Algorithmic Problems in Power Management

Sandy Irani
School of Information and Computer Science
University of California, Irvine

Kirk R. Pruhs *
Computer Science Department
University of Pittsburgh

1 Introduction

We survey recent research that has appeared in the theoretical computer science literature on algorithmic problems related to power management. We will try to highlight some open problems that we feel are interesting. This survey places more concentration on lines of research of the authors: managing power using the techniques of speed scaling and power-down which are also currently the dominant techniques in practice.

1.1 Motivation

The power consumption rate of computing devices has been increasing exponentially. Since the early 1970s, the power densities in microprocessors have doubled every three years [34]. This increased power usage poses two types of difficulties:

- **Energy Consumption**: As energy is power integrated over time, supplying the required energy may become prohibitively expensive, or even technologically infeasible. This is a particular difficulty in devices that rely heavily on batteries for energy, and will will become even more critical as battery capacities are increasing at a much slower rate than power consumption. Anyone using a laptop on a long flight is familiar with this problem.

- **Temperature**: The energy used in computing devices is in large part converted into heat. For high-performance processors, cooling solutions are rising at \$1 to \$3 per watt of heat dissipated, meaning that cooling costs are rising exponentially and threaten the computer industry's ability to deploy new systems [34]. In May 2004 Intel publicly acknowledged that it had hit a "thermal wall" on its microprocessor line. Intel scrapped the development of its Tejas and Jayhawk chips in order to rush to the marketplace a more efficient chip technology. Designers said that the escalating heat problems were so severe that they threatened to cause its chips to fracture [27]. For a striking example of the grievous effect of removing the fan from a modern processor, see `http://www.cs.pitt.edu/~kirk/cool.avi`.(You will need a DivX codec installed.)

These two factors have resulted in power becoming a first-class design constraint for modern computing devices [28].

There is an extensive literature on power management in computing devices. Overviews can be found in [11, 28, 36]. All of these techniques that have been investigated are similar in that they reduce or eliminate power to some or all components of the device. Sensor networks have emerged as an important new

* Supported in part by NSF grants CCR-0098752, ANI-0123705, CNS-0325353, and CCF-0448196.

paradigm in which power-aware computation is absolutely critical. The explosive interest in sensor networks is the result of the development of low-cost, low-power mutifunctional sensor devices, such as the Smart Dust Mote [1, 22], that are small in size and communicate untethered at short distance.

There is an inherent conflict between power reduction and performance; in general, the more power that is available, the better the performance that can be achieved. As a result, it is generally proposed that power reduction techniques be preferentially applied during times when performance is less critical. However, this requires a policy to determine how essential performance is at any given time and how to apply a particular power reduction technique. Current tools and mechanisms for power management are inadequate and require more research [14]. Furthermore, there is a growing consensus that these policies must incorporate information provided by applications and high levels of the operating system in order to achieve necessary advances [14].

We advocate formalizing power management problems as optimization problems, and then developing algorithms that are optimal by these criteria. The goal is to develop effective algorithms for specific problems within the domain of power management as well as to build a toolkit of widely applicable algorithmic methods for problems that arise in energy-bounded and temperature-bounded computation.

2 Speed Scaling

2.1 Formulation as a Scheduling Problem

Speed scaling involves dynamically changing the voltage and/or frequency/speed of the processor. A processor consumes less power when it is run at a lower speed. Both in academic research and practice, dynamic voltage/frequency/speed scaling is the dominant technique to reduce switching loss, which is currently the dominant form of energy consumption in microprocessors [11, 28, 36]. Current microprocessors from AMD, Intel and Transmeta allow the speed of the microprocessor to be set dynamically. Informally, speed scaling problems involve determining the speed of the processor at each point in time.

Theoretical investigations of speed scaling algorithms were initiated by Yao, Demers, and Shankar [37]. Yao *et al.* [37] propose formulating speed scaling problems as scheduling problems. The setting is a collection of tasks, where each task i has a release time r_i when it arrives into the system, and an amount of work w_i that must be performed to complete the task. A schedule specifies which task to run at each time, and at what speed that task should be run.

In particular, Yao *et al.* [37] consider the case that there is also a deadline d_i associated with each task that specifies the time by which the task should be completed. In some settings, for example, the playing of a video or other multimedia presentation, there may be natural deadlines for the various tasks imposed by the application. In other settings, the system may impose deadlines to better manage tasks or insure a certain quality of service to each task [12]. Yao *et al.* [37] assume that the system's performance measure is deadline feasibility; that is, each task must finish by its deadline.

They study the problem of minimizing the total energy used subject to the deadline feasibility constraints. Bansal, Kimbrel and Pruhs [7, 8] study the problem of minimizing the maximum temperature attained subject to the deadline feasibility constraints.

