., & Yao, F. (2005). An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal on Computing*, *35*(3), 658–671.

Seth, K., Anantaraman, A., Mueller, F., & Rotenberg, E. (2003). Fast: Frequency-aware static timing analysis. In *Proceedings of the 24th IEEE real-time system symposium* (pp. 40–51).

Wu, W., Li, M., & Chen, E. (2009). Min-energy scheduling for aligned jobs in accelerate model. In *Proceedings of 20th international symposium on algorithms and computation (ISAAC 09)* (pp. 462–472).

Yang, C. Y., Chen, J. J., & Kuo, T. W. (2007). Preemption control for energy-efficient task scheduling in systems with a DVS processor and non-DVS devices. In *Proceedings of the 13th IEEE international conference on embedded and real-time computing systems and applications* (pp. 293–300).

Yao, F., Demers, A., & Shenker, S. (1995). A scheduling model for reduced CPU energy. In *Proceedings of IEEE symposium on foundations of computer science (FOCS)* (pp. 374–382).