Algorithms for Energy Saving[⋆]

Susanne Albers[⋆⋆]

Department of Computer Science, University of Freiburg
Georges Koehler Allee 79, 79110 Freiburg, Germany
salbers@informatik.uni-freiburg.de

Abstract. Energy has become a scarce and expensive resource. There is a growing awareness in society that energy saving is a critical issue. This paper surveys algorithmic solutions to reduce energy consumption in computing environments. We focus on the system and device level. More specifically, we study power-down mechanisms as well as dynamic speed scaling techniques in modern microprocessors.

Keywords: Dynamic speed scaling, power-down mechanisms, scheduling, competitive analysis, probabilistic analysis, approximation algorithms.

1 Introduction

With increasing CPU clock speeds and higher levels of integration in processors, memories and controllers, power consumption has become a major concern in computer system design over the past years. Power dissipation is critical in battery operated mobile computing devices that have proliferated in recent years. In these devices, obviously, the amount of available energy is severely limited. Moreover, power consumption is a major concern in desktop computers and servers. Electricity costs impose a substantial strain on the budget of data and computing centers, where servers and, in particular, CPUs account for 50–60% of the energy consumption. In fact, Google engineers, maintaining thousands of servers, recently warned that if power consumption continues to grow, power costs can easily overtake hardware costs by a large margin [11]. In addition to cost, energy dissipation causes thermal problems. Most of the consumed energy is converted into heat, resulting in wear and reduced reliability of hardware components.

For these reasons, there has recently been considerable research interest in the design and analysis of *energy-efficient algorithms* that reduce the energy consumption while minimizing compromise to service. This survey focuses on energy saving mechanisms on the system and device level. In this context, there are basically two techniques to save energy.

[⋆] An extended and modified version of this survey, aiming at a different audience, will appear in the *Communications of the ACM.*

[⋆⋆] Work supported by a Gottfried Wilhelm Leibniz Award of the German Research Foundation.

(1) *Power-down mechanisms*: When a system is idle, it can be transitioned into low-power standby or sleep states. This technique is well-known and widely used to save energy. One has to find out when to shut down a system, taking into account that a transition back to the active mode requires extra energy.
(2) *Speed scaling*: Microprocessors currently sold by chip makers such as AMD and Intel are able to operate at variable speed. The higher the speed, the higher the power consumption is. The goal is to save energy by utilizing the full speed/frequency spectrum of a processor and applying low speeds whenever possible.

The power management problems described above are *online problems* in that a system is usually not aware of future events. A power-down mechanism, during an idle period, usually has no information when the period ends. Is it worthwhile to move to a lower-power state and benefit from the reduced energy consumption, given that the system must finally be powered up again at a cost to the active mode? A speed scaling algorithm typically does not know future jobs. Should lower speed levels be used at the expense of delaying the service of tasks that may arrive in the near future?

Despite the handicap of not knowing the future, an online strategy should achieve a provably good performance. Here we resort to *competitive analysis* [29], where an online algorithm ALG is compared to an optimal offline algorithm OPT that knows the entire future and can compute an optimal solution. Online algorithm ALG is called *c-competitive* if, for every input, the total energy consumption of ALG is at most c times that of OPT.

In this survey we first present the most important results known for power-down mechanisms. Then we address dynamic speed scaling algorithms.

2 Power-Down Mechanisms

Power-down mechanisms are a common technique to save energy. We encounter them on an every day basis. The display of our desktop turns off after some period of inactivity. Our laptop transitions to a standby or hibernate mode if it has been idle for a while. In these settings, there usually exist idleness thresholds that specify the length of time after which a system is powered down. From an algorithmic point of view, we would like to design strategies that determine such thresholds and perform well relative to the optimum.

