Competitive Online Algorithms for Multiple-Machine Power Management and Weighted Flow Time

Ho-Leung Chan **Sze-Hang Chan** **Tak-Wah Lam** * **Lap-Kei Lee**
Rongbin Li **Chi-Man Liu**

Department of Computer Science,
University of Hong Kong
Email: {hlchan, shchan, twlam, lklee, rbli, cmliu}@cs.hku.hk

Abstract

We consider online job scheduling together with power management on multiple machines. In this model, jobs with arbitrary sizes and weights arrive online, and each machine consumes different amount of energy when it is processing a job, idling or sleeping. A scheduler has to maintain a good balance of the states of the machines to avoid energy wastage, while giving an efficient schedule of the jobs. We consider a recently well-studied objective of minimizing the total weighted flow time of the jobs plus the total energy usage. For the special case where all jobs have the same weight, competitive algorithms have been obtained (Lam et al. 2009, Chan et al. 2011). This paper gives a non-trivial potential analysis of a weighted generalization of the power management algorithm in (Chan et al. 2011), coupled with a classic scheduling algorithm HDF. This leads to the first competitive result for minimizing weighted flow time plus energy. The result can be extended to the dynamic speed scaling model where the scheduler can vary the speed of individual machines to process the jobs and the energy usage depends on the speed of the machines.

Keywords: sleep management, weighted flow time, energy efficiency, potential analysis.

1 Introduction

Server farms with hundreds to thousands of machines are common nowadays, and energy consumption has become an important concern. It has been reported that the energy cost of running a machine for three years indeed exceeds the hardware cost of the machine (Belady 2007). When a machine is turned on, its energy consumption consists of the *static* power, which is consumed even when the machine is idle, and the *dynamic* power, which is the extra power used for processing a job. For example, an Intel Xeon E5320 server consumes 150W of energy when idling and 240W when working. To reduce energy usage, some machines can be put to the sleep state in which energy consumption is reduced to essentially zero. However, transiting a machine from the sleep state to the awake state requires an extra amount of *wake-up* energy, and it is energy inefficient to frequently switch the machines on and off. In this paper, we are interested in online algorithms for job scheduling that can optimize the energy consumption of a pool of machines as well as certain QoS (Quality of Service) measure of the schedule.

Weighted flow time. A commonly used QoS measure is the total weighted flow time (or simply weighted flow) of the jobs. Formally speaking, jobs arrive online at different times in an unpredictable manner. Each job j has a size $p(j)$ and weight $w(j)$. The weight $w(j)$ reflects the importance of job j. The flow time of a job is the length of the duration from its arrival until its completion, and its weighted flow time is simply its flow time multiplied by its weight. We assume that all jobs are sequential, i.e., a job can be processed by at most one machine at any time. We consider preemptive scheduling and a preempted job can be resumed at the point of preemption. Migration incurs overhead in practice, so we only consider non-migratory schedules in which each job is dispatched to one machine and is run entirely on that machine.

Minimizing weighted flow is fundamentally harder than minimizing unweighted flow even when energy usage is not a concern: On a single machine, it is well-known that the algorithm SRPT (Shortest Remaining Processing Time) is 1-competitive for unweighted flow time, while Bansal and Chan (Bansal et al. 2009) showed that no online algorithm can be $O(1)$-competitive for weighted flow time. To overcome the hardness, most studies on weighted flow time (without energy concern) consider resource augmentation where the online algorithm is given faster machines. Precisely, an s-speed machine can process s units of work in one unit of time. On a single machine, Becchetti et al. (Becchetti et al. 2006) showed that the greedy algorithm HDF (Highest Density First) is $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-competitive for any $\epsilon > 0$, i.e., its total weighted flow time when given a $(1+\epsilon)$-speed machine is at most $(1+\frac{1}{\epsilon})$ times that of the optimal offline schedule on a 1-speed machine. Note that the density of a job is its weight divided by its size. HDF always schedules the jobs with the highest density and it requires migration when there are more than one machine. On multiple machines, Becchetti et al. (Becchetti et al. 2006) also showed that HDF is $(2+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-competitive when migration is allowed. The first non-migratory result in the multiple machine setting was given by Chadha et al. (Chadha et al. 2009) who gave a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm. The competitive ratio was improved to $O(\frac{1}{\epsilon})$ recently by Anand et al. (Anand et al. 2012).

Energy-efficient scheduling. All the above results assume energy usage is not a concern and the ma-

*Tak-Wah Lam is partially supported by HKU-SPF 201109176197.

chines are always awake. This paper extends the study to the setting with sleep management. The goal is to obtain an algorithm that is efficient in both weighted flow time and energy usage. We model the energy usage as follows. We are given $m \geq 1$ machines. At any time, a machine can be in the *sleep* or *awake* state. A machine can process a job only when it is awake. Let $\sigma \geq 0$ and $\nu \geq 0$ be the static and dynamic power of a machine, respectively. Then, an awake machine consumes energy at a rate of σ if it is idling and at a rate of $\sigma + \nu$ when it is processing a job. For convenience, we let $\mu = \sigma + \nu$. A sleeping machine does not consume energy, but $\lambda > 0$ units of wake-up energy is needed to switch a machine from the sleep state to the awake state (we assume the reverse takes zero energy). Following the literature, we assume that state transition is immediate. Our objective is to minimize the total weighted flow time plus the total energy usage.

When all jobs have unit weight, Lam et al. (Lam et al. 2009) studied the single machine setting and they gave an $O(1)$-competitive algorithm for minimizing total unweighted flow plus energy usage. Their algorithm may put the machine to sleep even when there are unfinished jobs. Sleep management in the multiple machine setting is more complicated. It is natural to balance the workload evenly on all machines for minimizing total flow time, yet it may be more energy-efficient to overload some machine so as to let others sleep. Chan et al. (Chan et al. 2011) showed a multi-machine algorithm POOL that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive for minimizing total unweighted flow plus energy usage. They also showed that without resource augmentation, any algorithm is $\Omega(m)$-competitive, where m is the number of machines.

Our contributions. It has been conjectured that POOL can be generalized to the weighted setting when using HDF to schedule jobs on each machine. Yet the potential analysis of flow time given in (Chan et al. 2011) does not work in the weighted setting. In this paper, we resolve this conjecture in the affirmative by giving a weighted version of POOL which together with HDF gives a new algorithm (to be called WPOOL) that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for minimizing total weighted flow time plus energy usage. The main difficulty lies on the analysis, in which a new potential function is given to relate the weighted flow time plus energy usage of the online algorithm and the optimal schedule. This involves comparing the fractional flow time (which is a relaxed notion of weighted flow time).

Remarks on speed scaling. The above result can be extended to the dynamic speed scaling model where an awake machine can run at any speed between $[0, \infty)$ and each machine can scale its speed individually. The static power of an awake machine remains σ, yet the dynamic power $\nu(s)$ becomes a function of the current speed s. Following the literature (see (Albers 2010) for a survey), we assume $\nu(s) = s^\alpha$ where $\alpha > 1$ is a small constant (typically 3 for CMOS-based devices (Brooks et al. 2000)). When all jobs have the same weight, Chan et al. (Chan et al. 2011) showed that the algorithm POOL can be extended to the speed scaling model. POOL exploits the speed scaling algorithm AJC (Lam et al. 2008), which sets the processor speed based on the number of the unfinished jobs, and is $O(\alpha)$-competitive for unweighted flow plus energy. In this paper, we can extend WPOOL to the speed scaling model. We generalize the speed scaling algorithm BPS (Bansal et al. 2009) with the static power σ taken into consideration (precisely, the processor speed depends on the remaining fraction of the weight of unfinished jobs and the value of σ). This results in an $O(\frac{\alpha^3}{\ln \alpha})$-competitive algorithm for minimizing total weighted flow time plus energy usage. Due to the space limitation, we leave the details of the speed scaling result to the full paper.

Notations. Given a schedule S, let $F(S)$ denote the total weighted flow time of all jobs. In the fixed-speed setting, we assume that the online scheduler is using $(1+\epsilon)$-speed machines, each can process $(1+\epsilon)$ units of work in one unit of time while consuming energy at the same rate as a 1-speed machine used by the optimal offline algorithm (i.e., μ). The energy usage of a machine is the total energy usage when it is awake plus the total wake-up energy used for state transition. Let $E(S)$ denote the total energy usage of all machines. The objective is to minimize $G(S) = F(S) + E(S)$. A job is said to be active at time t if it has been released but not yet completed by time t.

2 The algorithm WPOOL

This section presents the algorithm WPOOL for scheduling jobs with arbitrary weights. We first show some basic facts related to the notions of fractional weight and remaining working cost. Below Off denotes the optimal offline algorithm that minimizes the total weighted flow plus energy for any input.

2.1 Fractional weighted flow and remaining working cost

At any time t, for any job j, let $p(j,t)$ be the unfinished size of j at time t. The *fractional weight* of j at time t, denoted $\tilde{w}(j,t)$, is its weight multiplied by the fraction of its unfinished size. I.e., $\tilde{w}(j,t) = \frac{p(j,t)}{p(j)} \cdot w(j)$. The fractional weighted flow of j is the integral of its fractional weight over time from its arrival until its completion. For any schedule S, denote $\tilde{F}(S)$ as the total fractional weighted flow of all jobs. Obviously, $\tilde{F}(S) \leq F(S)$. We first notice that it is sufficient to design an algorithm that is efficient in terms of total fractional weighted flow. A similar result was proved in (Becchetti et al. 2006) for the setting without energy concern.

