Non-clairvoyant Speed Scaling for Batched Parallel Jobs on Multiprocessors

Hongyang Sun
Nanyang Technological University
50 Nanyang Avenue
Singapore 639798
sunh0007@ntu.edu.sg

Yangjie Cao
Xi'an Jiaotong University
No.28, Xianning West Road,
Xi'an, Shaanxi
710049, P.R. China
caoyj@stu.xjtu.edu.cn

Wen-Jing Hsu
Nanyang Technological University
50 Nanyang Avenue
Singapore 639798
hsu@ntu.edu.sg

ABSTRACT

Energy consumption and heat dissipation have become key considerations for modern high performance computer systems. In this paper, we focus on non-clairvoyant speed scaling to minimize flow time plus energy for batched parallel jobs on multiprocessors. We consider a common scenario where the total power consumption cannot exceed a given budget and the power consumed on each processor is s^α when running at speed s. Extending the EQUI processor allocation policy, we propose two algorithms: U-EQUI and N-EQUI, which use respectively a uniform-speed and a non-uniform speed scaling function for the allocated processors. Using competitive analysis, we show that U-EQUI is $O(P^{(\alpha-1)/\alpha^2})$-competitive for flow time plus energy, and N-EQUI is $O(\sqrt[\alpha]{\ln P})$-competitive for the same metric when given sufficient power, where P is the total number of processors. Our simulation results confirm that U-EQUI and N-EQUI achieve better performance than a straightforward fixed-speed EQUI strategy. Moreover, moderate power constraint does not significantly affect the performance of our algorithms.

Categories and Subject Descriptors

F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*Sequencing and scheduling*; D.4.1 [**Software**]: Process Management—*Scheduling, Multiprocessing*

General Terms

Algorithms, Design, Experimentation, Theory

Keywords

Scheduling, Speed scaling, Non-clairvoyant, Batched parallel jobs, Multiprocessors, Flow time, Energy consumption, Power Budget, Simulations

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CF'09, May 18–20, 2009, Ischia, Italy.
Copyright 2009 ACM 978-1-60558-413-3/09/05 ...$5.00.

1. INTRODUCTION

As processors continue to increase in performance and speed, energy consumption and heat dissipation have become key considerations in the design of modern high performance computer systems. These issues are crucial to battery-based computers, which will soon be equipped with an unprecedented large number of cores, as well as supercomputers or server farms, whose heat dissipation has become an increasingly critical problem. *Dynamic speed scaling* [8, 17] is a popular technique to reduce energy consumption by taking advantage of the nonlinear relationship between the processor's speed and power. If a processor operates at speed s, then its power consumption u satisfies $u = s^\alpha$, where $\alpha > 1$ and is typically 2 or 3 [8]. Since good system performance and low energy consumption are mutually conflicting objectives, how to optimally balance the two has become an active research topic recently. (See, e.g., [32, 19, 9, 1, 5, 24, 23, 27].)

One commonly used performance metric in computing systems is the *average response time*, or equivalently the *flow time* of the jobs in the system. The flow time of a job is the duration between the release time and the completion time of the job, and the problem of minimizing the total flow time without power constraints has been studied extensively in the literature. (See [26] for a survey.) To take the energy consumption into account, Pruhs et al. [27] first studied the problem of minimizing the total flow time subject to an energy constraint. They proposed an optimal offline algorithm for sequential jobs of unit size. Albers and Fujiwara [1] later proposed combining the conflicting objectives of the flow time and the energy consumption into a single objective of flow time plus energy, which has since become a popular metric for studying the power management problems [5, 24, 3, 23, 10]. In fact, minimizing a combination of flow time plus energy can be naturally interpreted by looking at both objectives from a unified viewpoint and measuring them in economic terms. Similar metrics have been used before in the scheduling literature [30, 12] that considers the scheduling costs as part of the objective functions.

Although many excellent results have been obtained for scheduling sequential jobs on a single processor using flow time plus energy as the performance metric [5, 24, 3, 23, 10], relatively little work is known for parallel jobs on multiprocessor systems. Most previous results [28, 11, 9, 2] on multiprocessor scheduling that take energy issues into consideration are offline algorithms either for optimizing the system performance with an energy constraint, or for minimizing

the total energy consumption subject to deadlines. In [23], Lam et al. focused on online non-migratory scheduling for flow time plus energy on multiprocessors. They considered sequential jobs and assumed that the work requirement of each job is known.

In this paper, we adopt the performance metric of flow time plus energy and study online speed scaling for batched (i.e., all jobs are released at time 0) parallel jobs on multiprocessor systems. Compared to sequential jobs scheduling on a single processor, schedulers for parallel jobs on multiprocessors need to have both a *processor allocation policy* for determining the number of processors allocated to each job and a *speed scaling policy* for determining the speed of each allocated processor, assuming that per core/processor speed scaling [29, 22] is possible. Furthermore, we focus on the *non-clairvoyant* scheduling model, in which no knowledge about the work and the parallelism of the jobs is assumed when making scheduling decisions. Prior to our work, Lam et al. [24] and Chan et al. [10] have considered non-clairvoyant scheduling in speed scaling context. However, both results studied scheduling sequential jobs on a single processor while focusing on batched and non-batched settings, respectively. Lam et al. [24] also considered a bounded speed scenario where the processor speed at any time can not exceed a given maximum. In contrast to [24], we consider a scenario where at any time the total power consumption on all processors can not exceed a given power budget. This assumption has been previously considered by Isci et al. [20] in their empirical study.

As with Edmonds et al. [16, 15], we allow a parallel job to have time-varying execution characteristics and model it by multiple phases of speedup functions. However, unlike in [16, 15], where each phase of a job admits an arbitrary non-decreasing but sub-linear speedup, we consider a model where each phase has a linear speedup function up to a certain degree of parallelism, beyond which no further speedup can be gained. The execution of a given phase then depends on this ceiling parallelism, the number of processors allocated to it as well as the speed of the allocated processors. We consider two scenarios in this paper depending on whether a parallel job is executed by processors of the same speed or different speeds at any given time. We call a scheduling algorithm *uniform-speed* if it sets all processors allocated to a job to run at the same speed at any given time; otherwise, we call it *nonuniform-speed*. Setting processors to run at different speeds for a parallel job can have complicated effects on the job's execution, subject to the underlying dependencies among the tasks (or threads) of the job as well as the load balancing strategy. To ease analysis, we make the following simplifying assumptions. If the number of allocated processors is more than the maximum degree of parallelism, say h, in the current phase, then the *maximum utilization policy* [21, 6] is employed at the underlying task scheduling level such that the h fastest processors are used to execute the job. Otherwise, if there is sufficient parallelism in this phase, all allocated processors are used to execute the job. We further assume that the tasks of a phase are able to run independently of each other, such as those in some data-parallel jobs, and ideal load balancing can be achieved in scheduling the tasks, which can be approximated by the round-robin strategy provided that the phase is long enough. Hence, for both uniform-speed and nonuniform-speed schedules, the execution rate of a phase at any time is equal to the sum of the speeds of the processors utilized to execute the phase.