2.2 Energy and Temperature

Before proceeding further, we need to explain how speed, power, energy, and temperature are modeled, and how they are related. Yao *et al.* [37] assume a continuous function $P(s)$ such that if the device runs at speed s, then it consumes power at a rate of $P(s)$. For example, the well known cube-root rule for CMOS based devices states that the speed s is roughly proportional to the cube-root of the power P, or equivalently, $P(s) = s^3$, the power is proportional to the speed cubed [11]. Yao *et al.* [37] only assume that

$P(s)$ is strictly convex. This assumption implies that that the slower a task is run, the less energy is used to complete that task. Some simplicity of analysis, and little loss of applicability, comes from assuming that $P(s) = s^\alpha$ for some constant $\alpha > 1$.

The total energy used by the system is then $\int_0^\infty P(s(t))dt$ where $s(t)$ is the speed of the device at time t.

We now turn our attention to temperature. Cooling, and hence temperature, is a complex phenomenon that can not be modeled completely accurately by any simple model [33]. In [7], Bansal Kimbrel and Pruhs propose a model in which the environmental temperature is assumed to be constant. While this assumption certainly is not strictly true, the hope is that it is sufficiently close to being true that insight gained with this model will be useful in real settings. They also assume that the rate of cooling of the device adheres to Fourier's Law. Fourier's law states that the rate of cooling is proportional to the difference in temperature between the object and the environment. Without loss of generality one can scale temperature so that the environmental temperature is zero. A first order approximation for the rate of change T' of the temperature T is then $T' = aP - bT$, where P is the supplied power, and a, b are constants.

Some modern processors are able to sense their own temperature, and thus can be slowed down or shut down so that the processor temperature will stay below its thermal threshold [34]. If one views $\texttt{http}://\texttt{www.cs.pitt.edu}/\sim\texttt{kirk/cool.avi}$, this is the reason why the Pentium only slows down, and doesn't fry like the AMD processor.

Bansal and Pruhs [8] show that the maximum temperature is within a factor of 4 of a times the maximum energy used over any interval of length $\frac{1}{b}$. This observation also shows that there is a relationship between total energy and maximum temperature optimization and simplifies the task of reasoning about temperature. If the cooling parameter b is 0 then the temperature minimization problem becomes equivalent (within a constant factor) to the energy minimization problem. This also explains why some algorithms in the literature for energy management are poor for temperature management, that is, these algorithms critically use the fact that the parameter $b = 0$. If the cooling parameter b is ∞ then the temperature minimization problem becomes equivalent to the problem of minimizing the maximum power, or equivalently minimizing the maximum speed. We say that an algorithm is *cooling oblivious* if it is simultaneously $O(1)$-approximate for minimizing the maximum temperature for all values of a and b in the temperature equation $T' = aP - bT$, Thus a cooling oblivious algorithm is also $O(1)$-approximate for total energy and maximum speed/power. The energy minimization problem, when the speed to power parameter α is ∞ is also equivalent to minimizing the maximum power.

2.3 Energy Minimization with Deadline Feasibility

Yao, Demers and Shankar [37] study the problem of minimizing the total energy used to complete all tasks subject to the deadline feasibility constraints. They give an offline greedy algorithm (YDS) that optimally solves this problem. The algorithm YDS proceeds in a series of iterations. During each iteration, tasks in the maximum intensity interval are scheduled Earliest Deadline First at a speed equal to the intensity of this interval; the intensity of a time interval is defined to be the sum of the work requirements of all tasks whose release time and deadline are both contained within the interval divided by the length of an interval. The newly scheduled time interval is then blacked out, and all the remaining tasks must be executed during the remaining time that is not blacked out. It was shown by Bansal and Pruhs [8] that the energy optimality of the YDS schedule follows as a direct consequence of the well known KKT optimality conditions for convex programs.

Theorem 1. *The YDS algorithm is optimal for energy minimization.*

Proof. Consider a convex program

$$\begin{aligned} &\min f_0(x) \\ &\quad f_i(x) \;\; \leq 0 \qquad\qquad i = 1, \ldots, n \end{aligned}$$

Assume that this program is strictly feasible, that is, there is some point x where where $f_i(x) < 0$ for $i = 1, \ldots, n$. Assume that the f_i are all differentiable. Let λ_i, $i = 1, \ldots, n$ be a variable (Lagrangian multiplier) associated with the function $f_i(x)$. Then a necessary and sufficient KKT conditions for solutions x and λ to be primal and dual feasible are [10]:

$$\begin{aligned} f_i(x) &\leq 0 \qquad i = 1, \ldots, n && (1) \\ \lambda_i &\geq 0 \qquad i = 1, \ldots, n && (2) \\ \lambda_i f_i(x) &= 0 && (3) \\ \nabla f_0(x) + \sum_{i=1}^{n} \lambda_i \nabla f_i(x) &= 0 && (4) \end{aligned}$$