Formally, we are given a device that always resides in one of several states. In addition to the active state, there can be several standby and sleep modes. These states have individual power consumption rates. The energy incurred in transitioning the system from a higher-power to a lower-power state is usually negligible. However, a power-up operation consumes a significant amount of energy. Over time the device experiences an alternating sequence of active and idle periods. During active periods, the system must reside in the active mode to perform the required tasks. During idle periods, the system may be moved to lower-power states. An algorithm has to decide when to perform the transitions and to which states to move. The goal is to minimize the total energy consumption. As the energy consumption during the active periods is fixed, assuming that prescribed tasks have to be performed, we concentrate on energy minimization in the idle intervals. In fact, we focus on any idle period and optimize the energy consumption in any such time window.

In the following we will first study systems that consist of two states only. Then we will address systems with multiple states. We stress that we consider the minimization of energy. We ignore the delay that arises when a system is transitioned from a lower-power to a higher-power state.

2.1 Systems with Two States

Consider a two-state system that may reside in an active state or in a sleep state. We assume without loss of generality that the power consumption rate in the active state is 1, i.e. the system consumes one energy unit per time unit. The power consumption rate in the sleep mode is 0. The results we present in the following generalize to arbitrary consumption rates. Suppose that β, $\beta > 0$, energy units are required to transition the system from the sleep state to the active state. The energy of transitioning from the active to the sleep state is assumed to be 0. If this is not the case, we can simply fold the corresponding energy into the cost of β incurred in the next power-up operation. The system experiences an idle period whose length T is initially unknown.

We first observe that an optimal offline algorithm OPT, knowing T in advance, is simple to formulate. If the value of T, counted in time units, is smaller than the value of β, OPT remains in the active state throughout the idle period. If T is at least β, OPT transitions to the sleep state right at the beginning of the idle period and powers up to the active state at the end of the period.

The following deterministic online algorithm mimics the behavior of OPT.

Algorithm ALG-D: In an idle period, remain in the active state first. After β time units, if the period has not ended yet, transition to the sleep state.

Theorem 1. *ALG-D is 2-competitive and this is the smallest competitiveness a deterministic online algorithm can achieve.*

Proof. We first analyze $ALG\text{-}D$ and consider two cases. If the value of T is smaller than the value of β, then $ALG\text{-}D$ consumes T units of energy during the idle interval and this is in fact equal to the consumption of OPT. If T is at least β, then $ALG\text{-}D$ first consumes β energy units to remain in the active state. An additional power-up cost of β is incurred at the end of the idle interval. Hence, $ALG\text{-}D$'s total cost is 2β, while OPT incurs a cost of β for the power-up operation at the end of the idle period.

We next verify that no deterministic online algorithm can achieve a competitive ratio smaller than 2. If an algorithm transitions to the sleep state after exactly t time units, then in idle period of length t it incurs a cost of $t + \beta$ while OPT pays $\min\{t, \beta\}$ only. □

We remark that power management in two-state systems corresponds to the famous ski-rental problem, a cornerstone problem in the theory of online algorithms, see e.g. [16].

Interestingly, it is possible to beat the competitiveness of 2 using randomization. A randomized algorithm transitions to the sleep state according to a probability density function $p(t)$. The probability that the system powers down during the first t_0 time units of an idle period is $\int_0^{t_0} p(t)dt$. Karlin et al. [20] determined the best probability distribution.

Algorithm ALG-R: Transition to the sleep state according to the probability density function

$$p(t) = \begin{cases} \frac{1}{(e-1)\beta} e^{t/\beta} & 0 \leq t \leq \beta \\ 0 & \text{otherwise.} \end{cases}$$

Theorem 2. [20] *ALG-R achieves a competitive ratio of $\frac{e}{e-1}$, and this is the smallest competitive ratio a randomized strategy can achieve.*

Here $e \approx 2.71$ is the Eulerian number and hence $\frac{e}{e-1} \approx 1.58$, which is considerably below the deterministic bound of 2.

From a practical point of view, it is also instructive to study stochastic settings where the length of idle periods is governed by probability distributions. In practice, short periods might occur more frequently. Probability distributions can also model specific situations where either very short or very long idle periods are more likely to occur, compared to periods of medium length. Of course, such a probability distribution may not be known in advance but can be learned over time.