Lemma 1. *Let A be any algorithm with sleep management using s-speed machines. For any $\delta > 0$, we can transform A into another algorithm A' using $((1+\delta)s)$-speed machines such that $F(A') \leq (1+\frac{1}{\delta})\tilde{F}(A)$ and $E(A') \leq E(A)$.*

Proof. A' schedules each machine and manages its state identically as A, except that if a job j is already completed by A' but not A, then A' will remain idle when A is processing j. Obviously, $E(A') \leq E(A)$. At any time t, if a job j is not completed by A', the remaining size of j in A is at least $p(j) - \frac{p(j)}{(1+\delta)s} \cdot s = p(j)\frac{\delta}{1+\delta}$. Hence the fractional weight of j in A is at least $\frac{\delta}{1+\delta}w(j)$. It implies that the total weight of active jobs in A' is at most $(1+\frac{1}{\delta})$ times the total fractional weight of active jobs in A, and hence $F(A') \leq (1+\frac{1}{\delta})\tilde{F}(A)$. □

Hence, we first design an algorithm A with bounded total fractional weighted flow plus energy usage. Then by increasing the speed of A further by

a factor of $(1+\delta)$, we obtain an algorithm that is competitive with respect to the total weighted flow plus energy usage.

The density of a job j is the ratio of its weight to size, i.e., $w(j)/p(j)$. It is known that the algorithm HDF (Highest Density First) minimizes the total fractional weighted flow on a single machine when there is no energy concern (Becchetti et al. 2006). In fact, our algorithm WPOOL schedules jobs by HDF for each machine whenever it is awake. WPOOL dispatches a job to a machine once the job arrives. To decide which machine a job is dispatched to, WPOOL uses the following definition.

Definition 2. *At any time t, the* remaining working cost *(rwc) for a machine i, denoted $rwc_i(t)$, is the total fractional weighted flow plus energy to be incurred after time t by machine i while it is processing a job, assuming no more jobs arrive after time t and assuming it processes jobs by HDF whenever it is working.*

We can compute $rwc_i(t)$ as follows. Let T be the intervals of time after t during which machine i is processing a job. Let $\tilde{w}_i(x)$ be the total fractional weight of all jobs dispatched to machine i that are not yet completed by time x. Then $rwc_i(t) = \int_T \tilde{w}_i(x)dx + |T|\mu$, where $|T|$ is the total length of the time intervals in T. Note that $rwc_i(t)$ depends only on the time when machine i is working, but not the time when it is idle or sleeping. Furthermore, it assumes that the machine processes jobs by HDF whenever it is working.

The following lemma gives a more useful formula for calculating $rwc_i(t)$. The inverse density of a job j, denoted $q(j)$, is the inverse of its density, i.e., $q(j) = p(j)/w(j)$. At any time t, for any real $q \geq 0$, let $n_i(q,t)$ be the number of active jobs in machine i with inverse density at least q, and let $\tilde{w}_i(q,t)$ be the total fractional weight of those jobs. Let $\tilde{w}_i(t) = \tilde{w}_i(0,t)$, i.e., the total fractional weight of all active jobs in machine i at time t.

Lemma 3. *Suppose that machine i uses HDF to schedule the jobs whenever awake. Then, at any time t, $rwc_i(t) = \int_0^\infty \int_0^{\tilde{w}_i(q,t)} \frac{z+\mu}{s}\,\mathrm{d}z\,\mathrm{d}q$, where s is the speed of machine i. Moreover, if a job j of inverse density x arrives at time t and is dispatched to machine i immediately, the increase in rwc due to j equals $\int_0^x \int_{\tilde{w}_i(q,t)}^{\tilde{w}_i(q,t)+w(j)} \frac{z+\mu}{s}\,\mathrm{d}z\,\mathrm{d}q$ (note that $\tilde{w}_i(q,t)$ refers to the total fractional weight just before j arrives).*

Proof. Let $j_1, j_2, \ldots, j_n$ be the active jobs in machine i at time t, ordered in non-decreasing order of density. I.e., $\frac{w(j_1)}{p(j_1)} \leq \frac{w(j_2)}{p(j_2)} \leq \cdots \leq \frac{w(j_n)}{p(j_n)}$. If no job arrives after time t, then HDF executes the active jobs in the order of $j_n, j_{n-1}, \ldots, j_1$. Since $rwc_i(t)$ only includes the cost incurred when machine i is working, we assume w.l.o.g. that machine i does not sleep after time t.

Consider the maximal period $[t_k, t'_k]$ of time that HDF is processing the job j_k. Let $\tilde{G}(t_k, y)$ be the fractional weighted flow plus energy incurred during $[t_k, y]$ for some time $y \in [t_k, t'_k]$. Then, $\frac{d\tilde{G}(t_k,y)}{dy} = \tilde{w}_i(y)+\mu$ and $\tilde{w}_i(y)$ is decreasing (w.r.t. y) at a rate of $s\frac{w(j_k)}{p(j_k)} = \frac{s}{q(j_k)}$ (i.e., $\frac{d\tilde{w}_i(y)}{dy} = \frac{-s}{q(j_k)}$). Thus, $\frac{d\tilde{G}(t_k,y)}{d\tilde{w}_i(y)} = -(\tilde{w}_i(y)+\mu)\frac{q(j_k)}{s}$. At time t_k, $\tilde{w}_i(t_k) = \tilde{w}(j_1,t) + \cdots + \tilde{w}(j_k,t)$; also, at time t'_k, $\tilde{w}_i(t'_k) = \tilde{w}(j_1,t) + \cdots + \tilde{w}(j_{k-1},t)$. Thus,

$$\begin{aligned}\tilde{G}(t_k,t'_k) &= \int_{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_k,t)}^{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_{k-1},t)} -(z+\mu)\cdot\frac{q(j_k)}{s}\,\mathrm{d}z \\ &= q(j_k)\int_{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_{k-1},t)}^{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_k,t)} \frac{z+\mu}{s}\,\mathrm{d}z\ .\end{aligned}$$

Notice that $rwc_i(t) = \sum_{k=1}^n \tilde{G}(t_k,t'_k) = \sum_{k=1}^n q(j_k)\int_{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_{k-1},t)}^{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_k,t)} \frac{z+\mu}{s}\,\mathrm{d}z$. We expand $q(j_k)$ for $1 \leq k \leq n-1$ using $q(j_k) = q(j_n)+(q(j_{n-1})-q(j_n)) + \cdots + (q(j_k)-q(j_{k+1}))$. Collecting those terms with coefficient $(q(j_k)-q(j_{k+1}))$, the above summation becomes $q(j_n)\int_0^{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_n,t)} \frac{z+\mu}{s}\,\mathrm{d}z + \sum_{k=1}^{n-1}(q(j_k)-q(j_{k+1}))\cdot\int_0^{\tilde{w}(j_1,t)+\cdots+\tilde{w}(j_k,t)} \frac{z+\mu}{s}\,\mathrm{d}z = \int_0^\infty\int_0^{\tilde{w}_i(q,t)} \frac{z+\mu}{s}\,\mathrm{d}z\,\mathrm{d}q$.

If a job j with $q(j) = x$ arrives at time t and is dispatched to machine i, then $\tilde{w}_i(q,t)$ increases by $w(j)$ for $q \in [0,x]$. Thus, the increase in rwc_i due to j equals $\int_0^x \int_{\tilde{w}_i(q,t)}^{\tilde{w}_i(q,t)+w(j)} \frac{z+\mu}{s}\,\mathrm{d}z\,\mathrm{d}q$. □

2.2 Algorithm definition

WPOOL attempts to maintain a small pool P of *dispatchable machines*; P contains one sleeping machine initially and is always non-empty. At any time, machines in P are either all asleep or all awake, and P is said to be asleep or awake, respectively. Machines not in P are always asleep and do not have active jobs. WPOOL would gradually include more machines into P as jobs arrive, and they are immediately put into the same state as P.

WPOOL maintains three real-value counters $\mathrm{X_{AF}}$, $\mathrm{X_{IE}}$ and $\mathrm{X_{WC}}$ to keep track of the recent increase to the accumulated flow (when P is asleep), idling energy, and working cost, respectively. Initially, all counters equal 0. As detailed below, they each keep increasing until they trigger certain events, then they will be reset.

- When P is asleep, $\mathrm{X_{AF}}$ increases continuously at the rate of the total fractional weight of active jobs. Once $\mathrm{X_{AF}}$ reaches λ, we wake up all machines of P and reset $\mathrm{X_{AF}}$ to zero.
- When P is awake, $\mathrm{X_{IE}}$ increases continuously at the rate of σ times the number of idle machines. Once $\mathrm{X_{IE}}$ reaches λ, if P has two or more idle machines, we remove one idle machine from P, put it to sleep and reset $\mathrm{X_{IE}}$ to zero. See the algorithm below for the details of some boundary cases.
- $\mathrm{X_{WC}}$ increases only when a job arrives. Intuitively, whenever $\mathrm{X_{WC}}$ reaches a value at least λ, we try to include one machine into P and decrease $\mathrm{X_{WC}}$ by λ. Specifically, when a job j arrives, WPOOL first assumes that j is dispatched to a machine with no active jobs and calculates the increase in *rwc*. Denote this amount of increase as $null\_Inc\_rwc(j)$. If $\mathrm{X_{WC}} + null\_Inc\_rwc(j) \geq \lambda$ and $|P| < m$, WPOOL will add one machine into P and dispatch j to this machine (even if P already has idle or sleeping machines), and $\mathrm{X_{WC}}$ is set to $\mathrm{X_{WC}} + null\_Inc\_rwc(j) - \lambda$. Otherwise, WPOOL dispatches j to a machine i in P that minimizes the increase in *rwc*; below we denote

machine i as $\ell(j,P)$, and the amount of increase in rwc as $min\_Inc\_rwc(j,P)$. $\mathrm{X_{WC}}$ increases by $min\_Inc\_rwc(j,P)$ (which is at least $null\_Inc\_rwc(j)$). At the end, if $\mathrm{X_{WC}}$ is at least λ, we repeatedly try to add a machine into P and decrease $\mathrm{X_{WC}}$ by λ.