To state the energy minimization problem as a convex program, we break time into intervals $t_0, \ldots t_m$ at release times and deadlines of the tasks. Let $J(i)$ be the tasks that can feasibly be executed during the time interval $I_i = [t_i, t_{i+1}]$, and $J^{-1}(j)$ be intervals during which task j can be feasibly executed. We introduce a variable $w_{i,j}$, for $j \in J(i)$, that represents the work done on task j during time $[t_i, t_{i+1}]$. Our (interval indexed) mathematical program P is then:

$$\begin{aligned} \min \qquad & E && (5) \\ w_j \leq \; & \sum_{i \in J^{-1}(j)} w_{i,j} \qquad j = 1, \ldots, n && (6) \\ \sum_{i=1}^{m} \left(\frac{\sum_{j \in J(i)} w_{i,j}}{t_{i+1} - t_i} \right)^{\alpha} (t_{i+1} - t_i) \leq \; & E && (7) \\ w_{i,j} \geq \; & 0 \qquad i = 1, \ldots, m \qquad j \in J(i) && (8) \end{aligned}$$

By applying the KKT optimality conditions (see [8] for details) one can conclude that a sufficient condition for a primal feasible solution to be optimal is that:

- For each task j, the processor runs at the same speed, call it s_j, during the intervals i in which task j is run.
- And the processor runs at speed no less than s_j during intervals i, such that $j \in J(i)$, in which task j is not run.

The schedule produced bt the YDS algorithm clearly has these properties and hence is optimal. □

A naive implementation of YDS runs in time $O(n^3)$. This can be improved to $O(n^2)$ if the intervals have a tree structure [26]. It would be interesting to see if the cubic running time of YDS for arbitrary instances can be improved. For jobs with a fixed priority, Yun and Kim [39] show that it is NP-hard to compute an minimum energy schedule. They also give a fully polynomial time approximation scheme for the problem. Kwon and Kim [25] give a polynomial time algorithm for the case of a processor with discrete speeds.

In the online version of the problem, an algorithm only learns about a task at its release time, at which time it is given the exact work requirements of the job as well as its deadline. Yao *et al.* [37] define two simple online algorithms. The online algorithm Average Rate (AVR) runs each task in the optimal manner under the assumption that it is the only task in the system. That is, the work on each task is spread evenly between its release date and its deadline. The online algorithm Optimal Available (OA) at any point of time schedules the unfinished work optimally under the assumption that no more tasks will arrive. They give a lower bound of α^α on the approximation ratio for AVR and OA for energy minimization. In this instance $r_i = 0$, $d_i = i/n$ and $w_i = (n/i)^{\frac{\alpha+1}{\alpha}}$, for $i = 1, \dots, n$. They also prove, using a rather complicated spectral analysis, that the approximation ratio of AVR is at most $2^\alpha \alpha^\alpha$ for energy minimization. It was shown by Bansal, Kimbrel and Pruhs in [7] using a simple potential function argument, that OA is α^α-competitive with respect to energy minimization. That is, they show that at all times t, $P_{OA}(t) + \Phi'(t) \leq P_{Opt}(t)$, where $P_{OA}(t)$ is the power of OA at time t, $P_{Opt}(t)$ is the power of adversary at time t, and $\Phi'(t)$ is the change in a potential function $\Phi(t)$ that we define.

The general lower bounds in [37] on the competitive ratio, with respect to energy minimization, for an arbitrary algorithm are of the form $\Omega(c^\alpha)$ for some constant c. This suggests the question of whether an online algorithm exists that can achieve a competitive ratio that is $O(c^\alpha)$. Such an algorithm would be better than OA or AVR for large α. Bansal, Kimbrel and Pruhs [7] introduce an online algorithm and prove that it achieves such a competitive ratio. We refer to this algorithm as BKP. To explain the BKP algorithm, we need to first introduce some notation. Let $w(t, t_1, t_2)$ denote amount of work that has arrived by time t, that has release time $\geq t_1$ and deadline $\leq t_2$. Let $k(t)$ be the maximum over all $t' > t$ of $(w(t, et - (e-1)t', t'))/(e(t'-t))$. Note that $w(t, t_1, t_2)$ and $k(t)$ may be computed by an online algorithm at time t. At all times t, the BKP algorithm works at rate $e \cdot k(t)$ on the unfinished job with the earliest deadline. Intuitively, $k(t)$ is a lower bound, and a current estimate, of the speed of the optimal algorithm YDS at time t. The online algorithm has to run at a higher rate than $k(t)$ in case that more work arrives in the future, and its lower bound of $k(t)$ was too small. The constants are chosen to provide the best analysis.

Bansal, Kimbrel and Pruhs [7] show that the BKP is also e-competitive with respect to the maximum speed. Furthermore, this is optimal among deterministic online algorithms. That is, the objective is to minimize the maximum speed that the processor runs subject to the constraint that all jobs finish by their deadline. Therefore, BKP is also strongly e^α-competitive with respect to maximum power. In the instance that establishes this lower bound, the adversary works at a rate $a(t) = \frac{-1}{(\ln x)(1-t)}$ from time 0 to time $1 - x$. We look at the limit of the resulting instances as x goes to 0. Work is released at the rate that the adversary does the work, and the deadline of all work is $1 - x$. Understanding this instance is useful in understanding the underlying motivation for the definition of the BKP algorithm.