Let $Q = (q(T))_{0 \leq T < \infty}$ be a fixed probability distribution on the length T of idle periods. For any $t \geq 0$, the deterministic algorithm ALG_t that always powers down after exactly t time units incurs an expected cost of

$$E[ALG_t(Q)] = \int_0^t Tq(T)dT + (t+\beta) \int_t^\infty q(T)dT \tag{1}$$

on idle periods generated according to Q. Karlin et al. [20] proposed the following strategy to handle probabilistic settings.

Algorithm ALG-P: Given a fixed Q, let A_Q^* be the deterministic algorithm ALG_t that minimizes (1).

Theorem 3. [20] *For any Q, the expected energy consumption of ALG-P is at most $\frac{e}{e-1}$ times the expected optimum consumption.*

2.2 Systems with Multiple States

Many modern devices do not have only one but several low-power states. Specifications of such systems are given, for instance, in the Advanced Configuration and Power Management Interface (ACPI) that establishes industry-standard interfaces enabling power management and thermal management of mobile, desktop

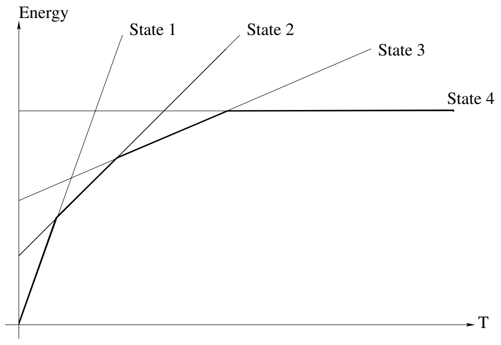

Fig. 1. Illustration of the optimum cost in a four-state system

and server platforms. A description of the ACPI power management architecture built into Microsoft Windows operating systems can be found at [1].

Consider a system with ℓ states $s_1, \ldots, s_\ell$. Let r_i be the power consumption rate of s_i. We number the states such that $r_1 > \ldots > r_\ell$. Hence s_1 is the active state and s_ℓ represents the state with lowest energy consumption. Let β_i be the energy required to transition the system from s_i to the active state s_1. We assume again that transitions from higher-power to lower-power states incur 0 cost because the corresponding energy is usually negligible. The goal is to construct a state transition schedule minimizing the total energy consumption in an idle period.

Irani et al. [17] presented online and offline algorithms. They first observe that the total energy incurred by an optimal offline algorithm *OPT* in an idle period of length T is given by

$$OPT(T) = \min_{1 \leq i \leq \ell} \{r_i T + \beta_i\}.$$

Hence, *OPT* chooses the state that allows for the smallest total cost consisting of energy consumption in the period and final power-up cost. Interestingly, the optimal cost has a simple graphical representation, see Figure 1. If we consider all linear functions $f_i(t) = r_i t + \beta_i$, then the optimum energy consumption is given by the lower envelope of the arrangement of lines.

We can use this lower envelope to guide an online algorithm which state to use at any time. Let $S_{OPT}(t)$ denote the state used by *OPT* in an idle period of total length t, i.e. $S_{OPT}(t)$ is the state $\arg\min_{1 \leq i \leq \ell}\{r_i t + \beta_i\}$. The following algorithm, proposed in [17], traverses the state sequence as suggested by the optimum offline algorithm.

Algorithm Lower-Envelope: In an idle period, at any time t, use state $S_{OPT}(t)$.

Intuitively, over time, *Lower-Envelope* visits the states represented by the lower envelope of the functions $f_i(t)$. If currently in state s_{i-1}, the strategy transitions to the next state s_i at time t_i, where t_i is the solution to the equation

$r_{i-1}t + \beta_{i-1} = r_i t + \beta_i$. Here we assume that states whose functions do not occur on the lower envelope, at any time, are discarded. Note that the algorithm is a generalization of *ALG-D* for two-state systems.

Theorem 4. [17] *Lower-Envelope is 2-competitive.*

The competitiveness of 2 is the smallest ratio achievable by a deterministic online algorithm.

