A Scheduling Model for Reduced CPU Energy

Frances Yao Alan Demers Scott Shenker

Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304
{yao, demers, shenker}@parc.xerox.com

(extended abstract)

Abstract

The energy usage of computer systems is becoming an important consideration, especially for battery-operated systems. Various methods for reducing energy consumption have been investigated, both at the circuit level and at the operating systems level. In this paper, we propose a simple model of job scheduling aimed at capturing some key aspects of energy minimization. In this model, each job is to be executed between its arrival time and deadline by a single processor with variable speed, under the assumption that energy usage per unit time, P, is a convex function of the processor speed s. We give an off-line algorithm that computes, for any set of jobs, a minimum-energy schedule. We then consider some on-line algorithms and their competitive performance for the power function $P(s) = s^p$ where $p \geq 2$. It is shown that one natural heuristic, called the Average Rate heuristic, uses at most a constant times the minimum energy required. The analysis involves bounding the largest eigenvalue in matrices of a special type.

1 Introduction

Computers are rapidly becoming more widespread and more portable. For portable computers running on batteries, energy conservation is critically important. In a typical laptop computer, energy use is dominated by the backlit display and the disk. It is difficult to modulate the power consumption of these devices while they are operating, so energy saving techniques primarily involve turning them off after a period of no use.

The new generation of very small portable computers (PDAs) often have no disk at all, and lack the backlight that consumes much of the display-related power. For such devices, the power consumption of the CPU itself becomes significant. This fact is important because there are energy conservation techniques for CPUs that do considerably better than simply turning off the device during its "idle loop". In particular, CPU circuitry can be designed so that slower clock speeds use lower supply voltage, thus resulting in lower energy consumption per instruction (see [1,2,4,7] for various approaches). Such variable speed processors can operate reliably over a range of clock speeds. The power (i.e., energy per unit time) consumed by such a processor is a convex function of its execution speed, with the exact form dependent on the details of the technology.

On a computer with a variable speed processor, the operating system can reduce the energy consumption by scheduling jobs appropriately. Scheduling to reduce power consumption was first discussed in [7], which described several scheduling heuristics and measured the energy savings on typical work loads. This work was later extended in [3].

In this paper, we provide a more formal analysis of the minimum-energy scheduling problem. We propose a simple model in which each job is to be executed between its arrival time and deadline by a single variable-speed processor as described above. A precise definition of the model is given in Section 2. In Section 3, we give an off-line algorithm that computes a minimum-energy schedule for any set of jobs, with no restriction on the power consumption function except convexity. We then consider on-line heuristics in Section 4, with special focus on what we call the Average Rate heuristic (AVR). In Section 5, we prove that AVR has a constant competitive ratio, i.e., it uses at most a constant times the minimum energy required,

0272-5428/95 $04.00 © 1995 IEEE

assuming a quadratic power function $P(s) = s^2$. Our analysis shows that the ratio lies between 4 and 8. In Section 6, we sketch a constant-ratio proof for the general case $P(s) = s^p$ where $p \geq 2$. There, the ratio is shown to be between p^p and $2^{p-1}p^p$. Finally, we close with a discussion of some simulation results and open problems.

2 The Model

Let $[t_0, t_1]$ be a fixed time interval. An instance of the scheduling problem is a set J of *jobs* to be executed during $[t_0, t_1]$. Associated with each job $j \in J$ are the following parameters:

- a_j its *arrival time*,
- b_j its *deadline* ($b_j > a_j$), and
- R_j its required number of CPU cycles.

We refer to $[a_j, b_j]$ as the *interval* of job j. A *schedule* is a pair $\mathcal{S} = (s, job)$ of functions defined over $[t_0, t_1]$:

- $s(t) \geq 0$ is the processor speed at time t;
- $job(t)$ defines the job being executed at time t (or *idle* if $s(t) = 0$).

We require that $s(t)$ and $job(t)$ be piecewise constant with finitely many discontinuities. A *feasible* schedule for an instance J is a schedule $\mathcal{S}$ that satisfies

$$\int_{a_j}^{b_j} s(t)\delta(job(t), j)dt = R_j$$

for all $j \in J$ (where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise). In other words, $\mathcal{S}$ must give each job j the required number of cyles between its arrival time and deadline (with perhaps intermittent execution). We assume that the power P, or energy consumed per unit time, is a convex function of the processor speed. The total energy consumed by a schedule S is[1]

$$E(\mathcal{S}) = \int_{t_0}^{t_1} P(s(t))dt.$$

The goal of the scheduling problem is to find, for any given problem instance, a feasible schedule that minimizes $E(\mathcal{S})$.

[1] In the remainder of this paper, unless otherwise specified, all integrals are taken with respect to t, with t_0 and t_1 as lower and upper limits. We will use abbreviated notations whenever possible.

3 The Minimum Energy Scheduler

In this section, we consider the off-line version of the scheduling problem. We first give a charaterization of an energy-optimal schedule for any set of n jobs, which then leads naturally to an $O(n \log^2 n)$ time algorithm for computing such schedules.

The characterization will be based on the notion of a *critical interval* for J, which is an interval in which a group of jobs must be scheduled at maximum, constant speed in any optimal schedule for J. The algorithm proceeds by identifying such a critical interval for J, scheduling those 'critical' jobs, then constructing a subproblem for the remaining jobs and solving it recursively. The optimal $s(t)$ is in fact unique, whereas $job(t)$ is not always so. The details are given below.

Definition. Define the *intensity* of an interval $I = [z, z']$ to be

$$g(I) = \frac{\sum R_j}{z' - z}$$

where the sum is taken over all jobs j with $[a_j, b_j] \subseteq [z, z']$.

