Competitive Non-migratory Scheduling for Flow Time and Energy

Tak-Wah Lam
Department of Computer Science
University of Hong Kong
twlam@cs.hku.hk

Lap-Kei Lee
Department of Computer Science
University of Hong Kong
lklee@cs.hku.hk

Isaac K. K. To
Department of Computer Science
University of Liverpool, UK
isaacto@liv.ac.uk

Prudence W. H. Wong *
Department of Computer Science
University of Liverpool, UK
pwong@liv.ac.uk

ABSTRACT

Energy usage has been an important concern in recent research on online scheduling. In this paper we extend the study of the tradeoff between flow time and energy from the single-processor setting [8, 6] to the multi-processor setting. Our main result is an analysis of a simple non-migratory online algorithm called CRR (classified round robin) on $m \geq 2$ processors, showing that its flow time plus energy is within $O(1)$ times of the optimal non-migratory offline algorithm, when the maximum allowable speed is slightly relaxed. This result still holds even if the comparison is made against the optimal migratory offline algorithm (the competitive ratio increases by a factor of 2.5). As a special case, our work also contributes to the traditional online flow-time scheduling. Specifically, for minimizing flow time only, CRR can yield a competitive ratio one or even arbitrarily smaller than one, when using sufficiently faster processors. Prior to our work, similar result is only known for online algorithms that needs migration [21, 23], while the best non-migratory result can achieve an $O(1)$ competitive ratio [14].

The above result stems from an interesting observation that there always exists some optimal migratory schedule $\mathcal{S}$ that can be converted (in an offline sense) to a non-migratory schedule $\mathcal{S}'$ with a moderate increase in flow time plus energy. More importantly, this non-migratory schedule always dispatches jobs in the same way as CRR.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*

General Terms

Algorithms, Performance, Theory

Keywords

Online scheduling algorithms, competitive analysis, dynamic speed scaling, energy minimization

*This research is partly supported by EPSRC Grant EP/E028276/1.

SPAA'08, June 14–16, 2008, Munich, Germany.

1. INTRODUCTION

Energy consumption has become a key issue in the design of modern processors. This is essential not only for battery-operated mobile devices with single processors but also for server farms or laptops with multi-core processors. A popular technology to reduce energy usage is *dynamic speed scaling* (see, e.g., [9, 15, 24, 28]) where the processor can vary its speed dynamically. Running a job at a slower speed is more energy efficient, yet it takes longer time and may affect the performance. In the past few years, a lot of effort has been devoted to revisiting classical scheduling problems with dynamic speed scaling and energy concern taken into consideration (e.g., [29, 7, 1, 8, 11, 2, 10, 26, 16]; see [17] for a survey). The challenge basically arises from the conflicting objectives of providing good "quality of service" (QoS) and conserving energy.

One commonly used QoS measurement for scheduling jobs on a processor is the total flow time (or equivalently, average response time). Here, jobs with arbitrary size are released at unpredictable times and the flow time of a job is the time elapsed since it arrives until it is completed. When energy is not a concern, the objective of a scheduler is simply to minimize the total flow time of all jobs. The study of energy-efficient scheduling was initiated by Yao, Demers and Shenker [29]. They considered deadline scheduling on a model where the processor can run at any speed between 0 and ∞, and incurs an energy of s^α per unit time when running at speed s, where $\alpha \geq 2$ (typically 2 or 3 [9, 22]). This model, which we call infinite speed model, also paves the way for studying scheduling that minimizes both the flow time and energy. In particular, Pruhs et al. [27] studied offline scheduling for minimizing the total flow time on a single processor with a given amount of energy. They gave a polynomial time optimal algorithm for the special case when jobs are of unit size. However, this problem does not admit

any constant competitive online algorithm even if jobs are of unit size [8].

Flow time and energy. To better understand the trade-off between flow time and energy, Albers and Fujiwara [1] proposed combining the dual objectives into a single objective of minimizing the sum of total flow time and energy. The intuition is that, from an economic viewpoint, it can be assumed that users are willing to pay a certain units (say, ρ units) of energy to reduce one unit of flow time. By changing the units of time and energy, one can further assume that $\rho = 1$ and thus would like to optimize total flow time plus energy. Albers and Fujiwara presented an online algorithm that is $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$-competitive for jobs of unit size. This result was recently improved by Bansal et al. [8], who gave a 4-competitive algorithm for jobs of unit size. They also considered the case for jobs with arbitrary size and weight, and presented an $O(1)$-competitive online algorithm for minimizing weighted flow time plus energy (precisely, the competitive ratio is $\mu_\epsilon \gamma_1$, where ϵ is any positive constant, $\mu_\epsilon = \max\{(1+1/\epsilon), (1+\epsilon)^{\alpha}\}$, and $\gamma_1 < 2\alpha/\ln\alpha$).