2.4 Maximum Temperature Minimization with Deadline Feasibility

Bansal, Kimbrel and Pruhs show in [7] that in principle the problem of minimizing maximum temperature, subject to deadline feasibility, can be stated as a convex program as follows. We break time into intervals $t_0, \dots t_m$ at release dates and deadlines of the jobs. We introduce a variable T_i that represents $T(t_i)$, the temperature at time t_i. Let $J(i)$ be the jobs j that can feasibly be executed during time $[t_i, t_{i+1}]$, that is, $r_j < t_{i+1}$ and $d_j > t_i$. We introduce a variable $W_{i,j}$, for $j \in J(i)$, that represents the work done on job j during time $[t_i, t_{i+1}]$. Let $MaxW(x, y, X, Y)$ be the maximum work that can be done starting at time x at temperature X and ending at time y at temperature Y subject to the temperature constraint $T \leq T_{max}$ throughout the interval $[x, y]$. Let $MaxT(x, y, X, y)$ be a corresponding temperature curve. Let $UMaxW(x, y, X, Y)$ and $UMaxT(x, y, X, Y)$ be similarly defined except that there is no maximum temperature constraint. We can then express the temperature problem as:

$$
\begin{aligned}
p_j &\leq \sum_{i:j\in J(i)} W_{i,j} \qquad 1 \leq j \leq n && (9)\\
\sum_{j\in J(i)} W_{i,j} &\leq MaxW(t_i, t_{i+1}, T_i, T_{i+1}) \\
&\qquad\qquad 1 \leq i \leq m-1 && (10)\\
0 &\leq T_i \qquad 1 \leq i \leq m && (11)\\
0 &\leq W_{i,j} \qquad 1 \leq i \leq m, 1 \leq j \leq n && (12)
\end{aligned}
$$

To solve this problem in polynomial time using the Ellipsoid algorithm, one needs to give an efficient algorithm for computing separating hyperplanes. For this problem, this boils down to understanding the temperature curves $MaxT$ and $UMaxT$. Such a problem falls under the rubric of calculus of variations [35]. Let $F = ((T' + bT)/a)^{1/\alpha}$. The fundamental theorem of calculus of variations states that the solution to our problem (without the thermal threshold) follows an Euler-Lagrange curve that satisfies $F_T - \frac{d}{dt}F_{T'} = 0$. Solving this using the standard Laplace transform technique, we get that $T = ce^{-bt} + de^{-\alpha bt/(\alpha-1)}$, where the constants c and d are determined by the boundary conditions. One can show that the optimal temperature curve T is either an Euler-Lagrange curve for the unconstrained problem, or follows an Euler-Lagrange curve up to T_{max}, stays at T_{max} for a while, and then cools to the final temperature along an Euler-Lagrange curve. After some work, this gives sufficient information to efficiently compute a separating hyperplane

Bansal and Pruhs show [8] that while the YDS schedule may not be optimal for temperature, YDS is cooling oblivious (recall, this means $O(1)$-approximate for temperature). Thus, this constructively shows that there are schedules that achieve an $O(1)$-approximation ratio with respect to both of the dual criteria: temperature and energy.

Bansal and Pruhs show [8] that the online algorithms OA and AVR, proposed in [37] in the context of energy management, are not $O(1)$-competitive with respect to temperature. Recall that these algorithms are $O(1)$-competitive with respect to energy. This illustrates analytically the observation from practice that power management techniques that are effective for managing energy may not be effective for managing temperature. They also show in [8] that the most intuitive temperature management algorithm, running at such a speed so that the temperature is constant, is surprisingly not $O(1)$-competitive with respect to temperature. In contrast, they show that the online speed scaling algorithm BKP is cooling oblivious. The temperature analysis of the online algorithm BKP compares BKP to YDS, and thus the temperature analysis of YDS was a necessary component to the temperature analysis of BKP. Note that as BKP is cooling oblivious, the $O(1)$-competitiveness of BKP for total energy minimization and maximum speed minimization are essentially a corollary of this result (although better constants can be obtained by doing a direct analysis [7]).

2.5 Flow Time

We can take any scheduling problem and consider an energy management version, or a temperature management version. We would then get a dual optimization problem, where the dual objectives are the original quality of service objective for the scheduling problem, and the second objective was either energy or temperature. The simplest way to get results for dual objective problems is to bound one objective and optimize for the other. It seems more natural here to bound energy or temperature, and to optimize the quality of service objective. In the case of temperature, it is reasonable to assume that that thermal threshold of the device is known, and one needs a schedule that keeps the temperature below the thermal threshold. In the case of energy, it is reasonable to assume that the energy stored in the battery is known, and one needs a schedule that doesn't deplete the battery.

The one paper that we are aware of that considers a power management problem for a scheduling objective other than deadline feasibility is [30]. In [30], Pruhs, Uthaisombut and Woeginger consider the problem of minimizing the total flow time subject to a bound E on the total energy used. The flow time of a task i is $F_i = C_i - r_i$, which is the difference between the completion time of the process and the release time of the process. From a mathematical point of view, it is easier to deal with deadline feasibility than flow time since the deadlines restrict the possible ways and times that energy might reasonably be spent. To try to get some handle on flow time, Pruhs *et al.* [30] restricted their attention to the case that all tasks have the same amount of work. This allowed them to decouple the scheduling policy from the speed setting policy, and concentrate our efforts on understanding speed setting alone. In the case of equi-work tasks, it can be easily shown that the optimal scheduling policy is First-Come-First-Served (FCFS).