Irani et al. [17] also studied the setting where the length of idle periods is generated by a probability distribution $Q = (q(T))_{0 \le T < \infty}$. They determined the time t_i when an online strategy should move from state s_{i-1} to s_i, $2 \le i \le \ell$. Let t_i be the time t that minimizes

$$\int_0^t r_{i-1} T q(T) dT + \int_t^\infty q(T)(r_{i-1}t + (T-t)r_i + \beta_i - \beta_{i-1}) dT.$$

Intuitively, the above expression is the expected cost of a deterministic algorithm ALG_t that powers down after t time units, assuming that only states s_{i-1} and s_i are available.

Algorithm ALG-P(ℓ): Change states at the transition times $t_2, \ldots, t_\ell$ defined above.

Note that *ALG-P(ℓ)* is a generalization of *ALG-P* for two-state systems.

Theorem 5. [17] *For any fixed probability distribution Q, the expected energy consumption of ALG-P(ℓ) is at most $\frac{e}{e-1}$ times the expected optimum consumption.*

Furthermore, Irani et al. presented an approach how to learn an initially unknown Q. They combined the approach with *ALG-P(ℓ)* and performed experimental tests for an IBM mobile hard drive with four power states. It shows that the combined scheme achieves low energy consumption close to the optimum and usually outperforms many single-value prediction algorithms.

Augustine et al. [4] investigate generalized multi-state systems in which the state transition energies may take arbitrary values. Let $\beta_{ij} \ge 0$ be the energy required to transition from s_i to s_j, $1 \le i, j \le \ell$. Augustine et al. demonstrate that *Lower-Envelope* can be generalized and achieves a competitiveness of $3 + 2\sqrt{2} \approx 5.8$. This ratio holds for any state system. Better bounds are possible for specific system. Augustine et al. devise a strategy that, for a given system S, achieves a competitive ratio of $c^* + \epsilon$, for any $\epsilon > 0$, where c^* is the best competitiveness possible for S. Finally, the authors consider stochastic settings and develop optimal state transition times.

3 Dynamic Speed Scaling

Many modern microprocessor can run at variable speed. Examples are the Intel SpeedStep and the AMD processor PowerNow. High speeds result in higher performance but also high energy consumption. Lower speeds save energy but performance degrades. If the processor runs at speed s, then the required power is s^{α}, where $\alpha > 1$ is a constant. In practical application α is usually in the range between 2 and 3. The well-known cube-root rule states that the power is proportional to s^3. Obviously, energy consumption is power integrated over time. The goal is to dynamically set the speed of a processor so as to minimize energy consumption, while still providing a desired quality of service.

Dynamic speed scaling leads to many challenging scheduling problems. At any time a scheduler has to decide not only which job to execute but also which speed to use. Consequently, there has been considerable research interest in the design and analysis of efficient scheduling algorithms. This section reviews the most important results developed over the past years. We first address scheduling problems with hard job deadlines. Then we consider the minimization of response times and other objectives.

In general, two scenarios are of interest. In the offline setting all jobs to be processed are known in advance. In the online setting jobs arrive over time and an algorithm, at any time, has to make scheduling decisions without knowledge of any future jobs. Recall that an online algorithm ALG is c-competitive if, for every input, the objective function value (typically the energy consumption) of ALG is within c times the value of an optimal solution.

3.1 Scheduling with Deadlines

In a seminal paper, initiating the algorithmic study of speed scaling, Yao, Demers and Shenker [30] investigated a scheduling problem with strict job deadlines. At this point this framework is by far the most extensively studied algorithmic speed scaling problem.

Consider n jobs $J_1, \ldots, J_n$ that have to be processed on a variable-speed processor. Each job J_i is specified by a release time r_i, a deadline d_i and a processing volume w_i. The release time and the deadline mark the time interval in which the job must be executed. The processing volume is the amount of work that must be done to complete the job. The processing time of a job depends on the speed. If J_i is executed at constant speed s, it takes w_i/s time units to finish the job. Preemption of jobs is allowed, i.e. the processing of a job may be suspended and resumed later. The goal is to construct a feasible schedule minimizing the total energy consumption.

The framework by Yao et al. assumes that there is no upper bound on the maximum processor speed. Hence there always exists a feasible schedule satisfying all job deadlines. Furthermore, it is assumed that a continuous spectrum of speeds is available. We will discuss later how to relax these assumptions.

