How to Schedule When You Have to Buy Your Energy

Kirk Pruhs[1,⋆] and Cliff Stein[2,⋆⋆]

[1] Computer Science Department. University of Pittsburgh, Pittsburgh, PA, USA
`kirk@cs.pitt.edu`

[2] Department of IEOR, Columbia University, New York, NY
`cliff@ieor.columbia.edu`

Abstract. We consider a situation where jobs arrive over time at a data center, consisting of identical speed-scalable processors. For each job, the scheduler knows how much income is lost as a function of how long the job is delayed. The scheduler also knows the fixed cost of a unit of energy. The online scheduler determines which jobs to run on which processors, and at what speed to run the processors. The scheduler's objective is to maximize profit, which is the income obtained from jobs minus the energy costs. We give a $(1+\epsilon)$-speed $O(1)$-competitive algorithm, and show that resource augmentation is necessary to achieve $O(1)$-competitiveness.

1 Introduction

As the price of server hardware has remained relatively stable, energy cost becomes one of the primary components in the total cost of ownership for computer server systems in data centers [11]. In fact, according to Dr. Eric Schmidt, CEO of Google:

> "What matters most to the computer designers at Google is not speed, but power, low power, because data centers can consume as much electricity as a city." [24].

A commonly used power management technique is speed scaling, changing the speed of the processor. As the dynamic power used by a processor is approximately the cube of the speed of the processor (this is called the cube-root rule for CMOS based processors [12,25]), even a modest reduction in speed can have a dramatic impact on power. Researchers at Google reported an approximately twenty percent energy savings from implementing the following reactive strategy: When the workload of a processor was light, the speed was scaled down, and when most processors were at maximum speed, some less time critical tasks were suspended, to be restarted when the system was not so heavily loaded [19].

⋆ Supported in part by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558, and an IBM Faculty Award.

⋆⋆ Supported in part by NSF grants CCF-0728733 and CCF-0915681.

Scheduling problems related to speed scaling and power management naturally have competing dual objectives: some quality of service (QoS) objective, and some power related objective. By now there are many tens of papers on speed scaling in the algorithmic literature (and many more in the general computer science literature). Roughly speaking, all of the formal problems considered in the algorithmic speed scaling literature fall into one of two categories. The first type of problem turns one of the QoS or power objectives into a constraint, and optimizes the other objective. An example is minimizing the total flow time subject to the constraint that the energy used doesn't exceed an energy bound representing the energy stored in a battery. The second type of problem optimizes the sum of the QoS and power objectives. An example of this type of problem is minimizing the sum of energy used and total flow time.

In this paper, we introduce a new class of speed scaling problems, which makes the monetary cost of energy more explicit, and we provide algorithmic results for a particular problem in this class. We assume that the scheduler is aware of the income obtainable from finishing particular jobs by particular times, and is aware of the cost of energy. We then naturally assume that the scheduler's goal is to maximize profit, which is the aggregate income minus the aggregate energy cost. One can easily formulate many natural problems within this framework, depending on how one formalizes income and energy costs (and also, of course, depending on the processor and job environments). Here we consider a rather general model for the income of jobs: We assume that there is an non-negative non-increasing income function $I_i(t)$ associated with each job i that specifies the income that is obtained if the job is finished at time t. And we consider the most natural and simple model for energy costs: We assume a fixed cost per unit of energy.

We now explain the job and machine environments that we consider in this paper. Jobs arrive over time at the data center consisting of m identical speed-scalable processors. There is an arbitrary power function $P(s)$ that specifies the power when a processor is run at speed s. Job i arrives at time r_i, with known work/size w_i, and known income function $I_i(t)$. The online scheduler must decide, at each time, which job to run on each processor, and at what speed to run each processor. We allow preemption and migration, that is, jobs can be suspended at any time, and restarted from the point of suspension at a later time, possibly on a different machine. Recall that our objective is to maximize the income from the scheduled jobs minus the total energy costs.

The standard measure of goodness for an online algorithm is competitiveness, which in this setting is, roughly speaking, the worst-case, over all possible inputs, of the relative error between the optimal profit and the profit achieved by the online algorithm. One generally seeks algorithms that are competitive, that is, where this relative error is bounded. The motivation for seeking competitive algorithms is that if the online algorithm achieves very little profit, then it must be because great profit was not achievable, and not because the algorithm was at fault.