This problem may then be expressed as a convex program P in the following manner:

$$\min \sum_{i=1}^{n} C_i \tag{13}$$

$$\sum_{i=1}^{n} \frac{1}{x_i^{p-1}} \leq E \tag{14}$$

$$C_{i-1} + x_i \leq C_i \qquad i = 2, \ldots, n \tag{15}$$

$$r_i + x_i \leq C_i \qquad i = 1, \ldots, n \tag{16}$$

$$x_i \leq \frac{1}{E^2} \qquad i = 1, \ldots, n \tag{17}$$

By applying the KKT optimality conditions, one can show that a feasible solution is optimal if:

- A total energy of E is used.
- Job n, and each job i, with $C_i < r_{i+1}$, are run with the same power, call it ρ.
- Each job i, with $C_i > r_{i+1}$, is run with power ρ more than the power of job $i+1$.
- Each job i, with $C_i = r_{i+1}$, is run with power at least ρ and at most the power of job $i+1$ plus ρ.

The problem is that the KKT conditions do not provide any real information on the value of ρ, although it is easy to see that ρ is a monotone function of E. Using these observations, Pruhs *et al.* [30] show that a relatively simple hill-climbing algorithm can solve this problem quite efficiently, on the order of $O(n^2 poly - log)$. It is easy to compute the optimal schedule for some very high energy E. Their algorithm then follows the optimal schedule as the available energy E decreases (or equivalently as ρ decreases).

Intuitively, the less energy available, the more slowly the tasks should be run in the optimal schedule. Surprisingly, it is shown in [30] that this intuition is not always correct. That is, as energy decreases, some tasks will actually be run at a higher speed in the optimal schedule. Thus, none of the obvious properties of a task (speed, energy used, etc.) are monotone functions of energy. However, luckily, there are only $O(n)$ structural changes in the optimal schedule, and the structural phase transitions are very well behaved. For example, while jobs speeds are not monotone functions of the available energy E, they are smooth/continuous functions of E.

2.6 Open Problems

We discuss some of the most obvious questions left open in the research to date.

A Simple Tight Competitive Analysis of AVR: The analysis of the AVR algorithm in [37] that does not achieve a tight bound is a rather complicated spectral analysis, which does not so easily yield an intuitive explanation of AVR's performance. It would be nice to provide a simple tight competitive analysis of AVR. We agree with Yao, Demers and Shankar who state in [37] that AVR is likely to α^α competitive. It seems likely that AVR will be the solution adopted in many technologies because of its simplicity and seeming fairness. Thus we think that it is important to better understanding of AVR scheduling.

A better analysis of the competitive ratio of BKP with respect to energy: There is significant evidence that the online BKP algorithm is the "right" power management algorithm. However, the upper bound on the competitive ratio of BKP with respect to energy is clearly not tight in an obvious way. As α goes to 1, the competitive ratio of every reasonable speed scaling algorithm goes to 1. This is because when $\alpha = 1$ there is no inefficiency when running at a higher speed. The current upper bound on the competitive ratio of BKP does not have this property. The main difficulty in improving the analysis for small α is that BKP needs to be credited with energy saved when there are no jobs to be run.

A Simple and Efficient Polynomial Time Offline Algorithm for Temperature: The Ellipsoid algorithm is useful for establishing theoretical results, but it is practically inefficient for moderate sized problems. The KKT conditions are not of any obvious help for this problem. There is some reason to think that perhaps a dynamic programming solution may be possible.

Computing an Optimal Flow Time Schedule for Arbitrary Sized Tasks with an Energy Constraint: The optimal scheduling policy is always to run the task with least remaining work. However, this does not completely specify a scheduling policy as the scheduler must also decide at what speed to run the selected task. One reason our task is more complicated for arbitrary sized jobs is that the job with the least remaining work depends on the prior scheduling decisions. This is in contrast to FCFS, the optimal scheduling policy for uniform-work jobs, in which scheduling priorities are independent of the speed at which previous jobs have been tun. The main difficulty is that with arbitrary sized tasks, none of the interesting parameters of the optimal schedule are necessarily continuous functions of the available energy E. For the uniform job case, the algorithm in [30] started with an optimal schedule for a very large energy E and decreased E while tracking the changes to the schedule. In the non-uniform case, the optimal schedule can change radically with an infinitesimal change in E, making the changes difficult or potentially impossible to track. It is possible to get a positive result using resource augmentation [23, 29]. That is, one can get an $(1 + \epsilon)$-speed $O(1)$-approximate polynomial time algorithm [5].

Online Algorithms for Flow Time: In most situations, the system does not know of a task until its release time. Thus we would like to develop and analyze online algorithms for this problem. The first challenge is to formalize the problem so that one can obtain insightful results. All of the most obvious formulations do not give interesting results, i.e. you get trivial impossibility results.