Fundamental Algorithms. Yao, Demers and Shenker [30] first study the offline setting and develop an algorithm for computing optimal solutions, minimizing total energy consumption. The strategy is known as *YDS* referring to the initials of the authors. The algorithm proceeds in series of iterations. In each iteration, a time interval of maximum density is identified and a corresponding partial schedule is constructed. The *density* Δ_I of a time interval $I = [t, t']$ is the total work to be completed in I divided by the length of I. More precisely, let S_I be the set of jobs J_i that must be processed in I, i.e. that satisfy $[r_i, d_i] \subseteq I$. Then

$$\Delta_I = \frac{1}{|I|} \sum_{J_i \in S_i} w_i.$$

Intuitively, Δ_I is the minimum average speed necessary to complete all jobs that must be scheduled in I.

Algorithm *YDS* repeatedly determines the interval I of maximum density. In such an interval I the algorithm schedules the jobs of S_I at speed Δ_I using the *Earliest Deadline First (EDF)* policy, i.e. among the available unfinished jobs the one with the earliest deadline is executed. Then *YDS* removes the set S_I as well as the time interval I from the problem instance. More specifically, for any unscheduled job J_i with $d_i \in I$, the new deadline time is set to $d_i := t$. For any unscheduled J_i with $r_i \in I$, the new release time is $r_i := t'$. Time interval I is discarded. We give a summary of the algorithm in pseudo-code.

Algorithm YDS: Initially $\mathcal{J} := \{J_1, \ldots, J_n\}$. While $\mathcal{J} \neq \emptyset$, execute the following two steps. (1) Determine the interval I of maximum density. In I process the jobs of S_I at speed Δ_I according to *EDF*. (2) Set $\mathcal{J} := \mathcal{J} \setminus S_I$. Remove I from the time horizon and update the release times and deadlines of unscheduled jobs accordingly.

Theorem 6. [30] *For any job instance, YDS computes an optimal schedule minimizing the total energy consumption.*

Obviously, when identifying intervals of maximum density, *YDS* only has to consider intervals whose boundaries are equal to the release times and deadlines of the jobs. A straightforward implementation of the algorithm has a running time of $O(n^3)$. Li et al. [25] showed that the time can be reduced to $O(n^2 \log n)$. Further improvements are possible if the job execution intervals form a tree structure [24].

Yao et al. [30] also devised two elegant online algorithms, called *Average Rate* and *Optimal Available*. Whenever a new job J_i arrives at time r_i, its deadline d_i and processing volume w_i are known. For any incoming job J_i, *Average Rate* considers the *density* $\delta_i = w_i/(d_i - r_i)$, which is the minimum average speed necessary to complete the job in time if no other jobs were present. At any time t the speed $s(t)$ is set to the accumulated density of jobs active at time t. A job J_i is *active at time* t if $t \in [r_i, d_i]$. Available jobs are scheduled according to the *EDF* policy.

Algorithm Average Rate: At any time t use a speed of $s(t) = \sum_{J_i : t \in [r_i, d_i]} \delta_i$. Available unfinished jobs are scheduled using *EDF*.

Yao et al. [30] analyzed *Average Rate* and proved an upper bound on the competitiveness.

Theorem 7. [30] *The competitive ratio of Average Rate is at most* $2^{\alpha-1}\alpha^\alpha$*, for any* $\alpha \geq 2$.

Bansal et al. [5] showed that the analysis is essentially tight by providing a nearly matching lower bound.

Theorem 8. [5] *The competitive ratio of Average Rate is at least* $((2-\delta)\alpha)^\alpha/2$*, where* δ *is a function of* α *that approaches zero as* α *tends to infinity.*

The second strategy *Optimal Available* is computationally more expensive than *Average Rate*. It always computes an optimal schedule for the currently available work load. This can be done using *YDS*.

Algorithm Optimal Available: Whenever a new job arrives, compute an optimal schedule for the currently available unfinished jobs.

