Nonclairvoyant Speed Scaling for Flow and Energy

Ho-Leung Chan* Jeff Edmonds† Tak-Wah Lam* Lap-Kei Lee*
Alberto Marchetti-Spaccamela‡ Kirk Pruhs§

Abstract

We study online nonclairvoyant speed scaling to minimize total flow time plus energy. We first consider the traditional model where the power function is $P(s) = s^\alpha$. We give a nonclairvoyant algorithm that is shown to be $O(\frac{\alpha^2}{\log \alpha})$-competitive. We then show an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any nonclairvoyant algorithm. We also show that there are power functions for which no nonclairvoyant algorithm can be $O(1)$-competitive.

1 Introduction

Energy consumption has become a key issue in the design of microprocessors. Major chip manufacturers, such as Intel, AMD and IBM, now produce chips with dynamically scalable speeds, and produce associated software, such as Intel's SpeedStep and AMD's PowerNow, that enables an operating system to manage power by scaling processor speed. Thus the operating system should have a *speed scaling* policy for setting the speed of the processor, that ideally should work in tandem with a *job selection* policy for determining which job to run. The operating system has dual competing objectives, as it both wants to optimize some schedule quality of service objective, as well as some power related objective.

In this paper, we will consider the objective of minimizing a linear combination of total flow and total energy used. For a formal definitions of the problem that we consider, see subsection 1.2. This objective of flow plus energy has a natural interpretation: suppose that the user specifies how much improvement in flow, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify that he is willing to spend 1 erg of energy from the battery for a decrease of 5 micro-seconds in flow. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 5$ times the energy

*The University of Hong Kong. {hlchan,twlam,lklee}@cs.hku.hk. This work of Ho-Leung Chan was done when he was a postdoc in University of Pittsburgh. Tak-Wah Lam is partially supported by HKU Grant 7176104.

†York University. jeff@cs.yorku.ca. Supported in part by NSERC Canada.

‡Dipartimento di Informatica e Sistemistica, Sapienza Università di Roma. alberto@dis.uniroma1.it. Supported in part by MIUR FIRB grant RBIN047MH9 and by EU ICT-FET grant 215270 FRONTS.

§University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by an IBM faculty award, and by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF-0830558.

used plus the total flow. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

In order to be implementable in a real system, the speed scaling and job selection policies must be online since the system will not in general know about jobs arriving in the future. Further, to be implementable in a generic operating system, these policies must be nonclairvoyant, since in general the operating system does not know the size/work of each process when the process is released to the operating system. All of the previous speed scaling literature on this objective has considered either offline or online clairvoyant policies. In subsection 1.1, we survey the literature on nonclairvoyant scheduling policies for flow objectives on fixed speed processors, and the speed scaling literature for flow plus energy objectives.

Our goal in this paper is to study nonclairvoyant speed scaling policies using competitive analysis.

We first analyze the nonclairvoyant algorithm whose job selection policy is Latest Arrival Processor Sharing (LAPS) and whose speed scaling policy is to run at the speed such that the power equals the number of active jobs. LAPS shares the processor equally among the latest arriving constant fraction of the jobs. We adopt the traditional model that the power function, which gives the power as a function of the speed of the processor, is $P(s) = s^\alpha$, where $\alpha > 1$ is some constant. Of particular interest is the case $\alpha = 3$ since, according to the well known cube-root rule, the dynamic power in CMOS based processors is approximately the cube of the speed. Using an amortized local competitiveness argument, we show in section 2 that this algorithm is $O(\frac{\alpha^2}{\log \alpha})$-competitive. The potential function that we use is an amalgamation of the potential function used in [8] for the fixed speed analysis of LAPS, and the potential functions used for analyzing clairvoyant speed scaling policies. This result shows that it is possible for a nonclairvoyant policy to be $O(1)$-competitive if the cube-root rule holds.