We propose a uniform-speed scheduling algorithm and a nonuniform-speed scheduling algorithm based on the well-known EQUI (equi-partitioning) processor allocation policy [16, 15], which shares the total number of processors equally among all active jobs at any time. We call our proposed algorithms U-EQUI (for Uniform-speed EQUI) and N-EQUI (for Nonuniform-speed EQUI), respectively. Both algorithms extend the natural speed scaling function [5, 24] used on a single processor, where the power consumption at any time is set proportionally to the number of active jobs. In the case of U-EQUI and N-EQUI, the power consumption is set to either this natural power or the given power budget, whichever is lower, and it is also distributed equally among all active jobs. However, this scheme can lead to a large amount of energy waste when jobs have relatively low parallelism. U-EQUI tackles this problem by reducing the processor allocations such that it maintains an optimal balance between the speedup and the energy consumption of the jobs. N-EQUI, on the other hand, takes advantage of the maximum utilization policy and distributes the power nonuniformly among the allocated processors of a job based on a scaled version of harmonic series.

Our contributions include the competitive analysis for both algorithms as well as their empirical evaluations. Specifically, we show that U-EQUI is $O(P^{(\alpha-1)/\alpha^2})$-competitive and N-EQUI is $O(\sqrt[\alpha]{n \ln P/\overline{m}})$-competitive with respect to flow time plus energy for any batched parallel job set, where P is the total number of processors, n is the total number of jobs, and $\overline{m}$ is a function of the capped power (defined in detail in Section 2). The ratio of U-EQUI also matches the lower bound for any uniform-speed non-clairvoyant algorithm (up to constant factor). Our simulation results on synthetic workloads show that U-EQUI and N-EQUI perform better than a straightforward fixed-speed EQUI strategy. Furthermore, moderate power constraint has little impact on the performances of our algorithms while reducing the power budget below 50% of the full power can lead to significant performance degradations.

The rest of this paper is organized as follows. Section 2 defines the job model and the scheduling model. Section 3 introduces two lower bounds for flow time plus energy on batched parallel jobs. Section 4 outlines the local competitive analysis framework used in this paper, followed by the presentation of U-EQUI and N-EQUI algorithms and their analysis in Section 5. Our simulation results are presented in Section 6. Section 7 discusses some related work, and finally, Section 8 gives the concluding remark.

2. PRELIMINARIES

We consider a set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n jobs to be scheduled on P processors with power budget U. Each job J_i in the job set, where $1 \le i \le n$, has time-varying parallelism modeled by multiple phases of speedup functions. Specifically, the job J_i contains q_i phases $\langle J_i^1, J_i^2, \ldots, J_i^{q_i} \rangle$, and each phase J_i^q, where $1 \le q \le q_i$, has an amount of *work* w_i^q, and a linear *speedup function* Γ_i^q up to a certain degree of parallelism h_i^q, where $h_i^q \ge 1$. The phase is *fully parallelizable* if $h_i^q = \infty$, and it is *sequential* if $h_i^q = 1$. The total work of job J_i is denoted by $w_i = \sum_{q=1}^{q_i} w_i^q$. At any time t, suppose that job J_i is allotted $a_i(t)$ processors. In this

paper, we do not restrict the allotted processors to have the same speed. Hence, let the jth allotted processor have speed $s_{ij}(t)$, where $1 \le j \le a_i(t)$, and without loss of generality, we assume $s_{i1}(t) \ge s_{i2}(t) \ge \ldots \ge s_{ia_i(t)}(t)$.

At any time t when job J_i is in its q-th phase, the execution of the job is based on the *maximum utilization policy* [21, 6], where faster processors are always utilized before slower ones. Hence, with ideal load balancing, the effective speedup or execution rate of the job at time t is given by

$$\Gamma_i^q(a_i(t)) = \begin{cases} \sum_{j=1}^{a_i(t)} s_{ij}(t) & \text{if } a_i(t) \le h_i^q, \\ \sum_{j=1}^{h_i^q} s_{ij}(t) & \text{if } a_i(t) > h_i^q, \end{cases}$$

and the power consumption of job J_i at time t is given by $u_i(t) = \sum_{j=1}^{a_i(t)} s_{ij}(t)^\alpha$.

A scheduling algorithm ALG for any set $\mathcal{J}$ of jobs specifies the number $a_i(t)$ of processors allocated to each job J_i at any time t, as well as the speed of each allocated processor. In order for the schedule to be valid, we require that at any time t the total processor allocation is not more than the total number of processors, i.e., $\sum_{i=1}^n a_i(t) \le P$, and the total power consumption is not more than the total power budget, i.e., $\sum_{i=1}^n u_i(t) \le U$. Let c_i^q denote the completion time of the q-th phase of job J_i, and let $c_i = c_i^{q_i}$ denote the *completion time* of job J_i. We also require that a valid schedule must complete all jobs in finite amount of time and cannot begin to execute a phase of a job unless it has completed all its previous phases, i.e., $0 = c_i^0 < c_i^1 < \ldots < c_i^{q_i} < \infty$, and $\int_{c_i^{q-1}}^{c_i^q} \Gamma_i^q(a_i(t))dt = w_i^q$ for all $1 \le q \le q_i$.

The job J_i is said to be *active* at time t if it is released but not completed at t, i.e., $0 < t < c_i$. The *flow time* f_i of the job is simply the completion time of the job in batched setting, i.e., $f_i = c_i$, and the *energy* e_i of the job is the total power consumed by the job integrated over time, i.e., $e_i = \int_0^\infty u_i(t)dt$. The *total flow time* $F(\mathcal{J})$ of the entire job set $\mathcal{J}$ is thus given by $F(\mathcal{J}) = \sum_{i=1}^n f_i$, or alternatively can be expressed as $F(\mathcal{J}) = \int_0^\infty n(t)dt$, where $n(t)$ is the number of active jobs at time t. The *total energy* is $E(\mathcal{J}) = \sum_{i=1}^n e_i$, or alternatively $E(\mathcal{J}) = \int_0^\infty u(t)dt$, where $u(t) = \sum_{i=1}^{n(t)} u_i(t)$ is the total power consumption at time t. For convenience, let us set the total power budget to $U = \frac{m}{\alpha - 1}$, where $m > 0$, and set $U_i = \min\{U, \frac{i}{\alpha-1}\}$. Let us also define the *capped power* $\overline{m}(t)$ at time t to be

$$\overline{m}(t) = \begin{cases} 1 & \text{if } m < 1, \\ m & \text{if } 1 \le m \le n(t), \\ n(t) & \text{if } m > n(t), \end{cases}$$

and let $\overline{m} = \overline{m}(0)$.

Our objective is to minimize the total flow time plus energy, $G(\mathcal{J}) = F(\mathcal{J}) + E(\mathcal{J})$, for which we use *competitive analysis* [7]. A scheduling algorithm ALG is said to be c-competitive if its total flow time plus energy is not more than c times that of an optimal schedule, i.e., $G_{\text{ALG}}(\mathcal{J}) \le c \cdot G_{\text{OPT}}(\mathcal{J})$ for any job set $\mathcal{J}$, where $G_{\text{OPT}}(\mathcal{J})$ is the total flow time plus energy of the optimal scheduler.

3. TWO LOWER BOUNDS ON TOTAL FLOW TIME PLUS ENERGY

In this section, we present two lower bounds on the total flow time plus energy for any batched parallel job set. These two lower bounds resemble the height bound and the squashed area bound [31, 13, 16, 18] in the literature for the flow time of batched parallel jobs, and they extend the lower bounds derived by Lam et al.[24, 23] for the flow time plus energy of sequential jobs.

Without loss of generality, we assume that the jobs in $\mathcal{J}$ are sorted according to their work such that $w_1 \ge w_2 \ge \ldots \ge w_n$. Also define $\overline{h}_i^q = \min\{P, h_i^q\}$ for each phase J_i^q of job J_i. The derivation of the lower bounds relies on the following property of the optimal scheduler.