The infinite speed model has provided a convenient model for studying power management, yet it is not realistic to assume infinite speed. Recently, Chan et al. [11] introduced the bounded speed model, where the speed can be scaled between 0 and some maximum T. Bansal et al. [6] successfully adapted the previous results on minimizing flow time plus energy to this model. For jobs of unit size and unit weight, they gave a 4-competitive online algorithm. For jobs of arbitrary size and weight, they gave a $(\mu_\epsilon \gamma_2)$-competitive algorithm that uses a processor with maximum speed $(1+\epsilon)T$ for any $\epsilon > 0$, where $\gamma_2 = (2 + o(1))\alpha/\ln\alpha$.

Multiprocessor scheduling. All the results above are about single-processor scheduling. In the older days, when energy was not a concern, flow time scheduling on multiple processors running at fixed speed was an interesting problem by itself (e.g., [4, 5, 19, 25, 13, 14]). In this setting, jobs remain sequential in nature and cannot be executed by more than one processor in parallel. Different online algorithms like SRPT and IMD that are $\Theta(\log P)$-competitive have been proposed respectively under the migratory and the non-migratory model, where P is the ratio of the maximum job size to the minimum job size. Recently, IMD is further shown to be $O(1+\epsilon)$-competitive, when using processors $(1+\epsilon)$ times faster [14]; if migration is allowed, SRPT can achieve a competitive ratio one or even smaller, when using processors two or more times faster [21, 23].

The only previous work on multi-processors taking flow time and energy into consideration was by Bunde [10], which is about an offline approximation algorithm for jobs of unit size. The literature also contains some multi-processor results on optimizing other classical objectives together with energy in the infinite speed model. Pruhs et al. [26] and Bunde [10] both studied offline algorithms for the makespan objective. Albers et al. [2] studied online algorithms for scheduling jobs with restricted deadlines.

Our work on flow time plus energy. We extend the study of online scheduling for minimizing flow time plus energy to the setting of $m \geq 2$ processors. This extension is not only of theoretical interest. Modern processors utilize speed scaling as well as multi-core technology (dual-core and quad-core are getting common); and a multi-core processor is essentially a pool of processors. To make the work more meaningful, we aim at schedules without migrating jobs among processors. In practice, migrating jobs requires overheads and is avoided in many applications.

To save energy in a multiprocessor setting, we want to balance the load of the processors so as to avoid running any processors at high speed. It is natural to consider some kind of round-robin strategy to dispatch jobs. Typical examples include IMD for flow time scheduling [4] and CRR for energy-efficient deadline scheduling in the infinite speed model [2]. The main contribution of this paper is a non-trivial analysis of CRR (classified round robin) for optimizing flow time plus energy. Unlike [2], we apply CRR according to job size rather than job density. Specifically, when a job arrives, CRR dispatches it immediately based on the following notion of *classes* of job sizes. Consider some $\lambda > 0$. A job is said to be in class k if its size is in the range $((1+\lambda)^{k-1}, (1+\lambda)^{k}]$. Jobs of the same class are dispatched to the m processors using a round-robin strategy (i.e., the i-th job of a class will be dispatched to processor $(i \bmod m)$). It is worth-mentioning that IMD is slightly more complicated than CRR as it dispatches a job to the processor with the smallest accumulated load of the corresponding class.

The most non-trivial result in this paper is that CRR always admits a non-migratory schedule S whose flow time plus energy is within a constant factor of the optimal migratory offline schedule. In an online setting, we do not know how to compute this S, in particular how each individual processor schedules jobs in S. Yet we can approximate S by using the online algorithm BPS [8, 6] separately for each processor. Since BPS is $O(1)$-competitive for minimizing flow time plus energy in a single processor [8, 6], CRR plus BPS would give a competitive result for the multiprocessor setting. Below is a summary of the results based on the bounded speed model (where T denotes the maximum speed). Our analysis can also be applied to the infinite speed model, but it is of less interest. In addition to the constants $\mu_\epsilon, \gamma_1, \gamma_2$ used in [8, 6], it is convenient to define a constant $\eta_\epsilon = (1+\epsilon)^{\alpha}[(1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\epsilon)/\epsilon^2]$.