1.1 Our Results

The most obvious first concern that arises when seeking a competitive algorithm for this problem is that one can imagine a situation where the online algorithm does not achieve a positive profit, even though a positive profit is achievable, immediately killing any hope of a competitive algorithm. We start by observing, in Section 3 that this situation cannot occur, that is, that there is a simple online algorithm that achieves a positive profit if it is possible to do so. Unfortunately, we show that, in some sense, this result is the best positive result possible for the competitive ratio by showing that the competitive ratio can not be bounded by any function of the number of jobs. The intuition behind this lower bound is that the online algorithm can be forced to run, and then later abandon, a high-cost low-profit job, thus wasting a lot of energy and money on this job.

Reflecting on this lower bound instance, one notices that if the processors used by the online algorithm were only slightly more energy efficient, then the online algorithm could be competitive on this instance. We show in Section 4 that this phenomenon holds for all instances. More specifically, we assume that the online algorithm has $(1+\epsilon)$-speed augmentation, which in this setting means that if a processor can run at power P and speed s, then the online algorithm can run the processor at power P and speed $(1+\epsilon)s$. We then give an online scheduling algorithm that we show is $O(\frac{1}{\epsilon^3})$-competitive in terms of profit. Using standard terminology [20,26,28,27], one could say that this algorithm is a scalable scheduling algorithm, that is, it is $(1+\epsilon)$-speed $O(1)$-competitive. Intuitively, scalable algorithms can handle almost the same load as optimal. For elaboration see [28,27].

We now give an overview of the development of our scalable algorithm. The first key idea is that of a critical speed function $\hat{s}_i(t)$, which, for job i, specifies the fastest speed that the adversary can run job i and still obtain a non-negative profit if the job completes at time t. When a job i is released, the online algorithm determines whether to admit the job, and if the job is admitted, determines a deadline d_i for the job. Whenever an admitted job i is run by the online algorithm, it will be run at speed slightly faster than the critical speed for its deadline, $\hat{s}_i(d_i)$. Fixing the speed for a job defines a density for the job, which is roughly the profit that will be obtained by the job if it is completed at its deadline divided by the time that the job must be run to be completed. Intuitively, the online algorithm always picks the highest density jobs to run. Also intuitively, when a job is released, the online algorithm sets the deadline to be the time where it will obtain maximum profit from this job, assuming that in the future no more jobs arrive and that the highest density jobs will be run at their critical speeds.

To show that the online algorithm is scalable, we show that the profit obtained by the online algorithm is a constant fraction of the profit of the jobs that the online algorithms admits, and that the profit of these admitted jobs is a constant fraction of the optimal profit. In order to accomplish the latter goal, we show that there is a near optimal schedule OPT', that, with modest speed augmentation, is $O(1)$-competitive in terms of profit with the optimal schedule, and OPT' has the property that it runs each job i at speed approximately equal to the critical speed of the job for the completion time $\hat{s}_i(C_i^O)$. of that job in the optimal schedule OPT. Thus OPT' is still nearly optimal, but is structurally similar to the online schedule in that jobs are run at their critical speeds. A priori, it is not clear that such a schedule OPT' exists since a job i may be run at very different speeds in OPT' and in the online schedule. In other words, $\hat{s}_i(C_i^O)$ and $\hat{s}_i(d_i)$ may be very different, since there is no reason that the completion time in the optimal schedule, C_i^O, and the deadline set by the online algorithm, d_i, need be similar.

Note that our algorithm can be converted into one that constructs non-migratory schedules using the results in [18].

The income model in our paper was considered in [7], a scalable algorithm for maximizing income on a single fixed speed processor was given. Our algorithm and analysis necessarily generalize the results in [7] as we have multiple processors instead of a single processor, our processors are speed scalable instead of fixed speed, and we have profit as the objective instead of income. The fact that the processors are speed scalable creates complications because the algorithm and analysis in [7] use the fact that the processing time for a job is fixed. The objective of profit also creates complications because the algorithm and analysis in [7] use the fact that income is monotonic in time, which isn't true for profit.

1.2 Related Results

The first theoretical study of speed scaling algorithms was in the seminal paper [31], which introduced the deadline feasibility framework, and considered minimizing energy usage on a single processor. This problem is the most investigated speed scaling problem in the literature [31,5,14,4,23,22,2,9]. In [31], the authors showed that the optimal offline schedule can be efficiently computed by a greedy algorithm. Several online algorithms for this problem have been proposed and analyzed, including AVR [31,5], OA [31,4], BKP [4], and qOA[9]. The competitive ratios of all of these algorithms grow in an unbounded manner as the power function becomes steeper, but the competitive ratio is $O(1)$ if the power function is bounded by a fixed polynomial. These results have been extended in several ways including to parallel processors[2], analyzing BKP with respect to temperature minimization[4], a variant in which one minimizes the recharge rate from a solar cell[3], and scalable algorithms for throughput optimization on a single speed scalable processor with a polynomial power function and an upper bound on the maximum speed[15,6].