3 Power-Down strategies to Conserve Energy

Another mechanism that is commonly used to conserve energy is to eliminate or reduce power to one or more components. At the architectural level, the technique of eliminating power to a functional component is called clock/power gating. Power gating is the only way to combat leakage loss, which is the energy lost independent of the device's state. It is predicted that leakage loss will become comparable to switching loss in the near term future [9]. Thus, power-gating strategies will become increasingly important at the architectural level. At a higher level, the powered-down component might be a disk drive or even the whole system (e.g., a laptop that hibernates). Furthermore, the device may have multiple power-down states. For example, the Advanced Configuration and Power Interface (ACPI) included in the BIOS on most newer computers has five power states, including hibernation and three levels of standby [2]. Generally, it is the case that bringing a powered-down component back to the active state requires additional energy and time before an incoming task can be serviced. *Dynamic Power Management* (DPM) in the systems literature refers to the problem of developing policies for powering down based on the dynamically changing system state, functionality and timing requirements [17, 19]. We will use the more descriptive term Power-Down (PD) for this problem.

There are numerous examples of systems that can be run at multiple speeds, have a sleep state, and receive tasks with deadlines. One example is the Rockwell WINS node, which is a mobile sensing/computing node with onboard environmental sensors [3, 4]. To fully utilize the power-saving capabilities of such devices, it is necessary to design policies that effectively combine speed scaling and alternate power-down states. We call this problem Speed Scaling with Power-Down (SS-PD). Combining speed scaling and power-down introduces challenges that do not appear in either of the original problems. For example, with just speed scaling, it is always in the best interest of the scheduler to run tasks as slowly as possible within the constraints of the release times and deadlines, due to the convexity of the power function. By contrast, in SS-PD, it may be beneficial to speed up a task in order to create an idle period in which the system can power-down.

Powering down devices will also be an important mechanism in minimizing energy consumption in sensor networks. Researchers have observed that a large fraction of wasted energy happens when sensors are idle [38, 15, 13] and have turned their attention to research issues related to powering down sensors when not receiving or transmitting. Different researchers assume different models as to how this can be achieved. Some examine models in which sensors can not be woken up. Instead, they set a timer before putting themselves to sleep and can only wake-up when the timer expires [38]. Others assume that there is a mechanism for a sensor to listen for messages at very lower power-consumption rate [15]. Then when a node senses that a message is being transmitted, it transitions to a waking state.

3.1 Previous Research

In [17, 19] Irani, Gupta and Shukla formalize Power-Down in the following manner. There are periods of unknown duration, during which there are no tasks to run and the device can be powered down. These idle periods end with the arrival of a critical task. The decision that the online algorithm has to make is when to transition to a lower power state, and what state to transition to. The power-down states are denoted by $\{s_0, ..., s_k\}$, with associated decreasing power consumption rates of $\{P_0, ..., P_k\}$. At the end of the idle period, the device must be returned back to the active state s_0. There is an associated transition energy e_{ij} and transition time t_{ij} to move from state s_i to s_j. The goal is to minimize the energy used over the idle period. Algorithms are typically evaluated according to a competitive ratio. When a probability distribution governing the length of the idle periods is assumed, one seeks only to minimize the expected cost. We call algorithms in this latter category *probability-based*.

The power-down problem for two states is a continuous version of the well-known Ski Rental Problem [18], and is well understood. There is a simple 2-competitive algorithm, which stays in the active state until the total energy consumed is equal to the transition energy. It is known that this algorithm is optimally competitive. Furthermore, if the idle period is generated by a known probability distribution, then there is a probability-based algorithm that is $(e/e-1)$-competitive [24], and this is optimally competitive. For some systems, like disk drives, the energy needed and time spent to go from a higher power state to a lower power state is negligible. Irani *et al.* show in [17] that for such systems, the two-state deterministic and probability-based algorithms can be generalized to systems with multiple sleep states so that the same competitive ratios can be achieved. The probability-based algorithm requires knowledge of a probability distribution that generates the length of the idle period. In [19] Irani *et al.* give an efficient heuristic for learning and summarizing the probability distribution based on recent history and show that this heuristic combined with the competitive probability-based policy outperforms other methods suggested in the systems literature on data from a mobile IBM hard drive with three sleep states [40].

In the cases where powering down requires a non-negligible cost, it may be costly to transition to many states as the device powers down and it may be advantageous to skip some sleep states. In this scenario, an algorithm must identify an optimal sequence of states as well as the time thresholds at which these states will be reached. Augustine, Irani and Swamy [6] develop a deterministic algorithm that achieves a competitive ratio of 8 for any system with arbitrary transition costs. The bound improves to $3 + 2\sqrt{2} \approx 5.8284$ under the reasonable assumption that $e_{ij} \leq e_{lj}$ whenever $l \leq i \leq j$. For multiple state systems, the optimal competitive ratio that can be achieved will, in general, be a complicated function of the parameters of the system. Thus, we have developed a meta-algorithm that takes as input the description of a system (set of states and associated parameters) and an error rate ϵ, and produces a power-down policy (set of states and thresholds) whose competitive ratio is within ϵ of the best possible that can be achieved for that system. The algorithm runs in time polynomial in $1/\epsilon$ and k, the number of states. This is the first mechanism that has been developed to get provably tight bounds for the optimal competitive ratio for a general multi-state system. In addition, we extended the probability-based method in [17] to give an algorithm that takes as input a probability distribution over the length of the next idle time and a description of the system and outputs a power-down strategy that minimizes the expected energy consumption.