Below is a complete description of the algorithm WPOOL.

Increase $\mathrm{X_{AF}}$ and wake up P: At any time, if P is asleep,

- $\mathrm{X_{AF}}$ increases at the rate of the total fractional weight of active jobs;
- **If** $(\mathrm{X_{AF}} = \lambda)$, **then** wake up all machines in P; reset $\mathrm{X_{AF}} = 0$.

Increase $\mathrm{X_{IE}}$, remove & sleep an idle machine of P: At any time, if P is awake,

- **If** $(\mathrm{X_{IE}} < \lambda)$, **then** $\mathrm{X_{IE}}$ increases at the rate of σ times the number of idle machines.
- **If** $(\mathrm{X_{IE}} = \lambda)$, **then**
  ◇ if P has ≥ 2 idle machines, remove and sleep one idle machine from P; reset $\mathrm{X_{IE}} = 0$;
  ◇ if P has only one idle machine and $|P| = 1$, put P to sleep; reset $\mathrm{X_{WC}} = \mathrm{X_{IE}} = 0$;
  ◇ otherwise, $\mathrm{X_{IE}}$ remains equal to λ.

Dispatch a new job, increase $\mathrm{X_{WC}}$ and expand P: When a job j arrives,

- **If** $((|P| < m)$ & $(\mathrm{X_{WC}} + null\_Inc\_rwc(j) \geq \lambda))$,
  then add a machine to P; dispatch j to this machine; $\mathrm{X_{WC}} = \mathrm{X_{WC}} + null\_Inc\_rwc(j) - \lambda$;
  else dispatch j to $\ell(j,P)$; $\mathrm{X_{WC}} = \mathrm{X_{WC}} + min\_Inc\_rwc(j,P)$;
- **While** $((\mathrm{X_{WC}} \geq \lambda)$ & $(|P| < m))$ **do**
  { add a machine to P; $\mathrm{X_{WC}} = \mathrm{X_{WC}} - \lambda$ }
- **If** $\mathrm{X_{WC}} > \lambda$, **then** $\mathrm{X_{WC}} = \lambda$.

Job scheduling in each machine of P: When awake, use HDF policy.

Property 4. *Every time after* WPOOL *executes the job dispatching procedure, it maintains the invariant that if $|P| < m$, there exists a machine with $rwc < \lambda$.*

Proof. Suppose, for the sake of contradiction, that there exists a time t, such that immediately after some job is dispatched at time t, $|P| < m$, and all machines in $|P|$ has $rwc \geq \lambda$. Note that $\lambda > 0$. Let $t_0 \leq t$ be the latest time not later than t, such that at t_0 all machines in P have $rwc > 0$, yet immediately before t_0 at least one machine in P has $rwc = 0$. Hence, from time t_0 to time t, all machines in P have $rwc > 0$ and no machine is removed from P (since WPOOL removes a machine from P only when it is idle, which implies it has $rwc = 0$). Now, consider all zero-rwc machines in P immediately before t_0. By the definition of WPOOL, at most one such machine can be removed from P at time t_0, and it cannot be removed if it is the only one idle machine in P. Hence, there exists a zero-rwc machine i_0 in P immediately before t_0, such that i_0 is still in P immediately after t_0 but with $rwc > 0$ (due to some jobs dispatched to it). Assume that there are x new machines added to P from t_0 to t (including the ones added at time t). As mentioned above, no machine will be removed from P from t_0 to t. Hence, i_0 and these x machines (totally $x+1$ machines) will remain in P at least until time t, and each of them has $rwc \geq \lambda$ at time t. So, each of them has an increase of rwc at least λ since t_0. By the definition of WPOOL, at least $x+1$ machines are added to P from t_0 to t, which contradicts the assumption that only x machines are added. □

Our main result is as follows, which will be proved in the next section.

Theorem 5. *When* WPOOL *is given $(1+\epsilon)$-speed machines, the total fractional weighted flow plus energy usage of* WPOOL *is at most $O(1+\frac{1}{\epsilon})$ times the total weighted flow plus energy usage of* Off.

Then, together with Lemma 1, we have the following corollary.

Corollary 6. *There exists an algorithm* WPOOL$'$ *that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive for total weighted flow plus energy usage.*

Proof. By Lemma 1 and Theorem 5, we can obtain an algorithm WPOOL$'$ that is $(1+\epsilon')(1+\delta)$-speed $O((1+\frac{1}{\epsilon'})(1+\frac{1}{\delta}))$-competitive. By putting $\epsilon' = \delta = \frac{\epsilon}{3}$, the result follows. □

3 Analysis of WPOOL

This section analyzes WPOOL and hence proves Theorem 5. Consider a certain input sequence of jobs. Recall that Off is the optimal offline schedule. W.L.O.G., we can assume that Off dispatches a job to a machine once the job arrives. A machine i in Off is said to be *procrastinating* at time t if some job dispatched to machine i is unfinished by time t yet machine i is asleep at time t. Instead of analyzing WPOOL with respect to Off directly, we analyze WPOOL with another offline schedule Opt with the following property.

Lemma 7. *We can transform* Off *into another schedule* Opt *such that* Opt *has at most one procrastinating machine at any time and* Opt *schedules a machine by HDF whenever the machine is working. Furthermore, $\tilde{F}(\mathrm{Opt}) \leq F(\mathrm{Off})$ and $E(\mathrm{Opt}) \leq 3E(\mathrm{Off})$.*

Proof. Chan et al. (Chan et al. 2011) showed that we can transform Off into another schedule Opt$'$ such that Opt$'$ has at most one procrastinating machine at any time, by changing some machines from the sleep state to awake state when needed. The energy usage of Opt$'$ is at most 3 times that of Off, i.e. $E(\mathrm{Opt}') \leq 3E(\mathrm{Off})$. Furthermore, the transformation guarantees that the completion time of every job j in Opt$'$ is no later than its completion time in Off, so the total weighted flow of Opt$'$ is at most that of Off, i.e. $F(\mathrm{Opt}') \leq F(\mathrm{Off})$. Then, we further transform Opt$'$ into the targeted schedule Opt as follows: for each job, Opt dispatches it to the same machine as Opt$'$ does. For each machine i, at any time t, the state (sleep, idle or working) and speed of i in Opt are the same as the state and speed of i in Opt$'$. However, for each machine i in Opt, i schedules jobs dispatched to it by HDF whenever working. Obviously, Opt is a valid schedule, and $E(\mathrm{Opt}) = E(\mathrm{Opt}')$. It is well known that, on single machine, HDF minimizes the fractional weighted flow when speed function is fixed, hence, $\tilde{F}(\mathrm{Opt}) \leq \tilde{F}(\mathrm{Opt}') \leq F(\mathrm{Opt}')$. □

In the following, we show that the total fractional weighted flow plus energy usage of WPOOL is at most $O(1+\frac{1}{\epsilon})$ times that of Opt. Let $\tilde{F}$ and E be the total fractional weighted flow and energy usage of WPOOL, respectively. We divide $\tilde{F}$ into two parts: the *working* flow $\tilde{F}_{\mathrm{W}}$ and the *sleeping* flow $\tilde{F}_{\mathrm{S}}$, which refer to

the total fractional weighted flow incurred by the machines when they are working and sleeping, respectively. Note that $\tilde{F} = \tilde{F}_{\text{W}} + \tilde{F}_{\text{S}}$. We also divide E into three parts: E_{I} is the idling energy (static energy usage during the idle state), E_{W} the working energy, and E_{U} the wake-up energy. Then $E = E_{\text{I}} + E_{\text{W}} + E_{\text{U}}$. Let $\tilde{G} = \tilde{F} + E$ be the *cost* of WPOOL. We use the same notations with an extra asterisk to denote the corresponding quantity in Opt. For example, $\tilde{F}^*$ is the total fractional weighted flow of Opt, and $\tilde{G}^*$ is the cost of Opt.

3.1 Sleeping flow and energy usage

It is relatively easy to upper bound the sleeping flow $\tilde{F}_{\text{S}}$ and the energy usage in terms of $\tilde{G}^*$ and $\tilde{F}_{\text{W}}$. They are summarized by the following lemma. The proof is similar to that in (Chan et al. 2011).

Lemma 8. (i) $\tilde{F}_{\text{S}} \le \tilde{G}^*$; (ii) $E_{\text{W}} \le E^*_{\text{W}}$; (iii) $E_{\text{U}} \le \tilde{F}_{\text{W}} + E_{\text{W}} + \tilde{G}^*$; (iv) $E_{\text{I}} \le E_{\text{U}} + E_{\text{W}}$.

Proof. We divide the timeline into intervals called P-intervals, each of which consists of a maximal asleep period of P, followed by a maximal awake period of P. Within a P-interval I, we use the corresponding notation with a suffix (I) to denote the flow or energy within this interval. Note that within each P-interval, the sleeping fractional weighted flow accumulated during asleep period is λ, and the idling energy accumulated during awake period is at least λ.