- Against the optimal non-migratory schedule: For any $\epsilon > 0$, CRR-BPS can be $(2\eta_\epsilon \mu_\epsilon \gamma_2)$-competitive for minimizing flow time plus energy, when the maximum allowable speed is relaxed to $(1+\epsilon)^3 T$. E.g., if $\alpha = 2$ and $\epsilon = 0.6$, the competitive ratio is at most $27\mu_\epsilon\gamma_2$.
- Against the optimal migratory schedule: The competitive ratio becomes $5\eta_\epsilon \mu_\epsilon \gamma_2$.

Implication for flow-time scheduling. Our work also contributes to the study of traditional flow-time scheduling, which assumes fixed-speed processors. We adapt the above result to show that for minimizing flow time only, CRR (plus SRPT for individual processor) would give a non-migratory online algorithm which, when compared with the optimal migratory algorithm, has a competitive ratio of one or even any constant arbitrarily smaller than one, when using sufficiently fast processors. Prior to our work, similar result is known only for online algorithms that need migration; in particular, McCullough and Torng [21] showed that for $m \geq 2$ processors, SRPT is s-speed $\frac{1}{s}$-competitive for flow time, where $s \geq 2 - \frac{1}{m}$. Our non-migratory result is less efficient. More precisely, for any $s \geq 1$, it is s-speed $\frac{5(\sqrt{s}+1)}{(\sqrt{s}-1)^2}$-competitive. E.g., if $s = 64$, the competitive ratio is 0.92.

This paper is about online algorithms, yet the key technique is an offline transformation. Given an optimal migratory (or non-migratory) offline schedule $\mathcal{O}$, we show how to construct a schedule S that follows CRR to dispatch jobs with the total flow time plus energy increasing by a constant factor only. Note that S takes advantage of $\mathcal{O}$ to determine how to schedule jobs and scale the speed within each processor.

1.1 Definitions, notations, and a simple lower bound

Given a job set $\mathcal{J}$, we want to schedule $\mathcal{J}$ on a pool of $m \geq 2$ processors. Note that jobs are sequential in nature and cannot be executed by more than one processor in parallel. All processors are identical and a job can be executed in any processor. Preemption is allowed and a preempted job can be resumed at the point of preemption. We differentiate two types of schedules: a migratory schedule can move partially-executed jobs from one processor to another processor without any penalty, and a non-migratory schedule cannot.

We use $r(j)$ and $p(j)$ to denote respectively the release time and work requirement (or size) of job j. We let $p(\mathcal{J}) = \sum_{j \in \mathcal{J}} p(j)$ be the total size of $\mathcal{J}$. The time required to complete job j using a processor with fixed speed s is $p(j)/s$.

With respect to a schedule $\mathcal{S}$ of a job set $\mathcal{J}$, we use the notations *max-speed*($\mathcal{S}$), $E(\mathcal{S})$ and $F(\mathcal{S})$ for the maximum speed, energy usage, and total flow time of $\mathcal{S}$, respectively. Note that $F(\mathcal{S})$ is the sum, over all jobs, of the time since a job is released until it is completed, or equivalently, the integration over time of the number of unfinished jobs. As processor speed can vary dynamically and the time to execute a job j is not necessarily equal to $p(j)$, we define the *execution time* of job j to be the flow time minus waiting time of j, and define the total execution time of $\mathcal{S}$ to be the sum of execution time over all jobs in $\mathcal{S}$.

It is convenient to define $G(\mathcal{S}) = F(\mathcal{S}) + E(\mathcal{S})$. The following lemma shows a lower bound on $G(\mathcal{S})$ which depends on $p(\mathcal{J})$, irrelevant of the number of processors.

LEMMA 1. *For any m-processor schedule $\mathcal{S}$ for a job set $\mathcal{J}$, $G(\mathcal{S}) \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} p(\mathcal{J})$.*

PROOF. Suppose that a job in $\mathcal{S}$ has flow time t. The energy usage for j is minimized if j is run at constant speed $p(j)/t$ throughout, and it is at least $(p(j)/t)^\alpha t = p(j)^\alpha / t^{\alpha-1}$. Since $t + p(j)^\alpha/t^{\alpha-1}$ is minimized when $t = (\alpha-1)^{1/\alpha} p(j)$, we have $t + p(j)^\alpha/t^{\alpha-1} \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} p(j)$. Summing over all jobs, we obtain the desired lower bound. □

2. THE ONLINE ALGORITHM

This section presents the formal definition of the online algorithm CRR-BPS, which produces a non-migratory schedule for $m \geq 2$ processors. In the following sections, we will show that, when compared with the optimal non-migratory or migratory schedule, this algorithm is $O(1)$-competitive for flow time plus energy, when the maximum allowable speed is slightly relaxed.