Clearly, $g(I)$ is a lower bound on the average processing speed, $\int_z^{z'} s(t)dt/(z' - z)$, that must be achieved by any feasible schedule over the interval $[z, z']$. Thus, by convexity of the power function, a schedule using constant speed $g(I)$ on $[z, z']$ is necessarily optimal on that interval (in the sense that no other feasible schedule can use less power on that interval).

Definition. Let $I^* = [z, z']$ be an interval that maximizes $g(I)$. We call I^* a *critical interval* for J, and the set of jobs $J_{I^*} = \{ j \mid [a_j, b_j] \subseteq [z, z'] \}$ the *critical group* for J.

Note that we can assume $I^* = [a_i, b_j]$ for some i, j. The following theorem shows that a critical interval will determine a segment of the optimal schedule. We omit the proof here.

Theorem 1. *Let I^* be a critical interval for J. If $\mathcal{S}$ is an optimal schedule for J, then the maximum speed of $\mathcal{S}$ is $g(I^*)$, and $\mathcal{S}$ runs at that speed for the entire interval I^*.*

(Moreover, S must execute every job of J_{I^*} completely within I^*, and execute no other jobs during I^*.) Theorem 1 immediately leads to the following algorithm, which finds an optimal schedule for J by computing a sequence of critical intervals iteratively.

Algorithm [Optimal-Schedule]

Repeat the following steps until J is empty:

1. Identify a critical interval $I^* = [z, z']$ by computing $s = \max g(I)$, and schedule the jobs of J_{I^*} at speed s over interval I^* by the *earliest deadline* policy (which is always feasible, see [6]);

2. Modify the problem to reflect the deletion of jobs J_{I^*} and interval I^*. This involves:
   let $J \leftarrow J - J_{I^*}$;
   reset any deadline $b_j \leftarrow t$ if $b_j \in [z, z']$, and
   $b_j \leftarrow b_j - (z' - z)$ if $b_j \geq z'$;
   reset the arrival times similarly.

Note that, after each iteration, the intensity $g(I)$ of some intervals I may increase (because I has been 'compressed'), which affects the evaluation of $\max g(I)$ in the next round. A straightward implementation of the above algorithm requires $O(n^2)$ time for $|J| = n$. By using a suitable data structure, such as the *segment tree*, one can reduce the running time to $O(n \log^2 n)$. We will skip the implementation details here.

A job instance J and its corresponding optimal schedule $\mathcal{S}$ are shown in Figure 1. To keep the diagram simple, there is only one job in each critical group. Job j is executed at speed s_j over interval I_j^*, which we represent by a shaded rectangle with base I_j^* and height s_j. The original job j is shown as a rectangle with base I_j and height d_j. (These two rectangles coincide when s_j is a local maximum, such as the case for $j = 1, 3$ in the example.) Note that, by the way $\mathcal{S}$ is constructed, the interval I of any job belonging to the j-th critical group must satisfy

$$I \subseteq \bigcup_{i \leq j} I_i^*. \tag{1}$$

4 On-line Scheduling Heuristics

Obviously there is a large space of possible heuristics for the online version of the minimum-energy scheduling problem. We will mention two simple heuristics that appear natural:

• **Average Rate:** Associated with each job j is its *average-rate requirement* or *density*

$$d_j = \frac{R_j}{b_j - a_j}.$$

We define a corresponding step function $d_j(t) = d_j$ for $t \in [a_j, b_j]$, and $d_j(t) = 0$ elsewhere. At any time t, the *Average Rate Heuristic* (AVR) sets the processor speed at

$$s(t) = \sum_j d_j(t),$$

and use the earliest-deadline policy to choose among available jobs. It is easy to see that the strategy yields a feasible schedule.

• **Optimal Available:** After each arrival, recompute an optimal schedule for the problem instance consisting of the newly arrived job and the remaining portions of all other available jobs. (Thus, the recomputation is done for a set of jobs all having the same arrival time.)

In the remainder of this paper, we will focus on the AVR heuristic and analyze its competitive ratio. Since the competitive ratio depends on the precise form of $P(s)$, and because the competitive analysis is fairly complex, we first focus our attention on the case where $P(s) = s^2$. This represents the simplest nontrivial version of the energy minimization problem.

Given a problem instance J, let $\mathrm{OPT}(J)$ denote the energy cost of an optimal schedule, and let

$$\mathrm{AVR}(J) = \int (\sum_j d_j(t))^2 dt \tag{2}$$

denote the cost of the heuristic schedule. The competitive ratio of the heuristic is defined to be, as usual, the least upper bound of $\mathrm{AVR}(J)/\mathrm{OPT}(J)$ over all J. We first look at how AVR performs in some specific cases. Let $[t_0, t_1] = [0, 1]$, and $|J| = n$ in the following examples.

Example 1. The ith job has interval $[0, 1/2^{i-1}]$. All jobs have density $d_i = 1/2$, except $d_n = 1$. (See Figure 2.)

The optimal schedule for this example has constant speed $s(t) = 1$, and executes the jobs in the order $J_n, \ldots, J_1$, with total energy cost 1. By evaluating Eq. 2, one finds that the energy used by AVR approaches 2 as $n \to \infty$, resulting in $\mathrm{AVR}(J)/\mathrm{OPT}(J) = 2$.

Example 2. The ith job has interval $[0, i/n]$, and density $d_i = (n/i)^e$ where $e \geq 1$. (See Figure 3.)

It can be verified that the jobs will form critical groups in the order $J_1, \ldots, J_n$. When $e = 1$, the optimal schedule has constant speed 1 and AVR has cost 2 as $n \to \infty$, giving a ratio of 2 again. With a careful analysis, one can prove that the ratio $\mathrm{AVR}(J)/\mathrm{OPT}(J)$ is maximized at 4 when e is chosen to be $3/2$.