Another class of problems considers flow time and energy. [29] give an offline algorithm to minimize flow time subject to an energy budget for a single processor. For a single processor [1] gives a competitive online algorithm for unit work jobs for the objective of total flow plus energy assuming the power function is a polynomial. Their results were extended to arbitrary sized and arbitrary weighted jobs in [10], and to arbitrary power functions in [8]. An extension to nonclairvoyant algorithms on a uniprocessor is given in [16]. An extension to nonclairvoyant algorithms on a multiprocessor is given in [17].

There have been several papers in the literature on speed scaling with the makespan objective [13,30].

The scalable algorithm for income in [7] generalizes a scalable algorithm given in [20] for the special case of maximizing the profit of jobs completed before their deadline (there are many papers on this problem in the literature).

2 Definitions

Jobs arrive over time at the data center. Job i (also referred to as job i)arrives at time r_i, with known work/size w_i, and known income function $I_i(t)$. The function $I_i(t)$, defined for all $t > 0$, gives the income earned if job i is completed at time t. We assume that the income function $I_i(t)$ is non-negative and non-increasing. We assume that the income goes to zero at the completion time approaches infinity, that is, $\lim_{t\to\infty} I_i(t) = 0$. And we assume that if a job doesn't complete, the income is zero.

We allow preemption, that is, jobs can be suspended at any time, and restarted at a later time, possibly on a different machine. However a schedule can run a job on at most one machine at a time. To formally define a schedule, one needs to describe, for each time on each machine, which job is run on that machine and the speed at which it is run. Due to the convexity of the power function, we need only consider schedules where each job is only run at a fixed speed. With this in mind, a schedule can be given by describing, for each job, the speed at which the job runs, and, for each time, which (if any) machine it is running on. A job i completes at the first time C_i where the speed that the job is run, integrated over all the times the that job is run, equals the work w_i. If a job does not complete, $C_i = \infty$.

We are also given an arbitrary power function $P(s)$ that, for any non-negative s, specifies the power used while running at speed s on each machine. As observed in [8], one can without loss of generality assume that P is convex and increasing. The energy used is the sum over the processors, of the integral over time of the power of that machine. The income associated with a job is $I_i(C_i)$. If a job does not complete, the income is zero. One can also compute the energy cost associated with running a job i at speed s_i as $E_i = P(s_i)w_i/s_i$, since it runs for w_i/s_i units of time, at power $P(s_i)$. The profit associated with job i is $p_i = I_i(C_i) - E_i$. Our objective is the total profit $\sum_i p_i$. We will sometimes superscript these quantities by A for the online algorithm, or by O for the optimal/adversary.

An online algorithm A is c-competitive if for all inputs the total profit achieved by A is at least $\frac{1}{c}$ of the maximum achievable profit. An online algorithm A is $(1+\epsilon)$-speed c-competitive if for all inputs the total profit achieved by A with power function $P(\frac{s}{1+\epsilon})$ is at least $\frac{1}{c}$ of the maximum achievable profit. More precisely, to have power function $P(\frac{s}{1+\epsilon})$ means that if the adversary can run at power P and speed s, then the online algorithm can run at power P and speed $(1+\epsilon)s$.

We ignore any issues about the time to access and solve equations involving the income and power functions. Presumably in most applications these functions will be compactly described, and sufficiently simple, that manipulating these functions, as required by our algorithms, should not be a significant issue.

3 No Resource Augmentation

Our first results concern the situation where we do not allow resource augmentation, that is the adversary and the on-line algorithm both have the same power function. We first note that it is always possible for an on-line algorithm to obtain positive profit if the adversary can also receive positive profit. In contrast, we then show that the competitive ratio can be arbitrarily large.

Lemma 1. *At the release time r_i for job i, an online algorithm can compute whether it is possible to achieve positive profit for job i.*

Proof. Assume job i is completed at time t. To minimize energy, job i should be run at constant speed $w_i/(t - r_i)$ during the time period $[r_i, t]$. Job i will then have energy cost $E_i(t) = P(s(t))w_i/(t - r_i)$. So the online algorithm need only determine whether there exists t such that $I_i(t) - E_i(t) > 0$.

Lemma 2. *If the adversary can obtain a positive profit, then so can the online algorithm.*

Proof. For each job i, when it arrives, the online algorithm computes whether it is possible to make a positive profit by running the job, using Lemma 1. For the first such job, the online algorithm runs the job and obtains a positive profit. If no such job arrives, then the adversary cannot obtain positive profit either.

We now show that it is possible that the competitive ratio can be arbitrary large.