Bansal, Kimbrel and Pruhs [8] gave a comprehensive analysis of the above algorithm and proved the following result.

Theorem 9. [8] *The competitive ratio of Optimal Available is exactly* α^α.

The above theorem implies that in terms of competitiveness, *Optimal Available* is better than *Average Rate*. Bansal et al. [8] also presented a new online algorithm, called *BKP* according to the initials of the authors, that approximates the optimal speeds of *YDS* by considering interval densities. For times t, t_1 and t_2 with $t_1 < t \leq t_2$, let $w(t, t_1, t_2)$ be the total processing volume of jobs that are active at time t, have a release time of at least t_1 and a deadline of at most t_2.

Algorithm BKP: At any time t use a speed of

$$s(t) = \max_{t'>t} \frac{w(t, et-(e-1)t', t')}{t'-t}.$$

Available unfinished jobs are processed using *EDF*.

Theorem 10. [8] *Algorithm BKP achieves a competitive ratio of* $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$.

For large values of α, the competitiveness of *BKP* is better than that of *Optimal Available*.

All the above online algorithms attain constant competitive ratios that depend on α but no other other problem parameter. The dependence on α is exponential. For small values of α, which occur in practice, the competitive ratios are reasonably small. A result by Bansal et al. [8] implies that the exponential dependence on α is inherent to the problem.

Theorem 11. [8] *Any randomized online algorithm has a competitiveness of at least* $\Omega((4/3)^\alpha)$.

Refinements. *Bounded speed:* The problem setting considered so far assumes a continuous, unbounded spectrum of speeds. However, in practice only a finite set of discrete speed levels $s_1 < s_2 < \ldots < s_d$ is available. The offline algorithm *YDS* can be adapted easily to handle feasible job instances, i.e. inputs for which feasible schedules exist using the restricted set of speeds. Note that feasibility can be checked easily by always using the maximum speed s_d and scheduling available jobs according to the *EDF* policy. Given a feasible job instance, the modification of *YDS* is as follows. We first construct the schedule according to *YDS*. For each identified interval I of maximum density we approximate the desired speed Δ_I by the two adjacent speed levels $s_k < \Delta_I < s_{k+1}$. Speed s_{k+1} is used first for some δ time units and s_k is used for the last $|I|-\delta$ time units in I, where δ is chosen such that the total work completed in I is equal to the original amount of $|I|\Delta_I$. An algorithm with an improved running time of $O(dn \log n)$ was presented by Li and Yao [26].

If the given job instance is not feasible, the situation is more delicate. In this case it is impossible to complete all the jobs. The goal is to design algorithms that achieve good *throughput*, which is the total processing volume of jobs finished by their deadline, and at the same time optimize energy consumption. Papers [6, 13] present algorithms that even work online. At any time the strategies maintain a pool of jobs they intend to complete. Newly arriving jobs may be admitted to this pool. If the pool contains too large a processing volume, jobs are expelled such that the throughput is not diminished significantly. The algorithm by Bansal et al. [6] is 4-competitive in terms of throughput and constant competitive with respect to energy consumption.

Temperature minimization: High processor speeds lead to high temperatures which impair a processor's reliability and lifetime. Bansal et al. [8] consider the minimization of the maximum temperature that arises during processing. They assume that cooling follows Newton's law, which states that the rate of cooling of a body is proportional to the difference in temperature between the body and the environment. Bansal et al. [8] show that algorithms *YDS* and *BKP* have favorable properties. For any jobs sequence, the maximum temperature is within a constant factor of the minimum possible maximum temperature, for any cooling parameter a device may have.

Sleep states: Irani et al. [19] investigate an extended problem setting where a variable-speed processor may be transitioned into a sleep state. In the sleep state, the energy consumption is 0 while in the active state even at speed 0 some non-negative amount of energy is consumed. Hence [19] combines speed scaling with power-down mechanisms. In the standard setting without sleep state, algorithms tend to use low speed levels subject to release time and deadline constraints. In contrast, in the setting with sleep state it can be beneficial to speed up a job so as to generate idle times in which the processor can be transitioned to the sleep mode. Irani et al. [19] develop online and offline algorithms for this extended setting. Baptiste et al. [10] and Demaine et al. [15] also study scheduling problems where a processor may be set asleep, albeit in a setting without speed scaling.