Consider any $\lambda > 0$. Recall that a job is said to be in class k if its size is in the range $((1+\lambda)^{k-1}, (1+\lambda)^k]$. In a *CRR($\lambda$)-dispatching* schedule, jobs of the same class are dispatched upon their arrival (ties are broken using job ids) to the m processors using a round-robin strategy, and different classes are handled independently. Jobs once dispatched to a processor will be processed there entirely; thus a CRR(λ)-dispatching schedule is non-migratory in nature. For notational ease, when the value of λ is clear in the context, we will simply use the term CRR-dispatching.

The intuition of using a CRR-dispatching schedule comes from a new offline result that there is a CRR-dispatching schedule such that the total flow time plus energy is $O(1)$ times that of the optimal (non-migratory or migratory) offline schedule and the maximum allowable speed is only slightly higher. Details are stated in Theorem 2 below (Sections 3, 4 and 5 are devoted to proving this theorem).

THEOREM 2. *Given a job set $\mathcal{J}$, let $\mathcal{N}^*$ be an optimal non-migratory schedule of $\mathcal{J}$, and let $\mathcal{S}^*$ be an optimal migratory schedule of $\mathcal{J}$. Then for any $\lambda, \epsilon > 0$,*

i. there is a CRR(λ)-dispatching schedule $\mathcal{S}$ for $\mathcal{J}$ such that $G(\mathcal{S}) \leq 2(1+\lambda)^\alpha((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon) G(\mathcal{N}^*)$, and max-speed($\mathcal{S}$) $\leq (1+\lambda)(1+\epsilon) \times$ max-speed($\mathcal{N}^*$); and

ii. there is a CRR(λ)-dispatching schedule $\mathcal{S}'$ for $\mathcal{J}$ such that $G(\mathcal{S}') \leq 5(1+\lambda)^\alpha((1+\epsilon)^{\alpha-1} + (1-1/\alpha)(2+\lambda)/\lambda\epsilon) G(\mathcal{S}^*)$, and max-speed($\mathcal{S}'$) $\leq (1+\lambda)(1+\epsilon) \times$ max-speed($\mathcal{S}^*$).

Theorem 2 naturally suggests an online algorithm that first dispatches jobs using a CRR(λ)-dispatching policy, and then schedules jobs in each processor independently and in a way that is competitive in the single-processor setting. For the latter we make use of the single-processor algorithm BPS [8, 6]. Below we review the algorithm BPS (the definition is for reference only, in this paper we only need to know the performance of BPS), and define the multi-processor algorithm CRR$_\lambda$-BPS$_\epsilon$, where $\lambda, \epsilon > 0$ are constants.

> **Algorithm BPS.** At any time t, run the job with the smallest size at speed $w_a(t)^{1/\alpha}$, where $w_a(t)$ is the sum of the remaining fraction (i.e., remaining work divided by original work) of all jobs. If $w_a(t)^{1/\alpha}$ exceeds the maximum allowable speed T (if any), just use the maximum speed T.
>
> **Algorithm BPS$_\epsilon$.** At any time t, select the job with the smallest size to execute, and the speed to be used is $(1+\epsilon)$ times the current speed of a simulated BPS schedule on the jobs. Note that the maximum allowable speed is relaxed to $(1+\epsilon)T$.
>
> **Algorithm CRR$_\lambda$-BPS$_\epsilon$.** Jobs are dispatched to the m processors with the CRR(λ)-dispatching policy. Jobs in each processor are scheduled independently using BPS$_\epsilon$.

It is known that BPS$_\epsilon$ performs well in minimizing flow time plus energy for a single processor in the infinite speed model [8] as well as the bounded speed model [6]. Recall that the constants μ_ϵ, γ_1 and γ_2 are defined as $\max\{(1+1/\epsilon), (1+\epsilon)^\alpha\}$, $2\alpha/\ln\alpha$, and $(2+o(1))\alpha/\ln\alpha$, respectively.

LEMMA 3. *[6, 8] Consider any $\epsilon > 0$. For minimizing flow time plus energy on a single processor, BPS$_\epsilon$ is $\mu_\epsilon\gamma_1$-competitive in the infinite speed model, and $\mu_\epsilon\gamma_2$-competitive in the bounded speed model, when the maximum speed is relaxed to $(1+\epsilon)T$.*