In [16] we (joint with Rajesh Gupta and Sandeep Shukla) initiated a theoretical investigation into Speed Scaling with Power-Down (SS-PD). This work assumes two states, an active state and a sleep state. The device can run at an arbitrary speed in the active state, but the power consumption rate is a convex function of the speed. When the device is in the active state and the speed is 0, power is still being consumed. The sleep state consumes no power, but there is a fixed cost to transition from the sleep state to the active state. We give an offline algorithm that produces a schedule whose total energy consumption for any set of tasks is within a factor of two of optimal. We also give an online algorithm A that makes use of an online algorithm B for pure speed scaling. If B is $O(1)$-competitive, and satisfies some very reasonable technical conditions, then A is $O(1)$-competitive. The exact relationship is a bit complex, but in the case of the cube-root rule, the competitive ratio of A for SS-PD is approximately three times the competitive ratio of B for speed scaling.

Central to this work was the identification of the concept of a critical speed, which is the speed at which it is no longer beneficial to increase the speed of a task so as to create an idle period in which the device can sleep. Work in the systems literature has further explored this idea in order to understand where this critical speed lies based on different leakage models [21]. All of the algorithms developed for SS-PD operate on the principle that if a device is shut down, it will continue to remain in that state, even if there are pending jobs, until waiting any longer would require running faster than the critical speed in order to complete all jobs by their deadlines. This idea of 'procrastination scheduling' has been studied further in [20] who examine the combined SS-PD problem under different assumptions about how jobs can be scheduled.

3.2 Open Problems

There are still important research questions that remain open in this area.

Study the Trade-off Between Energy and Latency in Multiple Idle Periods: The focus in work on competitive analysis has been on minimizing energy. However, when a system powers down, it takes some additional time (as well as energy) to bring the system back into an active state in order to begin work on a newly arrived task. For example, the IBM mobile hard drive used in the experimental studies in [19] requires 1.5 seconds to transition from the stand-by to the active state. As any laptop user, who has twiddled her thumbs while waiting for her laptop to return from an unfortunately timed transition to power saving mode knows, this transition latency can result in a significant degradation in performance as seen by the user. Looking at the extreme cases, if one simply wanted to minimize latency, the system would stay constantly in the active state. If one simply wanted to minimize energy, the system would stay powered down for long periods of time and only transition to the active state when enough tasks have accumulated. (Although, technically, such a strategy would not be allowed since the device is not allowed to be idle if there are pending jobs).

In the context of sensor networks, the energy and time for the wake-up can also be a significant cost, potentially even more than the cost of receiving and transmitting the message [13]. In cases where the transmitted message is time-critical, it will be important to consider the latency incurred in the start-up, especially if this delay accumulates through multiple hops in the network. A thorough understanding of the tradeoff between latency and energy use would enable systems and users to make an informed decision as to which resource is more critical to optimize.

There are two important directions to consider. The first is simply to look at a single idle period generated by a probability distribution. Given a fixed bound on the energy to be expended during the course of the idle period, how can we best minimize latency? Note that unlike all the previous versions of this problem, it may be advantageous to transition to the active state in a predictive manner, before the arrival of a new task. Depending on the energy bound and the probability distribution over the idle period length, it may even be worthwhile to transition back and forth between the two states more than once.

Typically, however, a system does not have a single period in isolation, but rather a series of idle periods separated by the arrival of tasks to be performed. When transition times are zero, the problem is the same when we consider each idle period in isolation. However, when transition latencies are non-negligible, the time incurred in making a transition to the active state will serve to delay the execution of tasks which will in turn shorten future idle periods. The interplay between power-down policies and latency has been examined experimentally in [31, 32] in the context of disk-drives. We would like to understand more formally, how the choice of power-down policy effects latency over a series of idle periods and develop algorithms which allow a user or system to preferentially choose between optimizing one resource over the other.

Offline Complexity of SS-PD: The most intellectually intriguing question related to SS-PD is whether the offline problem is NP-hard. In fact, it is not known (despite serious effort by good researchers) whether the following simplified version of the problem is NP-hard. The device can be in one of two power states: active or sleep. The active state consumes power at some fixed known rate and the sleep state consumes no power. Tasks must be run in the active state. It costs one unit of energy to transition from the sleep state to the full power state. The transition time is assumed to be negligible. Informally, the problem is to schedule the tasks in such a way that creates fewer, longer idle periods, during which the device can be put into the sleep state. Many seemingly more complicated problems in this area can be essentially reduced to this problem, so a polynomial time algorithm for this problem would have wide application.