PROPERTY 1. *At any time t, suppose that* OPT *is executing phase J_i^q of job J_i, then* OPT *allocates $a_i(t) \le \overline{h}_i^q$ processors of the same speed to job J_i.*

PROOF. It can be readily seen that if OPT allocates more processors than $\overline{h}_i^q$, then it will waste more energy without improving the flow time. Suppose that OPT does not set the same speed for the $a_i(t)$ allocated processors, then averaging their speed can result in the same execution rate at time t, and hence the same overall flow time, but less energy being consumed by the convexity of the power function. □

LEMMA 2. *The optimal total flow time plus energy satisfies $G_{\text{OPT}}(\mathcal{J}) \ge \max\{G_1(\mathcal{J}), G_2(\mathcal{J})\}$ for any set $\mathcal{J}$ of batched parallel jobs, where*

$$G_1(\mathcal{J}) = \frac{1+U_1}{U_1^{1/\alpha}} \sum_{i=1}^n \sum_{q=1}^{q_i} \frac{w_i^q}{(\overline{h}_i^q)^{1-1/\alpha}},$$

$$G_2(\mathcal{J}) = \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^n \frac{i+U_i}{U_i^{1/\alpha}} \cdot w_i.$$

PROOF. Suppose that in any infinitesimal interval of time $[t, t+\Delta t]$, where OPT does not change the number of processors and their speed allocated to job J_i, the job is currently executing its q-th phase J_i^q. According to Property 1, OPT allocates $a_i \le \overline{h}_i^q$ processors of the same speed, say s_i, to the job. The work done for the job during the interval is therefore $\Delta w_i = a_i s_i \Delta t$, and the energy consumed is $a_i s_i^\alpha \Delta t$. Hence, the flow time plus energy for the job during this interval is given by $\frac{\Delta w_i}{a_i s_i} + \frac{\Delta w_i}{a_i s_i} a_i s_i^\alpha$, which when minimized, subject to the total power constraint $a_i s_i^\alpha \le U$, gives $\frac{1+U_1}{U_1^{1/\alpha}} \cdot \frac{\Delta w_i}{a_i^{1-1/\alpha}} \ge \frac{1+U_1}{U_1^{1/\alpha}} \cdot \frac{\Delta w_i}{(\overline{h}_i^q)^{1-1/\alpha}}$. Generalizing the argument to all jobs at all time gives the first lower bound. Note that this first lower bound applies to non-batched parallel job sets as well.

For the second lower bound, define $\mathcal{J}' = \{J_1', J_2', \ldots, J_n'\}$ to be another set of jobs such that each job J_i' has a single fully parallelizable phase with work $w_i' = w_i$. The optimal schedule for $\mathcal{J}$ is certainly a valid schedule for $\mathcal{J}'$ as well. Therefore, we have $G_{\text{OPT}}(\mathcal{J}) \ge G_{\text{OPT}}(\mathcal{J}')$. As with scheduling for sequential jobs on single processor systems, OPT schedules $\mathcal{J}'$ using SPT (shortest processing time) policy, since otherwise, the total flow time can be reduced by swapping jobs without affecting the energy. Applying the same argument for the proof of the first lower bound to scheduling job J_i' by noting that the flow time contribution now becomes $\Delta t \cdot i$, we get the second lower bound. □

4. LOCAL COMPETITIVE ARGUMENT

To prove that an online algorithm is c-competitive using *local competitive argument* [25], we will show that at any point in time, its performance is no more than c times that of the optimal. For most scheduling problems, especially with

speed scaling, local competitiveness is usually not achievable by directly comparing the number of active jobs and the power consumption of the online algorithm with those of the optimal scheduler [5]. However, for scheduling batched parallel jobs with more than one clearly defined lower bounds, as can also be observed in the scheduling literature without power considerations [13, 16, 18], the competitive ratio of the algorithm can be obtained by a bounding its local performance in terms of the changes of the lower bounds.

Let ALG be any online scheduling algorithm. For any job $J_i \in \mathcal{J}$, let us define $J_i(\overleftarrow{t})$ to be the portion of job J_i executed by ALG before time t. Specifically, $J_i(\overleftarrow{t})$ consists of the first $q-1$ phases $\langle J_i^1, J_i^2, \ldots, J_i^{q-1}\rangle$ of job J_i, where $c_i^{q-1} < t \le c_i^q$, and if $q \le q_i$, followed by another phase with linear speedup function Γ_i^q up to h_i^q and work $\int_{c_i^{q-1}}^{t} \Gamma_i^q(a_i(t))dt$. Let us also define $\mathcal{J}(\overleftarrow{t})$ to be $\mathcal{J}(\overleftarrow{t}) = \{J_1(\overleftarrow{t}), J_2(\overleftarrow{t}), \ldots, J_n(\overleftarrow{t})\}$. To schedule any job set $\mathcal{J}$ for the total flow time plus energy, we show that at any time t, ALG satisfies

$$\dot{G}_{\text{ALG}}(\mathcal{J}(t)) \le c_1 \cdot \dot{G}_1(\mathcal{J}(t)) + c_2 \cdot \dot{G}_2(\mathcal{J}(t)), \qquad (1)$$

where $\dot{G}_{\text{ALG}}(\mathcal{J}(t)) = \frac{G_{\text{ALG}}(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_{\text{ALG}}(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ denotes the rate of change (increase) for flow time plus energy incurred by the scheduling of ALG at time t, where Δt is an infinitesimal interval of time. Similarly, we have $\dot{G}_1(\mathcal{J}(t)) = \frac{G_1(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_1(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$ and $\dot{G}_2(\mathcal{J}(t)) = \frac{G_2(\mathcal{J}(\overleftarrow{t+\Delta t})) - G_2(\mathcal{J}(\overleftarrow{t}))}{\Delta t}$, which denote respectively the rates of change for the two lower bounds given in Lemma 2. Note that the changes for the two lower bounds are between job set $\mathcal{J}(\overleftarrow{t+\Delta t})$ and job set $\mathcal{J}(\overleftarrow{t})$, whose difference lies only in the parts that are executed by the online algorithm ALG over time interval Δt. Now, integrating Inequality (1) over time, we get

$$G_{\text{ALG}}(\mathcal{J}) \le c_1 \cdot G_1(\mathcal{J}) + c_2 \cdot G_2(\mathcal{J}),$$

which implies that ALG is (c_1+c_2)-competitive with respect to the total flow time plus energy.

In the infinitesimal interval of time Δt, we can assume that no job completes or makes a transition from one phase to the next. We can also assume that the processor and speed allocations by ALG do not change in this interval. The rate of increase for flow plus energy at time t is thus given by

$$\dot{G}_{\text{ALG}}(\mathcal{J}(t)) = n(t) + \sum_{i=1}^{n(t)} \sum_{j=1}^{a_i(t)} s_{ij}(t)^\alpha.$$

The rate of change for G_1 due to the scheduling of ALG at time t is

$$\dot{G}_1(\mathcal{J}(t)) = \frac{1+U_1}{U_1^{1/\alpha}} \sum_{i=1}^{n(t)} \frac{\Gamma_i^q(a_i(t))}{(h_i^q)^{1-1/\alpha}}.$$

Since $\frac{i+U_i}{U_i^{1/\alpha}}$ is an increasing function of i, the rate of change for G_2 is minimized when the rates of execution for the active jobs at time t are sorted in non-decreasing order, that is, $\Gamma_1^q(a_1(t)) \ge \Gamma_2^q(a_2(t)) \ge \ldots \ge \Gamma_{n(t)}^q(a_{n(t)}(t))$, and we have

$$\dot{G}_2(\mathcal{J}(t)) \ge \frac{1}{P^{1-1/\alpha}} \sum_{i=1}^{n(t)} \frac{i+U_i}{U_i^{1/\alpha}} \cdot \Gamma_i^q(a_i(t)).$$

By evaluating the rates of execution for a particular algorithm ALG, we can get the rates of change for G_1 and G_2 at any time t. Together with the rate of increase for flow time plus energy, the competitive ratio of ALG can be obtained by setting the minimum values for c_1 and c_2 such that Inequality (1) is satisfied.