Lemma 3. *The competitive ratio of any deterministic algorithm can not be bounded by any function of n, even if jobs have unit work.*

Proof. We consider a two job instance and a power function for which $P(1) = 1$ and $P(1/\epsilon) = L/\epsilon$, where $\epsilon > 0$ is a small number and $L > 1$. (For intuition, think of L as large.) So each processor has only two possible speeds, 1 and $1/\epsilon$. Job 1 has $r_1 = 0$, $w_1 = 1$ and $I_1(t) = 1 + \epsilon$ if $t \leq 1$ and $I_1(t) = 0$ for $t > 1$. When this job is released at time 0, by the reasoning of Lemma 2, the algorithm has to run this job immediately, or else the algorithm will have non-positive profit while the adversary could run the job for positive profit. Therefore, we assume that the algorithm runs job 1 at speed 1 (any other speed would incur a loss).

Job 2 has $r_2 = 1 - \epsilon$, $w_2 = 1$, and $I_2(t) = L + 1 - \epsilon$ if $t \leq 1$ and $I_2(t) = 0$ for $t > 1$. When job 2 is released, the algorithm can either run job 2 or not. If the algorithm does not switch to job 2 and finishes job 1, then it obtains $p_1 = (1+\epsilon) - P(1) \cdot 1 = \epsilon$. If it switches, it obtains profit $p_2 = (L+1-\epsilon) - P(1/\epsilon)\epsilon = 1-\epsilon$ from job 2 , but it also has to pay the energy cost of running job 1 which is $1-\epsilon$. Thus, if it switches it actually obtains no profit. Therefore we can assume the algorithm does not switch and obtains a net profit of ϵ from running job 1. The adversary on the other hand, only runs job 2 and obtains a profit of $1 - \epsilon$. The competitive ratio is therefore at least $(1 - \epsilon)/\epsilon$ and by making ϵ small, we can make this ratio as large as we like.

4 The Online Algorithm and Its Analysis

Given the results in the previous section, we consider resource augmentation in the remainder of this paper. Our main result is the following theorem:

Theorem 1. *For any $\epsilon > 0$, there is an online algorithm A that is $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^3})$-competitive for profit maximization.*

The purpose of this section is to prove Theorem 1.

In subsection 4.1 we define the concept of critical speed function, which is required for both the definition and the analysis of our online algorithm. In subsection 4.2 we describe our online algorithm. In subsection 4.3 we prove the existence of a near-optimal schedule with nice structural properties that will facilitate the comparison with the online schedule. Finally, in subsection 4.4 we compare the online schedule to this structurally-nice near-optimal schedule.

4.1 Critical Speed Function

If i is completed at time t, then the minimum speed at job i is run is $s_i^{\min}(t) = w_i/(t - r_i)$. Recall that we can assume without loss of generality that each job runs at a fixed speed. Thus in order for the adversary to obtain positive profit from job i with power function $P(s)$, when completing the job at time t, it must be the case that:

$$I_i(t) - P(s_i^{\min}(t))\frac{w_i}{s_i^{\min}(t)} > 0. \tag{1}$$

Alternatively, a feasible schedule complete job i at time t by running job i at a faster speed than $s_i^{\min}(t)$, and then no running job i for some times during the time interval $[r_i, t]$. As the speed that job i is run increases (with the completion time fixed at t), the energy cost increases. Thus there is a maximum speed at which job i can run, and complete at time t, while still having non-negative profit. We call this speed the *critical speed function* $\hat{s}_i(t)$. For time t, $\hat{s}_i(t)$ is by the unique solution to the equation:

$$I_i(t) - P(\hat{s}_i(t))\frac{w_i}{\hat{s}_i(t)} = 0. \tag{2}$$

Dividing (2) through by $1+\epsilon$ and regrouping terms, we can rewrite (2) as

$$I_i(t) - P(\hat{s}_i(t))\frac{w_i}{(1+\epsilon)\hat{s}_i(t)} = \frac{\epsilon}{1+\epsilon}I_i(t) \tag{3}$$

Lemma 4. *In any schedule in which non-negative profit is earned from job i with power function $P(s)$, the speed s_i that job i runs is in the range $[s_i^{\min}(C_i), \hat{s}_i(C_i)]$.*

Proof. Immediate from equations 1 and 2.

4.2 The Description of the Online Algorithm

We break the description of the online algorithm into four parts: invariants that are maintained throughout the course of the algorithm, the policy for setting deadlines and assigning jobs, the policy for job selection, and the speed scaling policy.

At a high level, when a job arrives, we use the deadline setting and job assignment policy to set a deadline and assign the job to various time intervals on machines. This assignment is not a schedule, as we may assign multiple jobs to the same machine at the same time. We also set a speed via the speed scaling policy. We then use the job selection policy to take some jobs from the intervals and machines on which they are assigned and actually run them on machines.

