Speed Scaling for Weighted Flow Time

Nikhil Bansal [*] *Kirk Pruhs* [†] *Cliff Stein* [‡]

Abstract

Intel's SpeedStep and AMD's PowerNOW technologies allow the Windows XP operating system to dynamically change the speed of the processor to prolong battery life. In this setting, the operating system must not only have a *job selection policy* to determine which job to run, but also a *speed scaling* policy to determine the speed at which the job will be run. We give an online speed scaling algorithm that is $O(1)$-competitive for the objective of weighted flow time plus energy. This algorithm also allows us to efficiently construct an $O(1)$-approximate schedule for minimizing weighted flow time subject to an energy constraint.

1 Introduction

In addition to the traditional goal of efficiently managing time and space, many computers now need to efficiently manage power usage. For example, Intel's SpeedStep and AMD's PowerNOW technologies allow the Windows XP operating system to dynamically change the speed of the processor to prolong battery life. In this setting, the operating system must not only have a *job selection policy* to determine which job to run, but also a *speed scaling* policy to determine the speed at which the job will be run. These policies must be online since the operating system does not, in general, have knowledge of the future. In current CMOS based processors, the speed satisfies the well known cube-root-rule, that the speed is approximately the cube root of the power [Mud01, BBS+00]. Thus, in this work, we make the standard generalization that the power used by a processor is equal to speed to some power $\alpha \geq 1$, where one should think of α as being approximately 3 [YDS95, BKP07]. Energy is power integrated over time. An operating system is faced with a dual objective optimization problem as it both wants to conserve energy, and optimize some Quality of Service (QoS) measure of the resulting schedule.

By far the most commonly used QoS measure in the computer systems literature is *average response/flow time* or more generally *weighted response/flow time*. The flow time F_i of a job i is the time lag between when a job is released to the system and when the system completes that job. Pruhs, Uthaisombut, and Woeginger [PUW08] studied the problem of optimizing total flow time ($\sum_i F_i$) subject to the constraint that the total energy does not exceed some bound, say the energy in the battery, and showed how to efficiently construct

[*]IBM T.J. Watson Research, P.O. Box 218, Yorktown Heights, NY. nikhil@us.ibm.com.

[†]Computer Science Department. University of Pittsburgh. kirk@cs.pitt.edu. Supported in part by NSF grants CNS-0325353, CCF-0448196, CCF-0514058 and IIS-0534531.

[‡]Department of IEOR, Columbia University, New York, NY. Supported in part by NSF grant CCF-0728733 and an IBM faculty fellowship. cliff@ieor.columbia.edu.

offline an optimal schedule for instances with unit work jobs. For unit work jobs, all job selection policies that favor a partially executed job in favor of an unexecuted job are equivalent. Thus the job selection policy is essentially irrelevant.

If there is an upper bound on energy used, then there is no $O(1)$-competitive online speed scaling policy for total flow time. To understand intuitively why this is the case, consider the situation when the first job arrives. The scheduler has to allocate a constant fraction of the total energy to this job; otherwise, the scheduler would not be $O(1)$-competitive in the case that no more jobs arrive. However, if many more jobs arrive in the future, then the scheduler has wasted a constant fraction of its energy on only one job. By iterating this process, one obtains a bound of $\omega(1)$ on the competitive ratio with respect to total flow time. (See Section 6.)

Albers and Fujiwara [AF07] proposed combining the dual objectives of energy and flow time into the single of objective of energy used plus total flow time. Optimizing a linear combination of energy and total flow time has the following natural interpretation. Suppose that the user specifies how much improvement in flow time, call this amount ρ, is necessary to justify spending one unit of energy. For example, the user might specify to the Windows XP operating system that he is willing to spend 1 erg of energy from the battery for a decrease of 3 micro-seconds in response time. Then the optimal schedule, from this user's perspective, is the schedule that optimizes $\rho = 3$ times the energy used plus the total flow time. By changing the units of either energy or time, one may assume without loss of generality that $\rho = 1$.

[PUW08] observe that in any locally-optimal normal schedule, each job i is run at a power proportional to the number of jobs that depend on i. Roughly speaking, *normal* means that no job completes exactly when another job is released. We say that a job j *depends* on a job i if delaying i would delay j. In the online setting, an obvious lower bound to the number of jobs that depend on the selected job is the number of *active* jobs, where an active job is one that has been released but has not yet completed. Thus Albers and Fujiwara [AF07] propose the natural online speed scaling algorithm that always runs at a power equal to the number of active jobs. They again only consider the case of unit work jobs. They do not actually analyze this natural algorithm, but rather analyze a batched variation, in which jobs that are released while the current batch is running are ignored until the current batch finishes. They show that this batched algorithm is $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$-competitive with respect to the objective of total flow time plus energy, and also give a dynamic programming algorithm to compute the offline optimal schedule for unit work jobs.

One reason that both [PUW08] and [AF07] consider only unit work jobs is that it seems that the optimal schedule for arbitrary work jobs is quite difficult to characterize.

1.1 Our Results

We give significantly stronger results for the problem of minimizing the objective of (weighted) flow time plus energy. We both improve the algorithm and analysis in the special case (unit jobs, no weights) considered previously [AF07] and then we give algorithms for the more general problem with weights and arbitrary work jobs.

First, we show that the natural online speed scaling algorithm proposed in [AF07] is 4-competitive for unit work jobs. This guarantee is independent of α. In comparison, the competitive ratio $8.3e(\frac{3+\sqrt{5}}{2})^{\alpha}$ obtained in [AF07] is a bit over 400 when the cube-root rule holds ($\alpha = 3$).

More importantly, we consider the case of arbitrary work jobs, and consider a much more general QoS measure: weighted flow. We assume that each job has a positive integer weight w_i. The weighted flow objective is then the weighted sum of flow times, $\sum_i w_i F_i$. Weighted flow generalizes both total flow time, and total/average stretch, which is another common QoS measure in the systems literature. The stretch/slow-

down of a job is the flow time divided by the work of the job. Many server systems, such as operating systems and databases, have mechanisms that allow the user or the system to give different priorities to different jobs. For example, Unix has the