4 Conclusion

We believe that there are many interesting algorithmic problems, with potential real application, in the domain of power management. We hope that this survey will convince other researchers of the validity of our belief, and encourage them to work on problems in this area.

References

[1] http://www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/SmartDust/.

[2] http://www.microsoft.com/windows2000/techenthusiast/features/standby1127.asp.

[3] http://www2.parc.com/spl/projects/cosense/csp/ slides/Srivastava.pdf.

[4] http://www.rockwellscientific.com/hidra/.

[5] J. Augustine, S. Irani, K. Pruhs, and P. Uthaisombut. unpublished manuscript.

[6] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. In *Foundations of Computer Science*, 2004.

[7] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *IEEE Syposium on Foundations of Computer Science*, 2004.

[8] N. Bansal and K. Pruhs. Speed scaling to manage temperature. In *Symposium on Theoretical Aspects of Computer Science*, 2005.

[9] Shekhar Borkar. Design challenges of technology scaling. *IEEE Micro*, 19(4):23–29, 1999.

[10] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[11] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[12] Giorgio Buttazzo. *Hard Real-Time Computing Systems*. Kluwer, 1997.

[13] Anantha Chandrakasan, Rex Min, Manish Bhardwaj, Seong-Hwan Cho, , and Alice Wang. Power aware wireless microsensor systems. In *European Solid-State Circuits Conference*, 2002.

[14] Carla Schlatter Ellis. The case for higher-level power management. In *IEEE Workshop on Hot Topics in Operating Systems*, 1999.

[15] C. Guo, L. C. Zhong, and J. M. Rabaey. Low power distributed mac for ad hoc sensor radio networks. In *Proceedings of IEEE GlobeCom*, 2001.

[16] S. Irani, , R. K. Gupta, and S. Shukla. Algorithms for Power Savings. In *ACM/SIAM Symposium on Discrete Algorithms*, 2003.

[17] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In *IEEE Conference on Design, Automation and Test in Europe*, 2002.

[18] S. Irani and A. Karlin. Online computation. In Dorit Hochbaum, editor, *Approximations for NP-Hard Problems*. PWS Publishing Co., 1995.

[19] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Online strategies for dynamic power management in systems with multiple power saving states. *Trans. on Embedded Computing Sys.*, 2003. Special Issue on Power Aware Embedded Computing.

[20] R. Jejurikar and R. Gupta. Procrastination scheduling for fixed priority real-time systems. In *Proc. of ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 2004.

[21] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware energy efficient task scheduling in embedded real-time systems. In *Proceedings of the Design Automation Conference*, 2004.

[22] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for smart dust. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, 1999.

[23] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[24] A Karlin, M. Manasse, L. McGeoch, and S. Owicki. Randomized competitive algorithms for non-uniform problems. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 301–309, 1990.

[25] W. Kwon and T. Kim. Optimal voltage allocation techniques for dynamically variable voltage processors. In *Design Automation*, 2003.

[26] Minming Li, Becky Jie Liu, and Frances F. Yao. Min-energy voltage allocation for tree-structured tasks. In *International Computing and Combinatorics Conference*, 2005.

[27] John Markov. http://www.iht.com/articles/520233.html.

[28] Trevor Mudge. Power: A first-class architectural design constraint. *Computer*, 34(4):52–58, 2001.

[29] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook on Scheduling*. CRC Press, 2004.

[30] Kirk Pruhs, Patchrawat Uthaisombut, and Gerhard Woeginger. Getting the best response for your erg. In *Scandanavian Workshop on Algorithms and Theory*, 2004.

[31] D. Ramanathan, S. Irani, , and R. K. Gupta. Latency Effects of System Level Power Management Algorithms. In *IEEE International Conference on Computer-Aided Design*, 2000.

[32] D. Ramanathan, S. Irani, and R. Gupta. An Analysis of System Level Power Management Algorithms and their effects on Latency. *IEEE Trans. on Computer Aided Design*, 21(3), march 2002.

[33] Jerry E. Sergent and Al Krum. *Thermal Management Handbook*. McGraw-Hill, 1998.

[34] Kevin Skadron, Mircea R. Stan, Wei Huang, Sivakumar Velusamy, Karthik Sankaranarayanan, and David Tarjan. Temperature-aware microarchitecture. In *International Symposium on Computer Architecture*, pages 2–13, 2003.

[35] Donald R. Smith. *Variational Methods in Optimization*. Prentice-Hall, 1974.

[36] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez. Reducing power in high-performance microprocessors. In *Design Automation Conference*, pages 732–737, 1998.

[37] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *IEEE Syposium on Foundations of Computer Science*, page 374, 1995.

[38] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings 21st International Annual Joint Conference of the IEEE Computer and Communications Societies*, 2002.

[39] H.S. Yun and J. Kim. On energy-optimal voltage scheduling for fixed priority hard real-time systems. *ACM Transactions on Embedded Computing Systems*, 2(3):393–430, 2003.

[40] *Technical specifications of hard drive IBM Travelstar VP 2.5inch, available at.* http://www.storage.ibm.com/storage/oem/data/travvp.htm, 1996.