To make the discussion easy, we charge the wake-up energy of Opt to the time when it sleeps a machine. We will first show that within each P-interval I, the cost of Opt is at least λ, i.e. $\tilde{G}^*(I) \ge \lambda$. The argument is as follows. If Opt sleeps a machine during I, then it is obvious that $\tilde{G}^*(I) \ge \lambda$. If Opt never sleeps a machine during I, there are two cases: Case (1), if Opt has all machines asleep at the beginning of I and it does not wake up a machine during the asleep period of I, then Opt would let the jobs arrived during the asleep period of I to accumulate a fractional weighted flow at least λ, thus $\tilde{G}^*(I) \ge \lambda$. Case (2), if OPT has at least one machine awake at the beginning of I or it wakes up a machine during the asleep period of I, then the static energy incurred by this machine during I is at least the idling energy incurred by WPOOL during I when P has only one idle machine, which is at least λ. We are now ready to prove items (i) to (iv)

(i) At the beginning of a P-interval I, all machines in P are asleep and there are no active jobs. POOL wakes up machines in P as soon as the accumulated sleeping fractional flow increases to λ, and later, no sleeping fractional flow is accumulated till the end of I. Hence, $\tilde{F}_{\text{S}}(I) = \lambda \le \tilde{G}^*(I)$. Summing over all P-intervals, we obtain $\tilde{F}_{\text{S}} \le \tilde{G}^*$.

(ii) Note that WPOOL and Opt process the same amount of work and WPOOL is using machines with speed $s \ge 1$. Hence, the total amount of time that WPOOL is working is at most that of Opt. It implies that the working energy of WPOOL is at most that of Opt.

(iii) Note that at the beginning of a P-interval I, there is only one machine in P and this machine is asleep. Suppose that there are totally x machine additions within I. Hence, the wake-up energy of POOL within I is $E_{\text{U}}(I) = (x+1)\lambda$. Note that a machine is added into P only when the accumulated increase of rwc (i.e. the counter X_{WC}) is at least λ. When I ends, all jobs are completed and all the accumulated increase in rwc have become part of $\tilde{F}_{\text{W}}(I) + E_{\text{W}}(I)$. Hence, $x\lambda \le \tilde{F}_{\text{W}}(I) + E_{\text{W}}(I)$. It follows that $E_{\text{U}}(I) = (x+1)\lambda = x\lambda + \lambda \le \tilde{F}_{\text{W}}(I) + E_{\text{W}}(I) + \tilde{G}^*(I)$. Summing over all P-interval, we obtain $E_{\text{U}} \le \tilde{F}_{\text{W}} + E_{\text{W}} + \tilde{G}^*$.

(iv) E_{I} can be divided into two types. The first type is incurred when the counter $\text{X}_{\text{IE}} < \lambda$ and the other type is incurred when the counter $\text{X}_{\text{IE}} = \lambda$. For the first type E_{I}, it increases at the same rate as X_{IE}. At any time t that X_{IE} reaches λ, WPOOL would sleeps a machine at time t or later. Hence, the total of the first type E_{I} equals λ times the total number of times WPOOL sleeps a machine, which is exactly E_{U}.

For the second type E_{I}, it accumulates only when P has only one idle machine and $|P| > 1$. So, there exists another working machine in P when this type of E_{I} is accumulated. Note that a working machine consumes energy at a rate more than an idle machine. Hence, this type of E_{I} is at most E_{W}. □

3.2 Potential analysis of $\tilde{F}_{\text{W}}$

It remains to analyze the working flow $\tilde{F}_{\text{W}}$, which will be bounded by the cost $\tilde{G}^*$ of Opt and the sleeping flow $\tilde{F}_{\text{S}}$ of WPOOL. Precisely, we will show the following lemma.

Lemma 9. $\tilde{F}_{\text{W}} \le (12 + \frac{15}{\epsilon})\tilde{G}^* + \frac{1}{\epsilon}\tilde{F}_{\text{S}}$.

Notice that Lemmas 8 and 9 together imply that $\tilde{G} \le (43 + \frac{48}{\epsilon})\tilde{G}^*$. By Lemma 7, $\tilde{G}^* \le 3G(\text{Off})$. Hence, Theorem 5 follows.

The rest of the section proves Lemma 9 using a potential function that allows different match-ups between machines of WPOOL and Opt. Let $\tilde{F}_{\text{W}}(t)$ and $\tilde{F}_{\text{S}}(t)$ denote respectively the working flow and sleeping flow incurred by WPOOL up to time t. Let $\tilde{G}^*(t)$ be the cost incurred by Opt up to time t. Assume that machines are labeled with integers from 1 to m. At any time, we match each machine in WPOOL with a certain machine in Opt. Below we denote $x(i)$ as the machine in Opt currently matched with machine i in WPOOL. This matching is only for the purpose of analysis and not known to the algorithms. Initially $x(i) = i$ for all i. To show Lemma 9, we define a potential function $\Phi(t)$ that reflects WPOOL's remaining working cost discounted in view of Opt's workload in the corresponding machines. Technically, we want $\Phi(t)$ to satisfy the following conditions: **(i)** Boundary condition: $\Phi = 0$ before any job is released and after all jobs are completed. **(ii)** Job completion and state transition condition: Φ does not increase when a job is completed or a machine changes its state in WPOOL or Opt. **(iii)** Job arrival condition: When a job j arrives, we re-match the machines (for analysis sake) before j is dispatched. Φ may then increase, yet the total increase due to all job arrivals is upper bounded by $O(1 + \frac{1}{\epsilon})\tilde{G}^*$ (precisely, $(11 + \frac{14}{\epsilon}) \cdot \tilde{G}^*$). **(iv)** Running condition: At any other time t, $\frac{\text{d}\tilde{F}_{\text{W}}(t)}{\text{d}t} + \frac{\text{d}\Phi(t)}{\text{d}t} \le (1 + \frac{1}{\epsilon}) \cdot \frac{\text{d}\tilde{G}^*(t)}{\text{d}t} + \frac{1}{\epsilon} \cdot \frac{\text{d}\tilde{F}_{\text{S}}(t)}{\text{d}t}$. By integrating the above conditions over time, $\tilde{F}_{\text{W}} \le (11 + \frac{14}{\epsilon} + 1 + \frac{1}{\epsilon})\tilde{G}^* + \frac{1}{\epsilon}\tilde{F}_{\text{S}} = (12 + \frac{15}{\epsilon})\tilde{G}^* + \frac{1}{\epsilon}\tilde{F}_{\text{S}}$, and Lemma 9 follows.

Potential function Φ. For any machine i of WPOOL, for any $q \ge 0$, recall that $\tilde{w}_i(q,t)$ denotes the total fractional weight of active jobs dispatched

to machine i with inverse density at least q at time t. Define $\tilde{w}^*_i(q,t)$ similarly for Opt. We will drop the parameter t when t refers clearly to the current time. Let $(\cdot)_+ = \max(\cdot, 0)$. Below is the definition of the potential function $\Phi(t)$.

$$\Phi(t) = \sum_{i=1}^{m} \Phi_i(t), \quad \text{where}$$

$$\Phi_i(t) = \frac{1}{\epsilon} \int_0^\infty \int_0^{\tilde{w}_i(q)} (z - \tilde{w}^*_{x(i)}(q) + \mu)_+ \, \mathrm{d}z \, \mathrm{d}q \ .$$

Machine re-matching. Recall that machine re-matching is for the sake of analysis and not part of WPOOL or Opt. hWe allow it to operate based on a rather intricate view of the machine states. Details are as follows. At any time t, we define three different views of machine states in WPOOL and Opt, namely, $H_-(t)$, $H_+(t)$ and $H_R(t)$. The first two are from an operational viewpoint, and the last one is for the purpose of re-matching and analysis only. W.L.O.G., we assume that Opt, at any time, first performs all the required wake-up operations before moving to other operations.

- $H_-(t)$ refers to states of the machines in WPOOL or Opt just before time t;
- $H_+(t)$ refers to states of the machines WPOOL or Opt immediately after time t; and
- $H_R(t)$ is something in between $H_-(t)$ and $H_+(t)$, it refers to the states after WPOOL and Opt have performed only some operations at time t: WPOOL has removed all idle machines from P that should sleep at time t, but WPOOL has not yet handled any job arriving at t (and has not included more machine into P); and Opt has waken up all machines that are scheduled to wake up at time t, but Opt has not yet put any machine to sleep and has not dispatched any job arriving at t.

At any time t that a job j arrives, we re-match the machines of WPOOL and Opt with respect to $H_R(t)$. Re-matching actually means computing a new matching function $x(i)$ as follows.

> Let $x(i)$ be the current matching function. Note that $x(1), \ldots, x(m)$ is a permutation of $1, 2, \ldots, m$. With respect to $H_R(t)$, we figure out whether a machine is in P or not in accordance with WPOOL, as well as whether a machine is awake or sleep in accordance with Opt. Then, as long as we find machines i and i' satisfying the following conditions, we swap $x(i)$ and $x(i')$.
>
> - WPOOL has i not in P and Opt has $x(i)$ awake, or procrastinating, or to be procrastinating (sleeping w.r.t. $H_R(t)$, but j to be dispatched to $x(i)$); and
> - WPOOL has i' in P and Opt has $x(i')$ sleeping and not procrastinating.

Note that Φ_i, $\Phi_{i'}$ and hence Φ may change after a swapping. Interestingly, we can show that this change is always non-increasing.