Throughout this section, we use $\delta = \epsilon/2$.

Invariants: The online algorithm maintains a pool Q of *admitted* jobs. A job i remains in Q until it is completed or its deadline passes. Each job i in Q has several associated attributes:

– A deadline d_i assigned to job i when it was released.
– The critical speed $\hat{s}_i^A = \hat{s}_i^A(d_i)$ derived from the deadline d_i, and defined by (2). The online algorithm will run job i at speed $(1+2\delta)\hat{s}_i^A$.
– A collection of time intervals $J(i) = \{[t_1, t_1'], [t_2, t_2'], \ldots [t_h, t_h']\}$, where $r_i \leq t_1 \leq t_1' \leq t_2 \leq \ldots \leq t_h \leq t_h' = d_i$. This collection $J(i)$ is fixed when job i is released (but depends on previously scheduled jobs). The total length of the time intervals in $J(i)$ will be $\frac{(1+\delta)w_i}{(1+2\delta)\hat{s}_i^A}$.
– A processor $m_{i,k}$ associated with job i and each time interval $[t_k, t_k'] \in J(i)$ that was fixed at time r_i. Intuitively, at the time that job i was released, the online algorithm is tentatively planning on running job i on processor $m_{i,k}$ during the time period $[t_k, t_k']$. We say that job i is *assigned* to run on m_k during times $[t_k, t_k']$.

Deadline Setting and Job Assignment Policy: Consider a job i that is released at time r_i. Setting the deadline at some d_i will fix a critical speed $\hat{s}_i^A = \hat{s}_i(d_i)$ for job i, a job profit $p_i^A = I_i(d_i) - \frac{P(\hat{s}_i^A)w_i}{\hat{s}_i^A(1+2\delta)}$, and an online density $u_i^A = p_i^A \hat{s}_i^A / w_i$. The online algorithm considers the possible choices for deadlines by nonincreasing order of the resulting profit p_i^A. So assume that the online algorithm is considering setting the deadline d_i to be a time t. Let $c = 1 + \frac{2}{\delta}$. Let $X(\frac{u_i^A}{c})$ be the set of jobs in Q with density at least $\frac{u_i^A}{c}$. Consider the time interval $[r_i, t]$ and the associated intervals of jobs in $X(\frac{u_i^A}{c})$. Let A be the maximal subintervals of $[r_i, t]$ of times such that for each $[a, a'] \in A$, there is a processor m_k for which no job in $X(\frac{u_i^A}{c})$ is assigned to run on m_k during any time in $[a, a']$. We now consider two cases. In the first case assume that the total length of intervals in A is at least $\frac{(1+\delta)w_i}{(1+2\delta)\hat{s}_i^A}$. The deadline d_i is then set to be the time such that the measure of the portion A^* of A earlier than d_i is exactly $\frac{(1+\delta)w_i}{(1+2\delta)\hat{s}_i^A}$.

$J(i)$ is set to be A^*, and the processor associated with each interval in $J(i)$ is the processor m_k in the definition of the interval in A. If the job profit for the adversary with completion time $C_i^{OPT} = d_i$, $I_i(d_i) - P(\hat{s}_i^A)w_i/\hat{s}_i^A$, is positive, then job i is admitted to the pool Q, In the second case, when the total length of intervals in A is less than $\frac{(1+\delta)w_i}{(1+2\delta)\hat{s}_i^A}$, the online algorithm *rejects* this candidate deadline, and the next most profitable time t is considered for the deadline. A job is not admitted if there is no time t satisfying the stated conditions.

Speed Scaling Policy: Every job is run at its critical speed for its set deadline.

Job Selection Policy: At any time t, on any processor m_k, run the job i,assigned to m_k at time t, of maximum density.

4.3 Construction of a Structurally-Nice Near-Optimal Schedule *OPT′*

Using the results from [21], given an optimal schedule on m processors, one can create a *non-migratory* schedule on $6m$ processors that has objective value at least as large. A schedule is non-migratory if no job ever runs on more than one processor. Therefore, by taking the m processors with the largest total net profit, one can assume that the optimal schedule is non-migratory (modulo a factor of 6 in the profit objective). Thus for the rest of this subsection, we assume that the optimal schedule is non-migratory.

In order to facilitate the comparison of the online schedule to the optimal schedule, we assume that each job has a separate power function $P_i'(s)$ that is slightly smaller than $P(s)$ for speeds less than the critical speed $\hat{s}_i(C_i^O)$. More formally,

$$P_i'(s) = \begin{cases} P(s/(1+\epsilon)) & \text{if } s \in [s_i^{\min}(C_i), \hat{s}_i(C_i^O)] \\ P(s) & \text{otherwise} \end{cases} . \tag{4}$$

Notice that $P_i'(s) \leq P(s)$, so clearly the optimal schedule with respect to power function P' is at least as profitable as the optimal with respect to the original power function. We then show that with these modified per-job power functions, there is a near optimal schedule where each job runs at near this critical speed.