3.2 Minimizing Response Time

A classical objective in scheduling is the minimization of response times. A user releasing a task to a system would like to receive feedback, say the result of a computation, as quickly as possible. User satisfaction often depends on how fast a device reacts. Unfortunately, response time minimization and energy minimization are contradicting objectives. To achieve fast response times a system must usually use high processor speeds, which lead to high energy consumption. On the other hand, to save energy low speeds should be used, which result in high response times. Hence one has to find ways to integrate both objectives.

Consider n jobs $J_1, \ldots, J_n$ that have to be scheduled on a variable-speed processor. Each job J_i is specified by a release time r_i and a processing volume w_i. When a job arrives, its processing volume is known. Preemption of jobs is allowed. In the scheduling literature, response time is referred to as *flow time.* The flow time f_i of a job J_i is the length of the time interval between release time and completion time of the job. We seek schedules minimizing the total flow time $\sum_{i=1}^{n} f_i$.

Limited energy: Pruhs et al. [27] assume that a fixed energy volume E is given and the goal is to minimize the total flow time of the jobs. The authors consider unit-size jobs, i.e. all jobs have the same processing volume, and study the offline scenario where all the jobs are known in advance. Pruhs et al. [27] show that optimal schedules can be computed in polynomial time. However, in this framework with a limited energy volume it is hard to construct good online algorithms. If future jobs are unknown, it is unclear how much energy to invest for the currently available tasks.

Energy plus flow times: Albers and Fujiwara [2] proposed another approach to integrate energy and flow time minimization. They consider a combined objective function that simply adds the two costs. Let E denote the energy consumption of a schedule. We wish to minimize $g = E + \sum_{i=1}^{n} f_i$. Albers and Fujiwara concentrate on unit-size jobs and show that optimal offline schedules can be constructed in polynomial time using a dynamic programming approach. In fact the algorithm can also be used to minimize the total flow time of jobs given a fixed energy volume.

Most of [2] is concerned with the online setting where jobs arrive over time. Albers and Fujiwara present a simple online strategy that processes jobs in batches and achieves a constant competitive ratio. Batched processing allows one to make scheduling decisions, which are computationally expensive, only every once in a while. This is certainly an advantage in low-power computing environments. Nonetheless, Albers and Fujiwara conjectured that the following algorithm achieves a better performance with respect to the minimization of g: At any time, if there are ℓ active jobs, use speed $\sqrt[\alpha]{\ell}$. A job is active if it has been released but is still unfinished. This algorithm and variants thereof have been the subject of extensive analyses [6, 7, 9, 23], not only for unit-size but also for arbitrary size jobs. Moreover, unweighted and weighted flow times have been considered.

The currently best result is due to Bansal et al. [7]. They modify the above algorithm slightly by using a speed of $\sqrt[\alpha]{\ell + 1}$ whenever ℓ jobs are active. Inspired by a paper of Lam et al. [23] they apply the *Shortest Remaining Processing Time*

(SRPT) policy to the available jobs. More precisely, among the active jobs, the one with the least remaining work is scheduled.

Algorithm Job Count: At any time if there are $\ell \geq 1$ active jobs, use speed $\sqrt[\alpha]{\ell + 1}$. If no job is available, use speed 0. Always schedule the job with the least remaining unfinished work.

Theorem 12. [7] *Algorithm* Job Count *is 3-competitive for arbitrary size jobs.*

Further work considering the weighted flow time in objective function g can be found in [7, 9]. Moreover, [6, 23] propose algorithms for the setting that there is an upper bound on the maximum processor speed.

All the above results assume that when a job arrives, its processing volume is known. Papers [14, 23] investigate the harder case that this information is not available.