Lemma 10. *After some $x(i)$ and $x(i')$ are swapped, Φ_i and $\Phi_{i'}$ do not increase.*

Proof. Machine i is not in the pool P and has no active jobs, so $\Phi_i = 0$ before and after the swapping. Next, we consider $\Phi_{i'}$. Before swapping, machine $x(i')$ was sleeping and not procrastinating in Opt, thus $\tilde{w}^*_{x(i')}(q) = 0$ for all q. Since $\Phi_{i'}$ changes from $\frac{1}{\epsilon}\int_0^\infty \int_0^{\tilde{w}_{i'}(q)} (z - \tilde{w}^*_{x(i')}(q) + \mu)_+ \, \mathrm{d}z \, \mathrm{d}q$ to $\frac{1}{\epsilon}\int_0^\infty \int_0^{\tilde{w}_{i'}(q)} (z - \tilde{w}^*_{x(i)}(q) + \mu)_+ \, \mathrm{d}z \, \mathrm{d}q$, it can only decrease after the swapping. □

We are ready to consider the various conditions imposed on Φ. The boundary condition trivially holds. The job completion and state transition condition also hold as follows. When a job is completed by WPOOL or Opt, $\tilde{w}_i(q)$ and $\tilde{w}^*_{x(i)}(q)$ are unchanged for all i and all q, so Φ is unchanged. Furthermore, a state transition does not affect Φ.

The running condition depends solely on the job scheduling policy (HDF) and can be analyzed independently for each matched pair of machines using similar techniques for the single-machine analysis (Bansal et al. 2009); details will be given in Section 3.4.

The arrival condition depends on both sleep management and job dispatching policies. In the next subsection, we show that when a job j arrives, after machine re-matching and job dispatching, the increase of Φ due to j can be bounded in terms of some non-overlapping cost of Opt.

3.3 Arrival Condition

When a job arrives, machines may be re-matched, and then the job gets dispatched by WPOOL and Opt. Φ would possibly increase. This section is devoted to upper bounding such increase.

Lemma 11. *The sum over all jobs of the increase in Φ due to a job arrival (after machine re-matching and job dispatching) is at most $(11 + \frac{14}{\epsilon}) \cdot \tilde{G}^*$.*

By Lemma 10, machine re-matching can not increase Φ, hence, in the following, our analysis bases on the assumption that machine re-matching is already done.

Recall that $H_-(t)$, $H_+(t)$ and $H_R(t)$ are different views of machine states at time t. We define $h_-(t)$, $h_+(t)$ and $h_R(t)$ to be the number of machines in P with respect to $H_-(t)$, $H_+(t)$ and $H_R(t)$, respectively. Define $h^*_-(t)$, $h^*_+(t)$ and $h^*_R(t)$ similarly for the number of awake machines in Opt.

Type-0, Type-1 and Type-2 jobs. To prove Lemma 11, we divide the jobs into three types and analyze them separately. Define Type-0 jobs to be the jobs which WPOOL dispatches to a zero-rwc machine (i.e. machine with no active jobs). For any other job j, if at its arrival time t, $h_R(t) \le h^*_R(t)$ and $h_R(t) < m$, j is Type-1; otherwise, j is Type-2.

Roughly speaking, Type-2 jobs arrive when WPOOL is using more machines than Opt (or using all the m machines). It is relatively easy to show that for each Type-2 job j, Φ has limited increase (see Lemma 12).

For any Type-0 or Type-1 job j, we first observe that once j is dispatched, the increase in Φ can be upper bounded in terms of WPOOL's increase in rwc due to j. Let $\Delta rwc(j)$ be the increase in rwc to WPOOL due to j. Φ only increases due to the increase in Φ_i, where i is the machine to which WPOOL assigns j.

This increase, by definition of Φ, is at most

$$\begin{aligned}
&\frac{1}{\epsilon}\cdot\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z-\tilde{w}^*_{x(i)}(q)+\mu)_+\,\mathrm{d}z\,\mathrm{d}q\\
\le\ & (1+\tfrac{1}{\epsilon})\cdot\textstyle\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}\frac{z+\mu}{1+\epsilon}\,\mathrm{d}z\,\mathrm{d}q\\
=\ & (1+\tfrac{1}{\epsilon})\cdot\Delta rwc(j)
\end{aligned}$$

where the last equality follows from Lemma 3. In other words, consider all Type-0 and Type-1 jobs, the total increase in Φ is at most $(1+\frac{1}{\epsilon})$ times the total increase in *rwc* of WPOOL. Lemmas 13 and 14 below give upper bounds of the increase in *rwc* to WPOOL due to Type-0 and Type-1 jobs, respectively. Lemma 13 is relatively simple as we can show that for each Type-0 job, WPOOL's increase in *rwc* cannot exceed that of Opt. For Type-1 jobs, WPOOL might be using very few machines and WPOOL's increase in *rwc* can be way larger than Opt's. In Lemma 14, we analyze Type-1 jobs interval by interval (instead of job by job) and show that WPOOL's increase in *rwc* is bounded by the static and wakeup energy of Opt. The proofs of Lemmas 12 and 13 will be shown in Appendix A.

Lemma 12. *The total increase in Φ due to Type-2 jobs is at most $\frac{3}{\epsilon}\cdot\tilde{G}^*$.*

Lemma 13. WPOOL*'s total increase in rwc due to Type-0 jobs is at most $\tilde{G}^*$.*

Lemma 14. WPOOL*'s total increase in rwc due to Type-1 jobs is at most $10\tilde{G}^*$.*

To analyze Type-1 jobs, we define *lazy intervals* below, which would include all arrival times of Type-1 jobs. Roughly speaking, inside a lazy interval, WPOOL is lazy in the sense that WPOOL is using no more machines than Opt.

Lazy intervals. A lazy interval $\ell=[t_1,t_2]$, where $t_1\le t_2$, satisfies the following property. Consider any view of machine states $H_\gamma(t)$ where $t\in[t_1,t_2]$ and $\gamma\in\{+,-,R\}$. If $H_\gamma(t)\equiv H_{\_}(t_1)$ or $H_{+}(t_2)$, then with respect to $H_\gamma(t)$, the number of machines in P is greater than the number of awake machines of Opt (i.e., $h_\gamma(t)>h^*_\gamma(t)$); for any other view $H_\gamma(t)$, the number of machines in P is at most the number of awake machines of Opt (i.e., $h_\gamma(t)\le h^*_\gamma(t)$).

Before proving Lemma 14, we observe the following properties of WPOOL.

Property 15. *When a job j arrives, if* WPOOL *dispatches it to a machine with non-zero rwc, then at most two machines are added to P.*

Proof. Suppose a job j arrives at time t. Consider the moment just before WPOOL dispatches j. If $|P|=h_{_R}(t)\ge m-2$, at most two machines can be added to P and the lemma holds. Now assume that $|P|<m-2$. First note that $null\_Inc\_rwc(j)=\frac{1}{1+\epsilon}\int_0^{q(j)}\left(\frac{1}{2}(w(j))^2+\mu w(j)\right)dq$. Since j is dispatched to a machine with non-zero *rwc*, by the definition of WPOOL, $\mathrm{X}_{\mathrm{WC}}+null\_Inc\_rwc(j)<\lambda$. Just before j is dispatched, let i be the machine in WPOOL with the smallest *rwc*, and denoted the *rwc* of it by $rwc(i)$. Hence $rwc(i)=\frac{1}{1+\epsilon}\int_0^{\infty}\int_0^{\tilde{w}_i(q)}(z+\mu)dzdq=\frac{1}{1+\epsilon}\int_0^{\infty}\left(\frac{1}{2}(\tilde{w}_i(q))^2+\mu\tilde{w}_i(q)\right)dq$. Since $|P|<m$, by Property 4, $rwc(i)<\lambda$.

Let $Inc\_rwc(j,i,P)$ be the increase of *rwc* if j is dispatched to machine i. In the following, we will show that $\mathrm{X}_{\mathrm{WC}}+Inc\_rwc(j,i,P)<3\lambda$. Hence $\mathrm{X}_{\mathrm{WC}}+min\_Inc\_rwc(j,P)<3\lambda$ and at most two machines are added to P.

$$\begin{aligned}
&Inc_rwc(j,i,P)\\
=\ & \frac{1}{1+\epsilon}\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z+\mu)dzdq\\
=\ & \frac{1}{1+\epsilon}\int_0^{q(j)}\left(\tilde{w}_i(q)w(j)+\frac{1}{2}(w(j))^2+\mu w(j)\right)dq\\
=\ & \frac{1}{1+\epsilon}\int_0^{q(j)}\tilde{w}_i(q)w(j)dq+null_Inc_rwc(j)\\
\le\ & \frac{1}{1+\epsilon}\int_0^{q(j)}\frac{1}{2}(\tilde{w}_i(q))^2dq\\
&\quad+\frac{1}{1+\epsilon}\int_0^{q(j)}\frac{1}{2}(w(j))^2dq+null_Inc_rwc(j)\\
\le\ & rwc(i)+2\cdot null_Inc_rwc(j).
\end{aligned}$$

Therefore, $\mathrm{X}_{\mathrm{WC}}+Inc\_rwc(j,i,P)\le rwc(i)+2(\mathrm{X}_{\mathrm{WC}}+null\_Inc\_rwc(j))<3\lambda$. □

Property 16. *Every Type-1 job must arrive within a lazy interval.*

Proof. Suppose a Type-1 job j arrives at time t. By definition, $h_{_R}(t)\le h^*_{_R}(t)$. We find the largest $t_1\le t$ and the smallest $t_2\ge t$ satisfying the boundary conditions of a lazy interval (the boundaries t_1 and t_2 always exist because at time 0 and at the time t_e when both schedules of WPOOL and Opt end, $|P|=1$ and the number of awake machine in Opt is 0. Thus, $h_{\_}(0)=1>0=h^*_{\_}(0)$, and $h_{+}(t_e)=1>0=h^*_{+}(t_e)$). Consider any time $t'\in[t_1,t_2]$. If $h_{\_}(t')\le h^*_{\_}(t')$ or $h_{+}(t')\le h^*_{+}(t')$ then $h_{_R}(t')\le h^*_{_R}(t')$. It follows that with respect to any view $H_\gamma(t')$, except the two boundary views, we have $h_\gamma(t')\le h^*_\gamma(t')$. Therefore, $[t_1,t_2]$ is a lazy interval covering time t. □

Within a lazy interval $\ell=[t_1,t_2]$, let O_ℓ be the number of times WPOOL has removed a machine from P, let I_ℓ be the number of times WPOOL has added a machine into P except those added at t_2, and let W^*_ℓ be the number of times Opt has waken up a machine. The above definitions imply another useful property of a lazy interval.