Lemma 5. *There is a schedule OPT' such that in OPT' each job i that runs does so at speed $(1+\epsilon)\hat{s}_i(C_i^O)$, and the total profit obtained using the modified power functions is at least $\left(\frac{\epsilon}{1+\epsilon}\right)$ times the profit that OPT achieves using the power function P.*

Proof. For notation simplicity let $\hat{s}_i = \hat{s}_i(C_i^O)$, and $s_i^{\min} = s_i^{\min}(C_i^O)$. We modify the optimal schedule so that each job i is run at speed $\hat{s}_i$, and the profit p_i^O decreases by at most a factor of $\frac{\epsilon}{1+\epsilon}$. In OPT, by the definition of $\hat{s}_i$ in equation 2, we know that each job that runs is already running at speed at most $\hat{s}_i$. Combined with equation 1, we see that i is running at a speed s_i satisfying $s_i^{\min} \leq s_i \leq \hat{s}_i$. Thus, if we change the speed to $\hat{s}_i(1+\epsilon)$, we are speeding the job up, which implies that the schedule will certainly be feasible, i.e. each job still completes by its completion time C_i^O. Now, the net profit associated with job i is at least

$$\begin{aligned} I_i(C_i^O) - P_i'(\hat{s}_i(1+\epsilon))\frac{w_i}{\hat{s}_i(1+\epsilon)} &= I_i(C_i^O) - P(\hat{s}_i)\frac{w_i}{\hat{s}_i(1+\epsilon)} \\ &= \frac{\epsilon}{1+\epsilon} I_i(C_i^O) \\ &\geq \left(\frac{\epsilon}{1+\epsilon}\right) p_i^O . \end{aligned}$$

The first equality follows from the definition of P', the second equality follows from (3), and the final inequality follows because the income must be greater than the profit.

4.4 Analysis of the Online Algorithm

In this section we compare the online algorithm with $(1+\epsilon)^2$ speed augmentation to OPT' in terms of profit. To simplify the analysis, we will generously assume that the power function for OPT' is $P(s/(1+\epsilon))$, and that the power function for the online algorithm is $P(s/(1+\epsilon)^2)$. In our analysis, it will be convenient to scale work or speed so that the power functions for OPT' and the online algorithm are $P(s)$ and $P(s/(1+\epsilon))$ respectively. We also generously assume that the online algorithm only gains income $I_i(d_i)$ from finishing a job i before its deadline, when in fact its real income is $I_i(C_i^A)$. Superscripting or subscripting by the variable O means that we are referring to the schedule OPT'.

We define the notion of *adversarial density* of a job i as $u_i^O = p_i^O s_i^O / w_i$, where p_i^O is the profit obtained from job i in the schedule OPT'. The *density of the online schedule at time t on a processor m_k* is the density of the highest density admitted job assigned to processor m_k at that time t. The *density of the online schedule at time t* is the minimum density of any processor at that time. Let C be the set of jobs completed by the online algorithm, and let R be the set admitted jobs. For any set X of jobs, let $||X||_A$ be the total profit of jobs in X if each job in X was run at its critical speed and finished at its deadline. Similarly, let $||X||_O$ be the total profit of jobs in X if each job in X was run at its critical speed and finished at the completion time of the job in OPT'.

Observation 2. *At any time t, let i and j be two admitted jobs where there is a time t and a processor m_k such that both jobs are assigned to m_k at time t. Then, either $u_i^A > c \cdot u_j^A$ or $u_j^A > c \cdot u_i^A$.*

Observation 3. *Consider any job i and a time t that the deadline setting policy of the online algorithm considered, but decided not to use as the deadline. Let v be what the density for job i would have been, if d_i were set to t. Let L be the amount of time during $[r_i, t]$ such that the density of the online schedule at that time is at least v/c. Then, $L \geq \frac{\delta}{1+2\delta}(t - r_i)$.*

Lemma 6. *For C and R as defined above, $||C||_A \geq (1 - \frac{1}{\delta(c-1)})||R||_A$, or equivalently, $||R||_A \leq \frac{\delta(c-1)}{\delta(c-1)-1}||C||_A$.*

Proof. We use a charging scheme to prove the lemma. We initially give p_i^A units of credit to each job $i \in C$.The jobs in $R - C$ are initially given 0 units of credit. We will describe a method to transfer the credits such that at the end, each job $i \in R$ has credit at least $(1 - \frac{1}{\delta(c-1)})p_i^A$, which completes the proof.