5. TWO EQUI-BASED NON-CLAIRVOYANT ALGORITHMS

In this section, we propose two online non-clairvoyant scheduling algorithms based on the well-known EQUI (equi-partitioning) [16, 15] algorithm, which partitions the total number of processors equally among all active jobs at any time. The two algorithms are power-aware variants of EQUI, each augmented with a speed scaling function to assign the speed of the allocated processors. The first algorithm, called U-EQUI (for Uniform-speed EQUI), uses a uniform-speed scaling function, and we show that it achieves a competitive ratio of $O(P^{(\alpha-1)/\alpha^2})$, which matches the lower bound for any uniform-speed non-clairvoyant algorithm (up to constant factor). The second algorithm, called N-EQUI (for Nonuniform-speed EQUI), uses a nonuniform-speed scaling function and achieves a competitive ratio of $O(\sqrt[\alpha]{n \ln P/\overline{m}})$, which improves upon the asymptotic performance of U-EQUI when given sufficient power.

In this paper, we restrict ourselves to the case where the total number of jobs is no more than the total number of processors, i.e., $n \le P$. Hence, each active job can get at least one processor at any time. In addition, we assume that the processor allocations are always integral.[1] Let us define the following notations. An active job J_i is said to be *satisfied* at time t if its processor allocation is at least the parallelism h_i^q of the phase J_i^q in which it is currently executing, i.e., $a_i(t) \ge h_i^q$. Otherwise, the job is *deprived* if $a_i(t) < h_i^q$. Let $\mathcal{J}_S(t)$ denote the set of satisfied jobs at time t, and let $\mathcal{J}_D(t)$ denote the set of deprived jobs at t. Furthermore, we define the *deprived ratio* $x(t)$ at any time t to be $x(t) = |\mathcal{J}_D(t)|/n(t)$, and obviously, we have $0 \le x(t) \le 1$. We will present U-EQUI and N-EQUI algorithms and their analysis in the following two subsections. In Section 5.3, we give brief discussions on their performances in practice.

5.1 U-EQUI

U-EQUI algorithm uses a variant of EQUI to allot processors to active jobs at any time, and assigns the same speed to all allocated processors. Specifically, upon the arrival of all jobs at time 0 or the completion of any active job at time t, U-EQUI does the following:

(1) Assigns $a_i(t) = \frac{P^\lambda}{n(t)}$ processors to each active job $J_i \in \mathcal{J}(t)$, where $0 < \lambda \le 1$;

(2) Sets the speed of each processor allocated to job J_i to $s_i(t) = \left(\frac{U_{n(t)}}{n(t)a_i(t)}\right)^{1/\alpha}$.

We denote the algorithm that chooses a particular value of λ as U-EQUI$_\lambda$, where λ is the *allocation parameter*. Note that for $\lambda < 1$, the number of processors allocated to each active job by U-EQUI$_\lambda$ is less than that allocated by EQUI. The reason for this more conservative strategy is mainly for reducing the energy consumption of the algorithm, and the optimal value of λ will be given later in this subsection. The

[1] In case that the processor allocation a to a job J is non-integral, then round it to $\lceil a \rceil$ or $\lfloor a \rfloor$ will affect the processor allocation by at most a factor of two, and the competitive ratio by at most a constant factor.

following lemma uses the set of satisfied jobs and the set of deprived jobs to bound the rates of change for G_1 and G_2 respectively under U-EQUI$_\lambda$.

LEMMA 3. *Under* U-EQUI$_\lambda$*, the rates of change for* G_1 *and* G_2 *at any time* t *satisfy*
(1) $\dot{G}_1(\mathcal{J}(t)) \geq \frac{1}{P^{\lambda/\alpha}}(1-x(t))n(t)$;
(2) $\dot{G}_2(\mathcal{J}(t)) \geq \frac{1}{2P^{(1-\lambda)(\alpha-1)/\alpha}}x(t)^2 n(t)$.

PROOF. (1). For any satisfied job $J_i \in \mathcal{J}_S(t)$, its execution rate at time t is given by $\Gamma_i^q(a_i(t)) = h_i^q s_i(t) = h_i^q \left(\frac{U_{n(t)}}{n(t)a_i(t)}\right)^{1/\alpha}$. Since $h_i^q \geq 1$, hence, we have

$$\begin{aligned}\dot{G}_1(\mathcal{J}(t)) &\geq \frac{1+U_1}{U_1^{1/\alpha}} \sum_{i=1}^{|\mathcal{J}_S(t)|} \left(\frac{h_i^q U_{n(t)}}{n(t)a_i(t)}\right)^{1/\alpha} \\ &\geq \left(\frac{\overline{m}(t)}{n(t)a_i(t)}\right)^{1/\alpha}(1-x(t))n(t) \\ &\geq \frac{1}{P^{\lambda/\alpha}}(1-x(t))n(t).\end{aligned}$$

(2). For any deprived job $J_i \in \mathcal{J}_D(t)$, its execution rate at time t is $\Gamma_i^q(a_i(t)) = a_i(t)s_i(t) = \left(\frac{U_{n(t)}}{n(t)}\right)^{1/\alpha} a_i(t)^{1-1/\alpha}$. Hence, we have

$$\dot{G}_2(\mathcal{J}(t)) \geq \frac{1}{P^{1-1/\alpha}}\left(\frac{U_{n(t)}}{n(t)}\right)^{1/\alpha} a_i(t)^{1-1/\alpha} \sum_{i=1}^{|\mathcal{J}_D|} \frac{i+U_i}{U_i^{1/\alpha}}.$$

Let $k = \lfloor m \rfloor$, and assume $1 \leq k \leq |\mathcal{J}_D(t)|$. Then, we have $U_{n(t)} = \frac{m}{\alpha-1}$ and

$$\begin{aligned}&\sum_{i=1}^{|\mathcal{J}_D(t)|} \frac{i+U_i}{U_i^{1/\alpha}} \\ &= \sum_{i=1}^{k} \frac{i+\frac{i}{\alpha-1}}{\left(\frac{i}{\alpha-1}\right)^{1/\alpha}} + \sum_{i=k+1}^{|\mathcal{J}_D(t)|} \frac{i+\frac{m}{\alpha-1}}{\left(\frac{m}{\alpha-1}\right)^{1/\alpha}} \\ &\geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \sum_{i=1}^{k} i^{1-1/\alpha} + \left(\frac{\alpha-1}{m}\right)^{1/\alpha} \sum_{i=k+1}^{|\mathcal{J}_D(t)|} i \\ &\geq \frac{\alpha k^{2-1/\alpha}}{2(\alpha-1)^{1-1/\alpha}} + \left(\frac{\alpha-1}{m}\right)^{1/\alpha} \frac{|\mathcal{J}_D(t)|^2 - k^2}{2}. \quad (2)\end{aligned}$$

Substituting Inequality (2) into $\dot{G}_2(\mathcal{J}(t))$ and simplifying, we get

$$\dot{G}_2(\mathcal{J}(t)) \geq \left(\frac{1}{P}\right)^{(1-\lambda)(\alpha-1)/\alpha} \frac{x(t)^2}{2} n(t). \quad (3)$$

The rate of change for G_2 for the other two cases when $k < 1$ or $k > |\mathcal{J}_D(t)|$ turns out to satisfy Inequality (3) as well and can be easily verified. □

Now, let us set $\lambda_1 = (\alpha-1)/\alpha$. Given the rates of change for G_1 and G_2, the following theorem bounds the competitive ratio of U-EQUI$_{\lambda_1}$ for batched parallel job sets.