It is known that for essentially every power function, there is a 3-competitive *clairvoyant* speed scaling policy [3]. In contrast, we show that the competitiveness achievable by *nonclairvoyant* policies must depend on the power function. In the traditional model, we show in section 3 an $\Omega(\alpha^{1/3-\epsilon})$ lower bound on the competitive ratio of any deterministic nonclairvoyant algorithm. Further, we show in section 3 that there exists a particular power function for which there is no $O(1)$-competitive deterministic nonclairvoyant speed scaling algorithm. The adversarial strategies for these lower bounds are based on the adversarial strategies in [12] for fixed speed processors. Perhaps these lower bound results are not so surprising given the fact that it is known that without speed scaling, resource augmentation is required to achieve $O(1)$-competitiveness for a nonclairvoyant policy [9, 12]. Still a priori it wasn't completely clear that the lower bounds in [12] would carry over. The reason is that in these lower bound instances, the adversary forced the online algorithm into a situation in which the online algorithm had a lot of jobs with a small amount of remaining work, while the adversary had one job left with a lot of remaining work. In the fixed speed setting, the online algorithm, without resource augmentation, can never get a chance to get rid of this backlog in the face of a steady stream of jobs. However, in a speed scaling setting, one might imagine an online algorithm that speeds up enough to remove the backlog, but not enough to make its energy usage more than a constant times optimal. Our lower bound shows that

it is not possible for the online algorithm to accomplish this.

1.1 Related results

We start with some results in the literature about scheduling with the objective of total flow time on a fixed speed processor. It is well known that the online clairvoyant algorithm Shortest Remaining Processing Time (SRPT) is optimal. The competitive ratio of deterministic nonclairvoyant algorithm is $\Omega(n^{1/3})$, and the competitive ratio of every randomized algorithm against an oblivious adversary is $\Omega(\log n)$ [12]. A randomized version of the Multi-Level Feedback Queue algorithm is $O(\log n)$-competitive [5, 10]. The non-clairvoyant algorithm Shortest Elapsed Time First (SETF) is scalable, that is, $(1+\epsilon)$-speed $O(1)$-competitive [9]. SETF shares the processor equally among all jobs that have been run the least. The algorithm Round Robin RR (also called Equipartition and Processor Sharing) that shares the processor equally among all jobs is $(2+\epsilon)$-speed $O(1)$-competitive [7].

Let us first consider the traditional model where the power function is $P = s^\alpha$. Most of the literature assumes the *unbounded speed model*, in which a processor can be run at any real speed in the range $[0, \infty)$. In this model [14] gave an efficient offline algorithm to find the schedule that minimizes average flow subject to a constraint on the amount of energy used, in the case that jobs have unit work. This algorithm can also be used to find optimal schedules when the objective is a linear combination of total flow and energy used. The authors of [14] observed that in any locally-optimal schedule, essentially each job i is run at a power proportional to the number of jobs that would be delayed if job i was delayed. In [1] the authors proposed the natural online speed scaling algorithm that always runs at a power equal to the number of unfinished jobs (which is lower bound to the number of jobs that would be delayed if the selected job was delayed). [1] did not actually analyze this natural algorithm, but rather analyzed a batched variation, in which jobs that are released while the current batch is running are ignored until the current batch finishes, showing that, for unit work jobs, this batched algorithm is $O\left(\left(\frac{3+\sqrt{5}}{2}\right)^\alpha\right)$-competitive by reasoning directly about the optimal schedule. [1] also gave an efficient offline dynamic programming algorithm. [4] considered the algorithm that runs at a power equal to the unfinished work (which is in general a bit less than the number of unfinished jobs for unit work jobs) showing that, for unit work jobs, this algorithm is 2-competitive with respect to the objective of fractional flow plus energy using an amortized local competitiveness argument. in [4] it is also shown that the natural algorithm proposed in [1] is 4-competitive for total flow plus energy for unit work jobs.

In [4] the more general setting where jobs have arbitrary sizes and arbitrary weights and the objective is weighted flow plus energy has also been considered. The authors analyzed the algorithm that uses Highest Density First (HDF) for job selection, and always runs at a power equal to the fractional weight of the unfinished jobs showing that this algorithm is $O(\frac{\alpha}{\log \alpha})$-competitive for fractional weighted flow plus energy using an amortized local competitiveness argument. [4] then showed how to modify this algorithm to obtain an algorithm that is $O(\frac{\alpha^2}{\log^2 \alpha})$-competitive for (integral) weighted flow plus energy using the known resource augmentation analysis of HDF [6].

Recently, [11] improves on the obtainable competitive ratio for total flow plus energy