The method to transfer credit is as follows. At any time t, and any processor m_k, let S be the set of jobs assigned to m_k at time t. Let job i be the highest density job in S. Then, for each other job $j \in S$, at time t job i transfers credit to j at a rate of $(\frac{1+2\delta}{\delta})u_j^A$ units of credit per unit time.

We first show that every job $j \in R$ receives credit at least p_j^A either initially or transferred from other jobs. This clearly holds for jobs in C. For any job $j \in R - C$, as job j could not be completed during $J(j)$, it must have received credit for at least $\frac{\delta}{1+2\delta} \cdot \frac{w_j}{s_j^A}$ units of time. Thus, the total credit obtained is at least

$$\left(\frac{\delta}{1+2\delta}\right)\left(\frac{w_j}{s_j^A}\right)\left(\frac{1+2\delta}{\delta}\right)u_j^A = \frac{w_j u_j^A}{s_j^A} = p_j^A$$

We now show that the credit transferred out of each job i is at most $\frac{1}{\delta(c-1)}p_i^A$. When a job i is the highest density job in S, by observation 2 the remaining jobs in S have geometrically decreasing densities and hence their total density is at most $\frac{1}{c-1}u_i^A$. Therefore, the rate of credit transferring out of i is at most $(\frac{u_i^A}{c-1})(\frac{1+2\delta}{\delta})$. Since job i is the highest density job for at most $\frac{w_i}{\hat{s}_i^A(1+2\delta)}$ units of time, the total credit transferred out of job i is at most

$$\left(\frac{u_i^A}{c-1}\right)\left(\frac{1+2\delta}{\delta}\right)\left(\frac{w_i}{\hat{s}_i^A(1+2\delta)}\right) = \frac{1}{\delta(c-1)}p_i^A.$$

Next, we upper bound the profit obtained by the adversary. Let B be the set of jobs completed by the adversary. Let B_2 be the set of jobs in B for which the adversary's completion time is a deadline rejected by the online algorithm. Let $B_1 = B \setminus B_2$. For any $u > 0$, let $T(u)$ be the total length of time that the adversary is running a job in B_2 with adversarial density at least u. Let $L(\frac{u}{c})$ be the total length of time such that the density of the online schedule at that time is at least $\frac{u}{c}$.

Lemma 7. *For every $u > 0$, $T(u) \leq \frac{2(1+2\delta)}{\delta}L(\frac{u}{c})$.*

Proof. For any job $i \in B_2$, let the span of i be the time interval $[r_i, C_i^O]$, where C_i^O is the completion time for the adversary. For any $u > 0$, let $B_2(u)$ be the set of jobs in B_2 with density at least u. Consider the union of spans of all jobs in $B_2(u)$. This union may consist of a number of disjoint time intervals. Let ℓ be its total length. Clearly, $T(u) \leq \ell$.

Let $M \subseteq B_2$ be a minimal cardinality subset of B_2 such that the union of spans of jobs in M equals that of B_2. Note that the minimality property implies no three jobs in M have their spans overlapping at a common time. This implies that we can further partition M into M_1 and M_2 such that within M_1 (resp. M_2), any two jobs have disjoint spans. Now, either M_1 or M_2 has total span of length at least half of that of M. Without loss of generality, suppose that it is M_1. Note that each interval in M_1 corresponds to a span of some job in B_2. Applying Observation 3 to each such interval, it follows that the density of the online schedule is at least $\frac{u}{c}$ for at least $\frac{\delta}{1+2\delta}$ fraction of time during the intervals of M_1. Thus, $L(\frac{u}{c}) \geq \frac{\delta}{2(1+2\delta)} \cdot T(u)$.

Lemma 8. $||B||_O \leq (1 + \frac{2(1+\delta)c}{\delta})||R||_A$.

Proof. Let $\{\phi_1, \phi_2, \dots, \phi_m\}$ be the set of the adversarial densities of jobs in B_2, where $\phi_i > \phi_{i+1}$ for $i = 1, \dots, m-1$. For $i = 1, \dots, m$, let ℓ_i be the sum over all processors of the length of time that the adversary is running jobs of density ϕ_i on that processor. Similarly, for $i = 1, \dots, m$, let α_i be the sum over all processors of the length of time that the online schedule on that processor has density in the range $[\phi_i/c, \phi_{i-1}/c)$. Let q_i be the total profit for jobs whose density for the online algorithm is in the range of $[\phi_i/c, \phi_{i-1}/c)$. Then applying Lemma 7

$$||B_2||_O \leq \frac{2(1+2\delta)}{\delta} \sum_{i=1}^{m} \alpha_i \phi_i \leq \frac{2(1+\delta)c}{\delta} \sum_{i=1}^{m} q_i \leq \frac{2(1+\delta)c}{\delta} ||R||_A$$

The proof then follows by noting that $||B||_O = ||B_1||_O + ||B_2||_O \leq ||R||_A + ||B_2||_O$.