THEOREM 4. *The* U-EQUI$_{\lambda_1}$ *algorithm, where* $\lambda_1 = (\alpha-1)/\alpha$*, is* $O(P^{(\alpha-1)/\alpha^2})$*-competitive with respect to the total flow time plus energy for any batched parallel job set.*

PROOF. Based on the local competitiveness analysis outlined in Section 4, we show that at any time t, U-EQUI$_{\lambda_1}$ satisfies

$$\dot{G}_{\text{U-EQUI}_{\lambda_1}}(\mathcal{J}(t)) \leq c_1 \cdot \dot{G}_1(\mathcal{J}(t)) + c_2 \cdot \dot{G}_2(\mathcal{J}(t)), \quad (4)$$

where $c_1 = c_2 = \frac{2\alpha}{\alpha-1}P^{(\alpha-1)/\alpha^2}$. Hence, we get the specified competitive ratio.

The rate of increase for the flow time plus energy is given by $\dot{G}_{\text{U-EQUI}_{\lambda_1}}(\mathcal{J}(t)) = n(t) + \sum_{i=1}^{n(t)} a_i(t)s_i(t)^\alpha = n(t) + U_{n(t)} \leq n(t) + \frac{n(t)}{\alpha-1} = \frac{\alpha}{\alpha-1}n(t)$. Substituting it and the rates of change for G_1 and G_2 into Inequality (4) and simplifying, we get $(x(t)-1)^2 \geq 0$, which holds for all values of $x(t)$. □

Remarks. It can be seen from the analysis of Lemma 3 and Theorem 4 that U-EQUI$_\lambda$ is $O(P^{\max\{\lambda/\alpha,(1-\lambda)(\alpha-1)/\alpha\}})$-competitive for any value λ. Hence, the choice of $\lambda_1 = (\alpha-1)/\alpha^2$ gives the best competitive ratio for U-EQUI$_\lambda$ algorithm. Larger values of λ can lead to waste of energy when the parallelism of the job is relatively low, while smaller values of λ do not speed up the job enough when it has ample parallelism. In fact, as long as a deterministic non-clairvoyant algorithm uses uniform-speed scaling function, no further improvements on the competitive ratio can be attained (up to constant factor), as the following theorem gives a matching lower bound.

THEOREM 5. *Any deterministic non-clairvoyant algorithm that uses uniform-speed scaling is* $\Omega(P^{(\alpha-1)/\alpha^2})$*-competitive with respect to total flow time plus energy on parallel jobs.*

PROOF. The proof is based on a simple construction with a single job having constant parallelism $h \leq P$ and work $w > 0$. Without loss of generality, we can assume that any uniform-speed non-clairvoyant algorithm ALG allocates a fixed number a of processors with speed s to the job throughout its execution. The adversary, upon knowing the allocation a, can set the parallelism h of the job in such a way that it forces algorithm ALG to have a large competitive ratio.

The optimal algorithm on the other hand always allocates h processors to the job, each with speed $\left(\frac{1}{(\alpha-1)h}\right)^{1/\alpha}$. Thus, it has total flow time plus energy $G_{\text{OPT}} = \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{h^{1-1/\alpha}}$. We consider two cases depending on the allocation of ALG.

Case 1: Suppose that $a \geq P^{1-1/\alpha}$. Then setting $h = 1$ gives the total flow time plus energy $G_{\text{ALG}} = \frac{w}{s} + \frac{w}{s}as^\alpha \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot wa^{1/\alpha}$. The competitive ratio in this case is given by $G_{\text{ALG}}/G_{\text{OPT}} \geq a^{1/\alpha} \geq P^{(\alpha-1)/\alpha^2}$.

Case 2: Suppose that $a < P^{1-1/\alpha}$. Then setting $h = P$ gives the total flow time plus energy $G_{\text{ALG}} = \frac{w}{as} + \frac{w}{as}as^\alpha \geq \frac{\alpha}{(\alpha-1)^{1-1/\alpha}} \cdot \frac{w}{a^{1-1/\alpha}}$. The competitive ratio in this case is given by $G_{\text{ALG}}/G_{\text{OPT}} \geq (h/a)^{1-1/\alpha} > P^{(\alpha-1)/\alpha^2}$. □

Remarks. Despite the fact that the best competitive ratio for U-EQUI$_\lambda$ algorithm is obtained at $\lambda = (\alpha-1)/\alpha$, we can get different values of c_1 and c_2 with other choices of λ. This has an implication on the practical performance of U-EQUI$_\lambda$ under different workload conditions. More practical considerations for the U-EQUI$_\lambda$ algorithm are presented in Section 5.3 and in our simulation study in Section 6.

5.2 N-EQUI

We observe from the analysis of U-EQUI$_\lambda$ that for nonclairvoyant algorithms, it is important to maintain a balanced design that considers both jobs with high parallelism and jobs with low parallelism. Although U-EQUI$_{\lambda_1}$ provides the optimal balance between the two groups among all uniform-speed algorithms, the lower bound (Theorem 5) on the competitive ratio motivates us to also examine nonclairvoyant algorithms with nonuniform-speed scaling functions. In this subsection, we show that nonuniform-speed scaling indeed helps by presenting one specific algorithm, N-EQUI, with improved competitive ratio.

N-EQUI algorithm uses EQUI as the processor allocation policy, and a scaled version of harmonic series as the power distribution policy. Specifically, at any time t, let $a = \frac{P}{n(t)}$, and let H_a denote the a-th Harmonic number, that is, $H_a = \sum_{k=1}^{a} \frac{1}{k}$. The N-EQUI algorithm does the following:
(1) Assigns $a_i(t) = a$ processors to each active job $J_i \in \mathcal{J}(t)$;
(2) Sets the speed of the jth processor of job J_i to $s_{ij}(t) = \left(\frac{U_{n(t)}}{n(t)H_a \cdot j}\right)^{1/\alpha}$.

The following theorem bounds the performance of N-EQUI with respect to the flow time plus energy for batched parallel jobs.

THEOREM 6. *The* N-EQUI *algorithm is* $O(\sqrt[\alpha]{n \ln P/\overline{m}})$*-competitive with respect to the total flow time plus energy for any batched parallel job set.*

PROOF. The proof follows closely that of U-EQUI based on local competitiveness analysis. In particular, we show that at any time t, N-EQUI satisfies

$$\dot{G}_{\text{N-EQUI}}(\mathcal{J}(t)) \le c_1 \cdot \dot{G}_1(\mathcal{J}(t)) + c_2 \cdot \dot{G}_2(\mathcal{J}(t)), \qquad (5)$$

where $c_1 = \frac{2H_P^{1/\alpha}}{2^{1-1/\alpha}-1}\left(\frac{n}{\overline{m}}\right)^{1/\alpha}$ and $c_2 = \frac{2H_P^{1/\alpha}}{2^{1-1/\alpha}-1}$. Since it is well known that the harmonic number $H_P = \Theta(\ln P)$, the competitive ratio is given by the larger coefficient c_1, which is $O(\sqrt[\alpha]{n \ln P/\overline{m}})$ for constant values of α.