Property 17. *For any lazy interval ℓ, $I_\ell+1\le W^*_\ell+O_\ell$.*

Proof. By definition of a lazy interval, $h_{\_}(t_1)>h^*_{\_}(t_1)$ and $h_{_R}(t_2)\le h^*_{_R}(t_2)$. By definition of O_ℓ, I_ℓ and W^*_ℓ, $h_{\_}(t_1)-O_\ell+I_\ell\le h_{_R}(t_2)$, and $h^*_{_R}(t_2)\le h^*_{\_}(t_1)+W^*_\ell$. Therefore, $h_{\_}(t_1)-O_\ell+I_\ell\le h^*_{_R}(t_2)\le h^*_{\_}(t_1)+W^*_\ell$. Recall that $h_{\_}(t_1)>h^*_{\_}(t_1)$. Therefore, $-O_\ell+I_\ell<W^*_\ell$, or equivalently, $I_\ell<W^*_\ell+O_\ell$. The lemma then follows since I_ℓ, W^*_ℓ and O_ℓ are integers. □

We now prove Lemma 14. Note that we consider only the lazy intervals in which at least one Type-1 job arrives, and ignore those lazy intervals without any Type-1 job. Let $\Delta rwc'_\ell$ be the increase in *rwc* to WPOOL due to Type-1 jobs arriving in a lazy interval ℓ, and let $\Delta rwc'$ the increase in *rwc* to WPOOL due to all Type-1 jobs, respectively. Hence summing $\Delta rwc'_\ell$ over all ℓ gives $\Delta rwc'$. Let L be the set of all lazy intervals. Then $|L|$ is the total number of

lazy intervals. Define $I_L = \sum_{\ell \in L} I_\ell$ and similarly for O_L and W_L^*. It is useful to define E_ℓ^* to be the total wake-up energy plus the total static energy used by OPT during ℓ, and E_L^* to be the sum of E_ℓ^* over all $\ell \in L$. Obviously, $E_L^* \le \tilde{G}^*$. We will prove the following three relationships.

$$(A) \quad \Delta rwc' < (I_L + 3|L|)\lambda;$$
$$(B) \quad (W_L^* + O_L - |L|)\lambda \le E_L^*;$$
$$(C) \quad |L|\lambda \le E_L^* + 2\tilde{G}^*.$$

(A), (B) and (C), together with Property 17, would imply Lemma 14 immediately. The argument is as follows. By Property 17, for each $\ell \in L$, $I_\ell + 1 \le W_\ell^* + O_\ell$. Summing over all $\ell \in L$ gives $I_L + |L| \le W_L^* + O_L$. Therefore,

$$\begin{aligned} \Delta rwc' &< (I_L + 3|L|)\lambda \le (W_L^* + O_L + 2|L|)\lambda \\ &\le E_L^* + 3|L|\lambda \le 4E_L^* + 6\tilde{G}^* \le 10\tilde{G}^*. \end{aligned}$$

We come to the conclusion of this section.

Proof of Lemma 11. By Lemmas 13 and 14, the increase of *rwc* due to Type-0 and Type-1 jobs is at most $(1+10)\tilde{G}^*$, and hence the increase in Φ is at most $11(1+\frac{1}{\epsilon})\tilde{G}^*$. On the other hand, by Lemma 12, the increase of Φ due to Type-2 jobs is at most $\frac{3}{\epsilon}\cdot\tilde{G}^*$. Therefore, the total increase in Φ is at most $(11+\frac{14}{\epsilon})\tilde{G}^*$. □

It remains to prove (A), (B) and (C).

Proof of (A). Consider a lazy interval ℓ. Note that WPOOL adds a machine to P whenever the accumulated increase in *rwc* due to arrived jobs exceeds λ and $|P| < m$, and when $|P| = m$, arriving jobs are not Type-1. By the assumption that job arrival times are distinct, at most one job arrives at t_2. If that job is Type-1, by Property 15, it can cause at most two machines to be added to $|P|$. Thus, the accumulated increase in *rwc* due to Type-1 jobs arriving in ℓ is only enough to cause at most $I_\ell + 2$ machine additions to P, i.e., $\lfloor \frac{\Delta rwc'_\ell}{\lambda} \rfloor \le I_\ell + 2$. Note that $\lfloor \frac{\Delta rwc'_\ell}{\lambda} \rfloor > \frac{\Delta rwc'_\ell}{\lambda} - 1$ and hence $\Delta rwc'_\ell < (I_\ell + 3)\lambda$. Summing over all $\ell \in L$, $\Delta rwc' < (I_L + 3|L|)\lambda$. □

Proof of (B). Consider a lazy interval ℓ. For each time Opt wakes up a machine, we charge its wakeup energy λ. For each time except the first that WPOOL removes a machine from P, λ units of idling energy (counted in the counter $\mathrm{X_{IE}}$) must be accumulated inside ℓ by WPOOL. During the lazy interval ℓ, Opt has at least the same number of awake machines as WPOOL and must also have incurred λ units of static energy when WPOOL accumulates the λ units of idling energy. Therefore, $(W_\ell^* + O_\ell - 1)\lambda \le E_\ell^*$. Summing over all $\ell \in L$, $(W_L^* + O_L - |L|)\lambda \le E_L^*$. □

Proof of (C). For each lazy interval ℓ, we want to show that either $E_\ell^* \ge \lambda$ or we can charge λ non-overlapping units from $2G^*$. Then (C) follows. Let us first consider two trivial cases of ℓ. Suppose $(W_\ell^* + O_\ell) \ge 2$, then the charging scheme in the proof of (B) implies $E_\ell^* \ge \lambda$. Next, if $W_\ell^* = 1$ and $O_\ell = 0$, then we can charge the wakeup energy of Opt and $E_\ell^* \ge \lambda$. Note that the costs of E_ℓ^* charged in the two cases are non-overlapping.

It remains to consider the case when $W_\ell^* = 0$ and $O_\ell = 1$, which is indeed non-trivial. We call ℓ a lazy-01 interval. We will use another charging scheme to charge λ units from $2\tilde{G}^*$; in other words, any cost of Opt is charged at most twice. Let $\ell = [t_1, t_2]$. We first show that $t_1 \ne t_2$. Suppose, for the sake of contradiction, $t_1 = t_2$. Since we only consider lazy intervals in which Type-1 job arrives, by the assumption that job arrival times are distinct, one job arrives at t_1 and this job is a Type-1 job. By definition, $h_{\_}(t_1) > h_{\_}^*(t_1)$ and $h_{R}(t_1) \le h_{R}^*(t_1)$. Together with the assumption that $W_\ell^* = 0$, WPOOL must have removed a machine from P at t_1. By the definition of WPOOL, there exists another idle machine in P, and WPOOL would dispatch the job that arrives at t_1 to this zero-*rwc* machine, so the job is Type-0, contradicting that it is Type-1. Therefore, we must have $t_1 \ne t_2$.

By the definition of lazy interval, we have $h_{\_}(t_1) > h_{\_}^*(t_1)$, and for any $t \in (t_1, t_2]$, $h_{\_}(t) \le h_{\_}^*(t)$. Since $W_\ell^* = 0$ and $O_\ell = 1$, we can conclude that WPOOL removes a machine from P at t_1, and for any $t \in (t_1, t_2]$, $h_{\_}^*(t) = h_{\_}(t) \ge 1$, where the last inequality follows from that the size of P is always at least 1.

Let us consider all lazy-01 intervals in order of time and group them into disjoint subsequences such that in each subsequence $\{\ell_1, \ell_2, \cdots, \ell_p\}$ where $p \ge 1$, Opt does not change the state of any machine from the start of ℓ_1 to just before the end of ℓ_p (including the time in between ℓ_i and ℓ_{i+1} for all $1 \le i \le p-1$), and Opt wakes up at least one machine or puts at least one machine to sleep after the end of ℓ_p and before the start of any other lazy-01 interval.[1] If Opt wakes up a machine after ℓ_p and before another lazy-01 interval, then we charge its wakeup energy λ; otherwise, Opt puts a machine to sleep after ℓ_p and before another lazy-01 interval, and we charge the wakeup energy λ of the last wakeup in Opt before putting that machine to sleep. Thus, each wakeup cost of Opt is charged at most twice.