That the online algorithm is $(1+\epsilon)$-speed, $O(\frac{1}{\epsilon^3})$-competitive now follows immediately from Lemma 5, Lemma 6 and Lemma 8.

5 Conclusions

We introduced a new type of power management problem into the algorithmic literature, and showed that there is a scalable algorithm for the problem of maximizing profit when you have to buy your energy. It would be interesting to investigate other problems within this general framework.

Acknowledgments

We thank Christoph Durr and Nguyen Kim Thang for several helpful discussions.

References

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3(4) (2007)
2. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proc. ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 289–298 (2007)

3. Bansal, N., Chan, H.L., Pruhs, K.: Speed scaling with a solar cell. In: International Conference on Algorithmic Aspects in Information and Management (2008)
4. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. JACM 54(1) (2007)
5. Bansal, N., Bunde, D., Chan, H.L., Pruhs, K.: Average rate speed scaling. In: Latin American Theoretical Informatics Symposium (2008)
6. Bansal, N., Chan, H.-L., Lam, T.W., Lee, L.-K.: Scheduling for speed bounded processors. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 409–420. Springer, Heidelberg (2008)
7. Bansal, N., Chan, H.-L., Pruhs, K.: Competitive algorithms for due date scheduling. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 28–39. Springer, Heidelberg (2007)
8. Bansal, N., Chan, H.-L., Pruhs, K.: Speed scaling with an arbitrary power function. In: SODA, pp. 693–701 (2009)
9. Bansal, N., Chan, H.-L., Pruhs, K., Katz, D.: Improved bounds for speed scaling in devices obeying the cube-root rule. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5556, pp. 144–155. Springer, Heidelberg (2009)
10. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 805–813 (2007)
11. Barroso, L.A.: The price of performance. ACM Queue 3(7), 48–53 (2005)
12. 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.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
13. Bunde, D.P.: Power-aware scheduling for makespan and flow. J. Scheduling 12(5), 489–500 (2009)
14. Chan, H.L., Chan, W.-T., Lam, T.-W., Lee, L.-K., Mak, K.-S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: ACM-SIAM Symposium on Discrete Algorithms, pp. 795–804 (2007)
15. Chan, H.-L., Chan, W.-T., Lam, T.W., Lee, L.-K., Mak, K.-S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: SODA, pp. 795–804 (2007)
16. Chan, H.-L., Edmonds, J., Lam, T.W., Lee, L.-K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. In: STACS, pp. 255–264 (2009)
17. Chan, H.-L., Edmonds, J., Pruhs, K.: Speed scaling of processes with arbitrary speedup curves on a multiprocessor. In: SPAA, pp. 1–10 (2009)
18. Chan, H.-L., Lam, T.W., To, K.-K.: Nonmigratory online deadline scheduling on multiprocessors. SIAM J. Comput. 34(3), 669–682 (2005)
19. Fan, X., Weber, W.-D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: International Symposium on Computer Architecture, pp. 13–23 (2007)
20. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. J. ACM 47(4), 617–643 (2000)
21. Kalyanasundaram, B., Pruhs, K.: Eliminating migration in multi-processor scheduling. J. Algorithms 38(1), 2–24 (2001)
22. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. Journal of Combinatorial Optimization 11(3), 305–319 (2006)
23. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM J. on Computing 35, 658–671 (2005)

24. Markoff, J., Lohr, S.: Intel's huge bet turns iffy. New York Times (September 29, 2002)
25. Mudge, T.: Power: A first-class architectural design constraint. Computer 34(4), 52–58 (2001)
26. Phillips, C.A., Stein, C., Torng, E., Wein, J.: Optimal time-critical scheduling via resource augmentation. Algorithmica 32(2), 163–200 (2002)
27. Pruhs, K.: Competitive online scheduling for server systems. SIGMETRICS Performance Evaluation Review 34(4), 52–58 (2007)
28. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. In: Handbook on Scheduling. CRC Press, Boca Raton (2004)
29. Pruhs, K., Uthaisombut, P., Woeginger, G.: Getting the best response for your erg. In: Scandanavian Workshop on Algorithms and Theory (2004)
30. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory Comput. Syst. 43(1), 67–80 (2008)
31. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. IEEE Symp. Foundations of Computer Science, pp. 374–382 (1995)