The rate of increase for the flow time plus energy at time t is given by $\dot{G}_{\text{N-EQUI}}(\mathcal{J}(t)) = n(t) + \sum_{i=1}^{n(t)} \sum_{j=1}^{a_i(t)} s_{ij}(t)^\alpha = n(t) + U_{n(t)} \le n(t) + \frac{n(t)}{\alpha-1} = \frac{\alpha}{\alpha-1} n(t)$. The rates of change for G_1 and G_2 are again based on the set of satisfied jobs and the set of deprived jobs respectively under N-EQUI.

For any satisfied job $J_i \in \mathcal{J}_S(t)$, its execution rate at time t is given by $\Gamma_i^q(a_i(t)) = \sum_{j=1}^{h_i^q} s_{ij}(t) = \sum_{j=1}^{h_i^q} \left(\frac{U_{n(t)}}{n(t)H_a \cdot j}\right)^{1/\alpha} \ge \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{(h_i^q+1)^{1-1/\alpha}-1}{1-1/\alpha} \ge \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha} (h_i^q)^{1-1/\alpha}$. The last inequality is because $\frac{(x+1)^{1-1/\alpha}-1}{x^{1-1/\alpha}}$ is increasing function of x for $x \ge 1$. Hence, we have

$$\begin{aligned}
&\dot{G}_1(\mathcal{J}(t)) \\
&\ge \frac{1+U_1}{U_1^{1/\alpha}} \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha} \sum_{i=1}^{|\mathcal{J}_S(t)|} \frac{(h_i^q)^{1-1/\alpha}}{(h_i^q)^{1-1/\alpha}} \\
&\ge \left(\frac{\overline{m}(t)}{n(t)H_a}\right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha}(1-x(t))n(t).
\end{aligned}$$

For any deprived job $J_i \in \mathcal{J}_D(t)$, its execution rate at time t is $\Gamma_i^q(a_i(t)) = \sum_{j=1}^{a} s_{ij}(t) = \sum_{j=1}^{a} \left(\frac{U_{n(t)}}{n(t)H_a \cdot j}\right)^{1/\alpha} \ge \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{(a+1)^{1-1/\alpha}-1}{1-1/\alpha} \ge \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha} \cdot a^{1-1/\alpha}$.

The rate of change of G_2 is then at least

$$\begin{aligned}
&\dot{G}_2(\mathcal{J}(t)) \\
&\ge \frac{1}{P^{1-1/\alpha}} \left(\frac{U_{n(t)}}{n(t)H_a}\right)^{1/\alpha} \frac{2^{1-1/\alpha}-1}{1-1/\alpha} \cdot a^{1-1/\alpha} \sum_{i=1}^{|\mathcal{J}_D|} \frac{i+U_i}{U_i^{1/\alpha}}.
\end{aligned}$$

Let $k = \lfloor m \rfloor$, and assume $1 \le k \le |\mathcal{J}_D(t)|$. Substituting Inequality (2) into $\dot{G}_2(\mathcal{J}(t))$ above and simplifying, we get $\dot{G}_2(\mathcal{J}(t)) \ge \frac{2^{1-1/\alpha}-1}{1-1/\alpha}\left(\frac{1}{H_a}\right)^{1/\alpha} \frac{x(t)^2}{2} n(t)$, which again holds for $k < 1$ or $k > |\mathcal{J}_D(t)|$ as can be easily verified. Inequality (5) is satisfied by substituting the rate of increase for flow time plus energy, as well as the rates of change for G_1 and G_2. □

Remarks. Theorem 6 suggests that when given sufficient power with respect to the total number of jobs, the competitive ratio of N-EQUI becomes $O(\sqrt[\alpha]{\ln P})$, which improves the asymptotic performance of U-EQUI$_{\lambda_1}$ algorithm.

5.3 Performance in Practice

In the preceding two subsections, we have analyzed the performances of U-EQUI and N-EQUI algorithms in terms of their competitive ratios. In this subsection, we briefly discuss their performances in practice.

From the analysis given previously, we can see that the competitive ratio of an algorithm can be decomposed into two parts c_1 and c_2, which are obtained by considering the set of satisfied jobs and the set of deprived jobs, respectively. In practice, the performance of the algorithm will to a large extent depend on the *dominant coefficient*, which corresponds to the dominating set of jobs. For instance, the competitive ratio of U-EQUI$_1$ algorithm is $O(P^{1/\alpha})$, which is obtained by the larger coefficient c_1 on the set of satisfied jobs. However, if the system is under heavy workloads when there is a large number of active jobs with ample parallelism, then there tends to be more deprived jobs than satisfied ones. In this case, the dominant coefficient actually tends towards c_2, which is a constant with respect to P. On the other hand, the coefficient c_2 for the U-EQUI$_{\lambda_1}$ algorithm, where $\lambda_1 = (\alpha-1)/\alpha$, is given by $O(P^{(\alpha-1)/\alpha^2})$. Hence, despite the fact that U-EQUI$_{\lambda_1}$ has better worst-case competitive ratio than U-EQUI$_1$, U-EQUI$_1$ should perform better than U-EQUI$_{\lambda_1}$ when the system is under relatively heavy workload.

In addition, the *local* performance of U-EQUI at any time t when the system is under light workload is also dependent upon the capped power at t as can be seen from the derivation of $\dot{G}_2$ in Lemma 3. While this factor can have an impact on the practical performance of U-EQUI, it is neglected in the worst-case analysis to obtain the *global* competitive ratio. In Section 6, we will evaluate the impact of these factors through simulations.

6. SIMULATIONS

We use simulations to evaluate the performances of our proposed algorithms. Our main objectives include demonstrating the effect of allocation factor λ on the performance of U-EQUI$_\lambda$, comparing our dynamic speed-scaling algorithms with EQUI-based fixed-speed strategy, and evaluating the impact of different power budgets on their performances.

6.1 Simulation Setup

In our simulations, we construct synthetic workloads using parallel jobs with varying degrees of parallelism. For each job, we generate its parallelism based on the model given in [14], which uses two parameters: the average parallelism A, and the variance of parallelism σ, to specify the parallelism characteristics of the job. Specifically, for each job, we choose its average parallelism A uniformly from $[1, 100]$ and select the variance σ uniformly in the interval $[0, 2]$. Moreover, in order to evaluate the robustness of our algorithms, we allow the parallelism for each phase of a job to follow different distributions, such as Uniform, Weibull, and Lognormal distributions. As the experimental results finally turn out, the performances of our algorithms are fairly insensitive to the distributions we choose.

We generate a wide range of workloads by varying the number of jobs in our simulations. Specifically, we conduct over 1000 sets of experiments, and the number of jobs in each experiment is uniformly selected from 1 to 100. The performance ratio of an algorithm in each experiment is computed by normalizing its flow time plus energy with respect to the maximum of the two lower bounds given in Section 3. For experiments that have the same number of jobs, hence with similar workloads, we take their average ratio to evaluate the performance of an algorithm for robustness. The number of processors in the system is set to be $P = 1000$, and the power parameter is set to be $\alpha = 3$. The scheduling overheads due to reallocations and speed scalings of the processors are ignored, since they happen infrequently only when a job completes its execution.

6.2 Simulation Results

(1) *Effect of λ for* U-EQUI$_\lambda$.