Next, we show how to charge for $\ell_2, \cdots, \ell_p$. By the definition of the subsequence, $h_{\_}^*(t)$ remains a constant, say M, which is at least 1, from the start of ℓ_1 to the end of ℓ_p; and inside each ℓ_i, $h_{\_}(t) = h_{\_}^*(t) = M$. As shown before, a lazy-01 interval starts with that WPOOL removes a machine from P. Thus, at some time after ℓ_{i-1} and before ℓ_i (for $2 \le i \le p$), λ units of idling energy is incurred when WPOOL has exactly $M+1$ machines in P. Since Opt has M awake machines during these times and $M \ge \frac{M+1}{2}$, Opt has incurred at least $\frac{\lambda}{2}$ static energy, which can be charged twice for ℓ_i (giving an amount of at least λ units of static energy). Note that this static energy will not be charged again by any other lazy-01 interval. Therefore, we have charged Opt a cost of at most 2λ for each lazy-01 interval.

Summing over the three types of intervals, $|L|\lambda \le E_L^* + 2\tilde{G}^*$. □

3.4 Running condition

We show the following running condition of the potential analysis of $\tilde{F}_{\mathrm{W}}$, which considers how Φ changes in an infinitesimal amount of time $[t, t + \mathrm{d}t]$ when there is no job arrival or completion, and both WPOOL and Opt have no change in machine state.

Lemma 18. *Consider any time t without job arrival, completion, and machine state transition in both* WPOOL *and* Opt. *Then* $\frac{\mathrm{d}\tilde{F}_{\mathrm{W}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot \frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t} + \frac{1}{\epsilon}\frac{\mathrm{d}\tilde{F}_{\mathrm{S}}}{\mathrm{d}t}$.

[1] At least one sleep must exist since $h_{\_}^*(t) \ge 1$ and Opt must put this machine to sleep after ℓ_p.

Since $\Phi(t) = \sum_{j=1}^{m} \Phi_i(t)$, we analyze on a per-machine basis. We focus on analyzing a certain machine i in WPOOL and the matching machine $x(i)$ in Opt. Let t be the current time. Let $\tilde{w}_i(t)$ and $s_i(t)$ be respectively the total fractional weight and the speed of machine i in WPOOL at time t. Define $\tilde{w}^*_{x(i)}(t)$ and $s^*_{x(i)}(t)$ similarly for machine $x(i)$ in Opt. We drop the parameter t when it refers to the current time.

Lemma 19. *Suppose $\tilde{w}_i > 0$.* **(i)** *If $\tilde{w}^*_{x(i)} > \tilde{w}_i + \mu$, $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} = 0$.* **(ii)** *If $\tilde{w}^*_{x(i)} \le \tilde{w}_i + \mu$, then $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le \frac{1}{\epsilon}(\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)(-s_i + s^*_{x(i)})$*

Proof. We focus on machine i in WPOOL and machine $x(i)$ in Opt, and consider how the processing of WPOOL and Opt changes Φ_i. Let q_a and q_o be the smallest inverse density of an active job in machine i of WPOOL and machine $x(i)$ of Opt, respectively (0 if no active jobs). The processing of WPOOL causes $\tilde{w}_i(q)$ to decrease by $\frac{s_i \mathrm{d}t}{q_\mathrm{a}}$ for $q \in [0, q_\mathrm{a}]$. Similarly, the processing of Opt causes $\tilde{w}^*_{x(i)}(q)$ to decrease by $\frac{s^*_{x(i)}\mathrm{d}t}{q_\mathrm{o}}$ for $q \in [0, q_\mathrm{o}]$.

(i) Suppose $\tilde{w}^*_{x(i)} > \tilde{w}_i + \mu$. Processing of WPOOL cannot increase Φ_i. We focus on bounding the increase in Φ_i due to Opt. For any $q \in [0, q_\mathrm{o}]$, $\tilde{w}^*_{x(i)}(q) = \tilde{w}^*_{x(i)} > \tilde{w}_i + \mu \ge \tilde{w}_i(q) + \mu$, so $\tilde{w}_i(q) - \tilde{w}^*_{x(i)}(q) + \mu < 0$. Thus, Φ_i does not increase due to Opt and $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le 0$.

(ii) $\tilde{w}^*_{x(i)} \le \tilde{w}_i + \mu$. Consider the change to Φ due to the working of WPOOL. The inner integral of Φ decreases by $\int_{\tilde{w}_i(q) - \frac{s_i \mathrm{d}t}{q_\mathrm{a}}}^{\tilde{w}_i(q)} (z - \tilde{w}^*_{x(i)}(q) + \mu)_+ \,\mathrm{d}z = \frac{s_i \mathrm{d}t}{q_\mathrm{a}}(\tilde{w}_i(q) - \tilde{w}^*_{x(i)}(q) + \mu)$. The change of Φ due to WPOOL is $-\frac{1}{\epsilon}\int_0^{q_\mathrm{a}} [\frac{s_i \mathrm{d}t}{q_\mathrm{a}}(\tilde{w}_i(q) - \tilde{w}^*_{x(i)}(q) + \mu)]\,\mathrm{d}q \le -\frac{1}{\epsilon}\int_0^{q_\mathrm{a}} [\frac{s_i \mathrm{d}t}{q_\mathrm{a}}(\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)]\,\mathrm{d}q = -\frac{1}{\epsilon} s_i \mathrm{d}t (\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)$. Similarly, the change of Φ due to Opt is $\frac{1}{\epsilon} s^*_{x(i)} \mathrm{d}t (\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)$. □

Lemma 19 allows us to prove Lemma 18 as follows.

Proof of Lemma 18. We focus on machine i in WPOOL and machine $x(i)$ in Opt. Let $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i$ be the rate of change of $\tilde{F}_\mathrm{W}$ due to machine i, and $\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t} = \sum_{i=1}^{m} (\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i$. Similarly define $(\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_i$ and $(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$ for $\tilde{G}^*$ and $\tilde{F}_\mathrm{S}$. To show the lemma, it suffices to show that $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$. If $\tilde{w}_i = 0$, WPOOL is not working and hence $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i = 0$. Furthermore, $\tilde{w}_i(q) = 0$ for all q, so $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} = 0$. It is trivial that $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$. Henceforth, we assume $\tilde{w}_i > 0$.

We will consider four cases depending on s_i and $s^*_{x(i)}$.

Case 1: $s_i > 0$ and $s^*_{x(i)} > 0$. In this case, we have $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i = \tilde{w}_i$, $(\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} \ge \tilde{w}^*_{x(i)}$, $s_i = 1 + \epsilon$ and $s^*_{x(i)} = 1$. We split the analysis into subcases depending on $\tilde{w}_i$ and $\tilde{w}^*_{x(i)}$.

Case 1.1: $\tilde{w}^*_{x(i)} > \tilde{w}_i + \mu$. By Lemma 19, $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} = 0$. Thus, $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} = \tilde{w}_i \le \tilde{w}^*_{x(i)} \le (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$.

Case 1.2: $\tilde{w}^*_{x(i)} \le \tilde{w}_i + \mu$. By Lemma 19, $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le \frac{1}{\epsilon}(\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)(-s_i + s^*_{x(i)})$. Thus, $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le \tilde{w}_i - \frac{1}{\epsilon}(\tilde{w}_i - \tilde{w}^*_{x(i)} + \mu)(-s_i + s^*_{x(i)}) \le \tilde{w}^*_{x(i)} + \mu \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$.

Case 2: $s_i > 0$ and $s^*_{x(i)} = 0$. In this case, we have $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i = \tilde{w}_i$, $(\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} \ge \tilde{w}^*_{x(i)}$, $s_i = 1 + \epsilon$ and $s^*_{x(i)} = 0$. Similarly as in Case 1, we can divide the analysis into subcases and verify in each subcase that $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$.

Case 3: $s_i = 0$ and $s^*_{x(i)} > 0$. For WPOOL, $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i = 0$ but $(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i = \tilde{w}_i$. For Opt, $(\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} \ge \tilde{w}^*_{x(i)} + \mu$. Furthermore, $s_i = 0$ and $s^*_{x(i)} = 1$. Similarly as before, we can also divide the analysis into subcases and verify that $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$.

Case 4: $s_i = 0$ and $s^*_{x(i)} = 0$. In this case, $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i = 0$. By Lemma 19, $\frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le 0$, so it is trivial that $(\frac{\mathrm{d}\tilde{F}_\mathrm{W}}{\mathrm{d}t})_i + \frac{\mathrm{d}\Phi_i}{\mathrm{d}t} \le (1 + \frac{1}{\epsilon}) \cdot (\frac{\mathrm{d}\tilde{G}^*}{\mathrm{d}t})_{x(i)} + \frac{1}{\epsilon}(\frac{\mathrm{d}\tilde{F}_\mathrm{S}}{\mathrm{d}t})_i$. □

References

Albers, S. (2010), 'Energy-efficient algorithms', *CACM*, **53**(5), 86–96.

Anand, S., Garg, N., & Kumar, A. (2012), Resource augmentation for weighted flow-time explained by dual fitting, *in* 'Proc. SODA', pp. 1228–1241.

Bansal, N. & Chan, H.L. (2009), Weighted flow time does not admit $O(1)$-competitive algorithms, *in* 'Proc. SODA', pp. 1238–1244.

Bansal, N., Pruhs, K., & Stein, C. (2009), 'Speed scaling for weighted flow time', *SIAM Journal on Computing*, **39**(4), 1294–1308.

Becchetti, L., Leonardi, S., Marchetti-Spaccamela, A. & Pruhs., K. (2006), 'Online weighted flow time and deadline scheduling'. *J. Discrete Algorithms*, **4**(3), 339–352.

Belady, C. (2007), 'In the data center, power and cooling costs more than the IT equipment it supports', *Electronics Cooling Magazine*, **13**(1), 24–27.