Our first set of simulations focuses on the effect of allocation factor λ on the performance of U-EQUI$_\lambda$ algorithm. As indicated in Section 5.3, larger values of λ give better performance with heavy system workload while smaller values of λ tend to perform better when the workload is light. We choose five values of λ uniformly between $1-1/\alpha$ and 1, and their performance ratios on different workloads are given in Figure 1. Clearly, the results match our analysis, and the crossover happens roughly when the number of jobs reaches 10 in our simulation. Because of the superior performance of U-EQUI$_1$ on the majority of our workloads, we use it to represent the class of U-EQUI algorithms in the subsequent experiments.

(2) *Comparison with fixed-speed strategy.*

We evaluate the performances of U-EQUI and N-EQUI algorithms by comparing them with a fixed-speed strategy, which uses EQUI as the processor allocation policy, and always assigns the same fixed speed to all processors at all time. To demonstrate the advantage of our speed-scaling algorithms, we conduct simulations for each set of jobs using the fixed-speed strategy with a wide range of speeds, and choose the best one to compare with U-EQUI and N-EQUI. Figure 2 shows the comparison results. As can be seen, N-EQUI significantly outperforms the best fixed-speed strategy under light workloads, and U-EQUI performs slightly better than the best fixed-speed strategy over the entire workload range. N-EQUI and U-EQUI have similar performances under heavy system workloads, when the ratio of U-EQUI becomes dominated by the constant coefficient, and the ratio of N-EQUI is small enough by our choice of the parameters to match that of U-EQUI. On average, N-EQUI improves the overall performance ratio of the best fixed-speed strategy by 15.6% and U-EQUI improves the performance ratio by 3.6%.

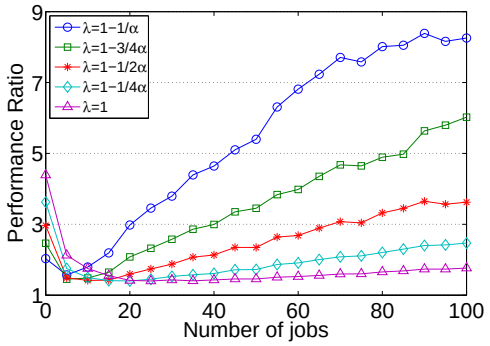

Figure 1: Effect of allocation factor λ on the performance of U-EQUI$_\lambda$.

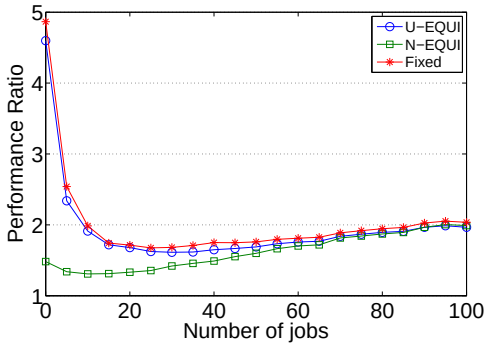

Figure 2: Comparison of U-EQUI and N-EQUI with the best fixed-speed strategy.

(3) *Impact of power budget.*

We study the effect of different power budget on the performance of U-EQUI and N-EQUI by changing the parameter m (i.e. $(\alpha - 1)$ times the power budget) with respect to the total number n of jobs for each job set. Specifically, we reduce m/n from 100% to 10%. Figure 3(a) and Figure 3(b) show the impact of power reduction on the performances of U-EQUI and N-EQUI. We can see that except for the very light and heavy workloads, where the impact of power reduction is relatively small, clear performance degradations can be observed for the wide range of medium workloads on both algorithms. The exception for the light workload follows the analysis in Section 5 that the performances of our algorithms degrade with reduced power budget until the parameter m is capped at 1. When the workload is heavy, on the other hand, deprived jobs will dominate so that reducing the power budget will have little effect on the performance. Figure 4 shows the performance degradation of N-EQUI in terms of the increase in percentage of the average performance ratio when reducing the power budget. (Almost identical results are observed for U-EQUI.) Clearly, the performance is not significantly affected by moderate power reduction, but it degrades quickly after the power budget drops below 50%. The performance degradation reaches 36% when the power is reduced to only one tenth of the full power.

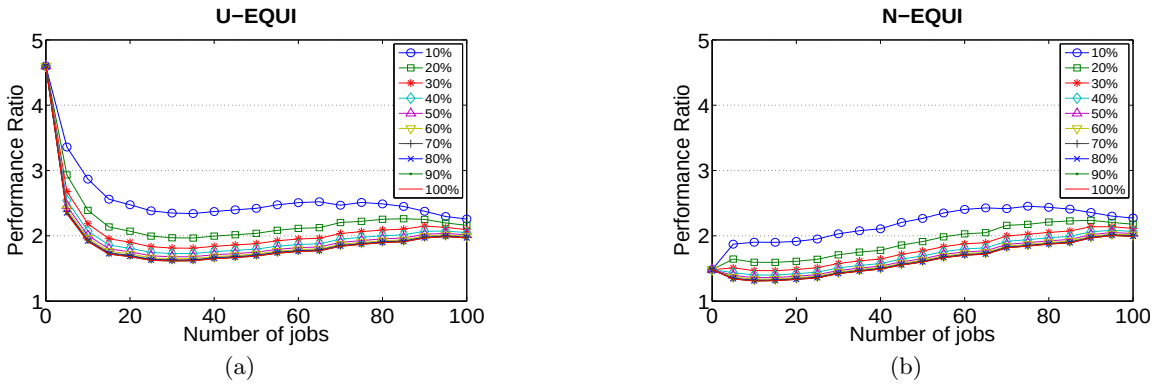

Figure 3: Impact of different power budgets on the performances of U-EQUI and N-EQUI.

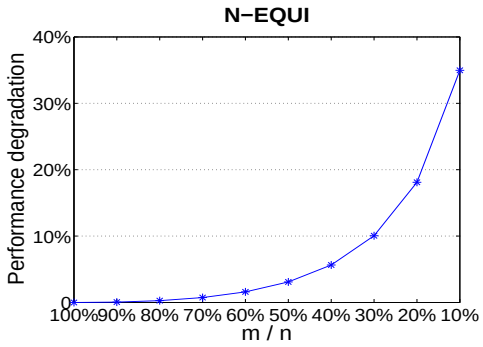

Figure 4: Performance degradation of N-EQUI when reducing the power budget.

7. RELATED WORK

There are many prior studies that focus on power-efficient scheduling with dynamic speed scaling. One line of research considers scheduling with deadline constraints. Yao et al. [32] initially considered a minimum energy schedule and average rate heuristic for clairvoyant job scheduling with deadlines. They gave an optimal offline algorithm and an online algorithm whose competitive ratio is between α^α and $2^\alpha\alpha^\alpha$ ($\alpha \geq 2$). Bansal et al. [4] further improved the online scheduling result with a new algorithm that is $2(\alpha/(\alpha-1))^\alpha e^\alpha$-competitive. Some other results for scheduling with deadlines can be found in the survey [19].

Another line of research studies speed scaling to minimize the total flow time plus energy. Almost all results along this line use the natural speed scaling function that sets the processor speed at any time t to $n(t)^{1/\alpha}$, where $n(t)$ is the number of active jobs at t. Albers and Fujiwara [1] first considered this problem for unit-size jobs and proposed an $O\left(\left(\frac{3+\sqrt{5}}{2}\right)^\alpha\right)$-competitive algorithm. Bansal et al. [5] later improved its competitive ratio to 4. Moreover, they also considered a more general setting, where jobs have arbitrary sizes and arbitrary weights, and showed that HDF (highest density first) is $\max\{2, \frac{2(\alpha-1)}{\alpha-(\alpha-1)^{1-1/(\alpha-1)}}\}$-competitive for fractional weighted flow plus energy. The same results were achieved in [3] for one processor with bounded speed. Lam et al. [24] showed that SRPT (shortest remaining processing time) is $2/(1-\frac{\alpha-1}{a^{\alpha/(\alpha-1)}})$-competitive without speed constraint and $2(\alpha+1)/(\alpha-\frac{\alpha-1}{a^{\alpha/(\alpha-1)}})$-competitive with bounded speed. They also showed that the non-clairvoyant algorithm RR (round robin) achieves $(2-1/\alpha)$-competitive for batched jobs without speed bound and 2-competitive with speed bound. Recently, Chan et al. [10] showed that the non-clairvoyant algorithm LAPS (Latest Arrival Processor Sharing) is $O(\alpha^3)$-competitive for non-batched jobs.

All results above focused on uniprocessor scheduling. Attention has also been paid to speed scaling in multiprocessor settings. For real-time systems, Chen [11] proposed an energy efficient algorithm to schedule frame-based tasks over multiprocessors. Isci et al. [20] developed globally aware policies to dynamically tune the speeds of the processors under a chip-wide power constraint. Pruhs [28] considered minimizing the makespan for jobs with precedence constraints on multiprocessors with bounded energy. Bunde [9] gave an optimal offline algorithm for the makespan of equal-size jobs on multiprocessors with an energy constraint, and an arbitrarily good approximation algorithm for the total flow time. Albers et al. [2] studied offline scheduling for jobs with deadline constraints to minimize the total consumed energy on parallel processors. Lam et al. [23] studied online non-migratory scheduling for sequential jobs on multiprocessors while minimizing the total flow time plus energy.

For scheduling without power constraints, Edmonds et al. [16] considered non-clairvoyant scheduling on batched parallel jobs modeled by multiple phases of arbitrary speedup functions. They showed that EQUI is $(2+\sqrt{3})$-competitive with respect to the total flow time. Edmonds [15] also showed that for arbitrarily released parallel jobs, EQUI is $(2+\frac{4}{\epsilon})$-competitive with respect to the total flow time when given $(2+\epsilon)$ more processors than the optimal scheduler.

8. CONCLUDING REMARKS

In this paper, we have focused on non-clairvoyant speed scaling for batched parallel jobs to minimize the total flow time plus energy. Based on the EQUI processor allocation policy, we have proposed two algorithms, namely U-EQUI and N-EQUI, which use uniform-speed and nonuniform-speed scaling functions, respectively, and we have studied the performance of our algorithms using both theoretical analysis and empirical simulations.

Extending the analysis in this paper and in [13], we can show that combining DEQ (dynamic Equi-partitioning) processor allocation policy, which utilizes the jobs' instanta-

neous parallelism at any time to allocate processors, with the uniform-speed scaling function presented in this paper, $O(1)$-competitiveness can be achieved in terms of the total flow time plus energy for batched parallel jobs. This observation reveals the benefit of clairvoyance when it comes to scheduling for flow time plus energy with uniform-speed scaling, although the non-clairvoyant algorithm EQUI possesses $O(1)$-competitiveness for flow time alone [16]. For nonuniform-speed scaling, whether the current $O(\sqrt[\alpha]{\ln P})$-competitiveness of N-EQUI can be further improved is an open question, and it will also be of interest to provide a non-trivial lower bound in this setting.

9. REFERENCES

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

[2] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA*, pages 289–298, San Diego, CA, USA, 2007.

[3] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *ICALP*, pages 409–420, Reykjavik, Iceland, 2008.

[4] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS*, pages 520–529, Rome, Italy, 2004.

[5] N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. In *SODA*, pages 805–813, New Orleans, LA, USA, 2007.

[6] M. A. Bender and M. O. Rabin. Scheduling Cilk multithreaded computations on processors of different speeds. In *SPAA*, pages 13–21, Bar Harbor, ME, USA, 2000.

[7] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, New York, NY, USA, 1998.

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

[9] D. P. Bunde. Power-aware scheduling for makespan and flow. In *SPAA*, pages 190–196, Cambridge, MA, USA, 2006.

[10] H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *STACS*, pages 409–420, Freiburg, Germany, 2009.

[11] J.-J. Chen. Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *ICPP*, pages 13–20, Oslo, Norway, 2005.

[12] Z.-L. Chen. Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, 129:135–153, 2004.

[13] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive scheduling of parallel jobs on multiprocessors. In *SODA*, pages 159–167, Philadelphia, PA, USA, 1996.

[14] A. B. Downey. A parallel workload model and its implications for processor allocation. In *HPDC*, page 112, Portland, OR, USA, 1997.

[15] J. Edmonds. Scheduling in the dark. In *STOC*, pages 179–188, Atlanta, GA, USA, 1999.

[16] J. Edmonds, D. D. Chinn, T. Brecht, and X. Deng. Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics. In *STOC*, pages 120–129, El Paso, TX, USA, 1997.

[17] D. Grunwald, I. Charles B. Morrey, P. Levis, M. Neufeld, and K. I. Farkas. Policies for dynamic clock scheduling. In *OSDI*, pages 6–6, San Diego, CA, USA, 2000.

[18] Y. He, W.-J. Hsu, and C. E. Leiserson. Provably efficient two-level adaptive scheduling. In *JSSPP*, Saint-Malo, France, 2006.

[19] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[20] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, Orlando, FL, USA, 2006.

[21] J. M. Jaffe. An analysis of preemptive multiprocessor job scheduling. *Mathematics of Operations Research*, 5(3):415–421, 1980.

[22] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *HPCA*, pages 123–134, Salt Lake City, UT, USA, 2008.

[23] T.-W. Lam, L.-K. Lee, I. K. K. To, and P. W. H. Wong. Competitive non-migratory scheduling for flow time and energy. In *SPAA*, pages 256–264, Munich, Germany, 2008.

[24] T. W. Lam, L.-K. Lee, I. K.-K. To, and P. W. H. Wong. Speed scaling functions for flow time scheduling based on active job count. In *ESA*, pages 647–659, Karlsruhe, Germany, 2008.

[25] K. Pruhs. Competitive online scheduling for server systems. *ACM SIGMETRICS Performance Evaluation Review*, 34(4):52–58, 2007.

[26] K. Pruhs, E. Torong, and J. Sgall. Online scheduling. In handbook of scheduling: Algorithms, models, and performance analysis, chapter 15, CRC Press. 2004.

[27] K. Pruhs, P. Uthaisombut, and G. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4(3):1–17, 2008.

[28] K. Pruhs, R. van Stee, and P. Uthaisombut. Speed scaling of tasks with precedence constraints. In *WAOA*, pages 307–319, Mallorca, Spain, 2005.

[29] H. Sebastian and M. Diana. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *ISLPED*, pages 38–43, Portland, OR, USA, 2007.

[30] D. B. Shmoys and Éva Tardos. Scheduling unrelated machines with costs. In *SODA*, pages 448–454, Austin, Texas, USA, 1993.

[31] J. Turek, U. Schwiegelshohn, J. L. Wolf, and P. S. Yu. Scheduling parallel tasks to minimize average response time. In *SODA*, pages 112–121, Philadelphia, PA, USA, 1994.

[32] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, page 374, Milwaukee, WI, USA, 1995.