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

Chadha, J., Garg, N., Kumar, A. & Muralidhara, V. (2009), A competitive algorithm for minimizing weighted flow time on unrelated machines with speed augmentation, *in* 'Proc. STOC', pp. 679–684.

Chan, S.H., Lam, T.W., Lee, L.K., Liu, C.M. & Ting, H.F. (2011), Sleep management on multiple machines for energy and flow time, *in* 'Proc. ICALP', pp. 219–231.

Gupta, A., Krishnaswamy, R. & Pruhs, K. (2010), Scalably scheduling power-heterogeneous processors, *in* 'Proc. ICALP', pp. 312-323.

Khuller, S., Li, J. & Saha, B. (2010), Energy efficient scheduling via partial shutdown, *in* 'Proc. SODA', pp. 1360-1372.

Lam, T.W., Lee, L.K., Ting, H.F., To, I. & Wong, P. (2009), Sleep with guilt and work faster to minimize flow plus energy, *in* 'Proc. ICALP', pp. 665–676.

Lam, T. W., Lee, L. K., To I. & Wong, P. (2008), Speed scaling functions for flow time scheduling based on active job count, *in* Proc. ESA, pp. 647–659.

Appendix A: Omitted Proofs

In this appendix, we give the proofs omitted from Section 3.

Lemma 12. *The total increase in Φ due to Type-2 jobs is at most $\frac{3}{\epsilon}\cdot\tilde{G}^*$.*

Proof. We analyze job by job. Consider a Type-2 job j arriving at time t. Let Δrwc and Δrwc^* be the increase in the rwc due to j in WPOOL and Opt, respectively. We will show that after WPOOL and OPT dispatch j, the change of Φ, denoted $\Delta\Phi$, is at most $\frac{3}{\epsilon}\cdot\Delta rwc^*$. Summing Δrwc^* over all jobs equals the working cost of Opt, which is at most $\tilde{G}^*$. Hence, the total increase in Φ due to Type-2 jobs is at most $\frac{3}{\epsilon}\cdot\tilde{G}^*$.

Suppose WPOOL and Opt dispatch j to machines i and k, respectively. Let $x(i)$ be the matching function updated just before dispatching j. Also define u such that $x(u)=k$. Both $\tilde{w}_i(q)$ and $\tilde{w}^*_k(q)$ increase by $w(j)$ for $q\in[0,q(j)]$. We consider two cases depending on whether $x(i)=k$.

Case 1. When $x(i)\neq k$, i.e. $i\neq u$. We can show that when j arrives at time t, u is in P just before WPOOL dispatches j. The argument is as follows. Consider $H_{R}(t)$, since j is a Type-2 job, $h_{R}(t)>h^*_{R}(t)$ or $h_{R}(t)=m$. By definition, Opt has at most one procrastinating machine at time t, or precisely, with respect to $H_{-}(t)$, $H_{+}(t)$ and hence $H_{R}(t)$. Thus, at $H_{R}(t)$, $|P|$ $(=h_{R}(t))$ is at least the number of awake or procrastinating machines in Opt. All those machines in Opt, including k, must be matched with a machine in P. As $x(u)=k$, we have $u\in P$.

By Lemma 3, $\Delta rwc^* = \int_0^{q(j)} w(j)\big(\tilde{w}^*_{x(u)}(q)+\frac{1}{2}w(j)+\mu\big)\,\mathrm{d}q$. Note that $\Delta\Phi=\Delta\Phi_i+\Delta\Phi_u$. First consider $\Delta\Phi_i$. $\Delta\Phi_i = \frac{1}{\epsilon}\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z-\tilde{w}^*_{x(i)}(q)+\mu)_+\,\mathrm{d}z\,\mathrm{d}q \le \frac{1}{\epsilon}\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z+\mu)\,\mathrm{d}z\,\mathrm{d}q$. As shown above, $u\in P$ when WPOOL handles j, but WPOOL assigns j to machine i instead of u. By definition of WPOOL, $\Delta rwc = \int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z+\mu)/(1+\epsilon)\,\mathrm{d}z\,\mathrm{d}q \le \int_0^{q(j)}\int_{\tilde{w}_u(q)}^{\tilde{w}_u(q)+w(j)}(z+\mu)/(1+\epsilon)\,\mathrm{d}z\,\mathrm{d}q$. Hence, we have

$$\begin{aligned}\Delta\Phi_i &\le \frac{1}{\epsilon}\int_0^{q(j)}\int_{\tilde{w}_u(q)}^{\tilde{w}_u(q)+w(j)}(z+\mu)\,\mathrm{d}z\,\mathrm{d}q\\ &= \frac{1}{\epsilon}\int_0^{q(j)} w(j)\left(\tilde{w}_u(q)+\frac{1}{2}w(j)+\mu\right)\mathrm{d}q.\end{aligned}$$

Then consider $\Delta\Phi_u$. For any $q\in[0,q(j)]$, let $\delta(q)=\int_0^{\tilde{w}_u(q)}\big((z-\tilde{w}^*_{x(u)}(q)+\mu)_+-(z-\tilde{w}^*_{x(u)}(q)-w(j)+\mu)_+\big)\,\mathrm{d}z$. Then $-\Delta\Phi_u=\frac{1}{\epsilon}\int_0^{q(j)}\delta(q)\,\mathrm{d}q$. Define $S\subseteq[0,q(j)]$ such that $\tilde{w}^*_{x(u)}(q)+w(j)-\mu\le 0$ for all $q\in S$. Define $\bar{S}=[0,q(j)]\setminus S$. For any $q\in S$, $\delta(q)=w(j)\tilde{w}_u(q)$; for any $q\in\bar{S}$, $\delta(q)\ge w(j)\big(\tilde{w}_u(q)-\tilde{w}^*_{x(u)}(q)-w(j)+\mu\big)$. Hence, we have

$$\begin{aligned}-\Delta\Phi_u &\ge \frac{1}{\epsilon}\Big(\int_{q\in S} w(j)\tilde{w}_u(q)\,\mathrm{d}q+\\ &\qquad\int_{q\in\bar{S}} w(j)\left(\tilde{w}_u(q)-\tilde{w}^*_{x(u)}(q)-w(j)+\mu\right)\mathrm{d}q\Big)\end{aligned}$$

It follows from the bounds of $\Delta\Phi_i$ and $\Delta\Phi_u$ that,

$$\begin{aligned}\Delta\Phi &= \Delta\Phi_i+\Delta\Phi_u\\ &\le \frac{1}{\epsilon}\Big(\int_{q\in S} w(j)\Big(\frac{1}{2}w(j)+\mu\Big)\,\mathrm{d}q\\ &\quad+\int_{q\in\bar{S}} w(j)\Big(\tilde{w}^*_{x(u)}(q)+\frac{3}{2}w(j)\Big)\,\mathrm{d}q\Big)\\ &\le \frac{1}{\epsilon}\left(\int_0^{q(j)} w(j)\Big(\tilde{w}^*_{x(u)}(q)+\frac{3}{2}w(j)+\mu\Big)\,\mathrm{d}q\right)\\ &\le \frac{3}{\epsilon}\Delta rwc^*\end{aligned}$$

Case 2. When $x(i)=k$, i.e. $i=u$. By Lemma 3, $\Delta rwc^*=\int_0^{q(j)} w(j)\big(\tilde{w}^*_{x(i)}(q)+\frac{1}{2}w(j)+\mu\big)\,\mathrm{d}q$. Note that $\Delta\Phi=\Delta\Phi_i$. For any $q\in[0,q(j)]$, let $\delta(q)=\int_0^{\tilde{w}_i(q)}\big((z-\tilde{w}^*_{x(i)}(q)+\mu)_+-(z-\tilde{w}^*_{x(i)}(q)-w(j)+\mu)_+\big)\,\mathrm{d}z$. Let $\Delta_1=\frac{1}{\epsilon}\int_0^{q(j)}\int_{\tilde{w}_i(q)}^{\tilde{w}_i(q)+w(j)}(z-\tilde{w}^*_{x(i)}(q)-w(j)+\mu)_+\,\mathrm{d}z\,\mathrm{d}q$, and let $\Delta_2=-\frac{1}{\epsilon}\int_0^{q(j)}\delta(q)\,\mathrm{d}q$. Then we have $\Delta\Phi_i=\Delta_1+\Delta_2$. Observe that Δ_1 and Δ_2, respectively, plays the same role as $\Delta\Phi_i$ and $\Delta\Phi_u$ in Case 1. By a similar calculation, we have $\Delta\Phi=\Delta\Phi_i=\Delta_1+\Delta_2\le\frac{3}{\epsilon}\Delta rwc^*$. □

Lemma 13. WPOOL*'s total increase in rwc due to Type-0 jobs is at most $\tilde{G}^*$.*

Proof. Recall that WPOOL dispatches a Type-0 job j to a zero-rwc machine. Suppose Opt dispatches j to machine k. By Lemma 3, the increase in rwc to WPOOL due to j is $\int_0^{q(j)}\int_0^{w(j)}(\frac{z+\mu}{1+\epsilon})\,\mathrm{d}z\,\mathrm{d}q$, which must be less than $\int_0^{q(j)}\int_{\tilde{w}^*_k(q)}^{\tilde{w}^*_k(q)+w(j)}(z+\mu)\,\mathrm{d}z\,\mathrm{d}q$ (the increase in rwc to Opt due to j). Summing over all Type-0 jobs, the total increase in rwc to WPOOL due to dispatching all Type-0 jobs is at most $\tilde{G}^*$. □