3.3 Extensions and Other Objectives

Parallel processors: The results presented so far address single-processor architectures. However, energy consumption is also a major concern in multi-processor environments. Currently, relatively few results are known. Albers et al. [3] investigate deadline-based scheduling on m identical parallel processors. The goal is to minimize the total energy on all the machines. The authors first settle the complexity of the offline problem by showing that computing optimal schedules is NP-hard, even for unit-size jobs. Hence, unless $P \neq NP$, optimal solutions can not be computed efficiently. Albers et al. [3] then develop polynomial time offline algorithms that achieve constant factor approximations, i.e. for any input the consumed energy is within a constant factor of the true optimum. They also devise online algorithms attaining constant competitive ratios. Lam et al. [21] study deadline-based scheduling on two speed-bounded processors. They present a strategy that is constant competitive in terms of throughput maximization and energy minimization.

Bunde [12] investigates flow time minimization in multi-processor environments, given a fixed energy volume. He presents hardness results as well as approximation guarantees for unit-size jobs. Lam et al. [22] consider the objective function of minimizing energy plus flow times. They design online algorithms achieving constant competitive ratios.

Makespan minimization: Another basic objective function in scheduling is makespan minimization, i.e. the minimization of the point in time when the entire schedule ends. Bunde [12] assumes that jobs arrive over time and develops algorithms for single and multi-processor environments. Pruhs et al. [28] consider tasks having precedence constraints defined between them. They devise algorithms for parallel processors given a fixed energy volume.

4 Conlusions

This article has surveyed algorithmic approaches to save energy. Another survey on algorithmic problems in power management was written by Irani and

Pruhs [18]. The past months have witnessed considerable research activity and it is conceivable that energy conservation from an algorithmic point of view will continue to be an active area of investigation. Many open problems remain. With respect to power-down mechanisms, for instance, it would be interesting to design strategies that take into account the latency that arises when a system is transitioned from a sleep state to the active state. As for speed scaling techniques, we need a better understanding of strategies for multi-processor environments as multi-core architectures become more and more common not only in servers but also in desktops and laptops.

References

1. `http://www.microsoft.com/whdc/system/pnppwr/powermgmt/default.mspx`
2. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. ACM Transactions on Algorithms 3 (2007)
3. Albers, S., Müller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proc. 19th ACM Symposium on Parallelism in Algorithms and Architectures, pp. 289–298 (2007)
4. Augustine, J., Irani, S., Swamy, C.: Optimal power-down strategies. SIAM Journal on Computing 37, 1499–1516 (2008)
5. Bansal, N., Bunde, D.P., Chan, H.-L., Pruhs, K.: Average rate speed scaling. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 240–251. Springer, Heidelberg (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.: Speed scaling with an arbitrary power function. In: Proc. 20th ACM-SIAM Symposium on Discrete Algorithm, pp. 693–701 (2009)
8. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. Journal of the ACM 54 (2007)
9. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 805–813 (2007)
10. Baptiste, P., Chrobak, M., Dürr, C.: Polynomial time algorithms for minimum energy scheduling. In: Arge, L., Hoffmann, M., Welzl, E. (eds.) ESA 2007. LNCS, vol. 4698, pp. 136–150. Springer, Heidelberg (2007)
11. Barroso, L.A.: The price of performance. ACM Queue 3 (2005)
12. Bunde, D.P.: Power-aware scheduling for makespan and flow. In: Proc. 18th Annual ACM Symposiun on Parallel Algorithms and Architectures, pp. 190–196 (2006)
13. Chan, H.-L., Chan, W.-T., Lam, T.-W., Lee, K.-L., Mak, K.-S., Wong, P.W.H.: Energy efficient online deadline scheduling. In: Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 795–804 (2007)
14. Chan, H.-L., Edmonds, J., Lam, T.-W., Lee, L.-K., Marchetti-Spaccamela, A., Pruhs, K.: Nonclairvoyant speed scaling for flow and energy. In: Proc. 26th International Symposium on Theoretical Aspects of Computer Science, pp. 255–264 (2009)
15. Demaine, E.D., Ghodsi, M., Hajiaghayi, M.T., Sayedi-Roshkhar, A.S., Zadimoghaddam, M.: Scheduling to minimize gaps and power consumption. In: Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 46–54 (2007)

16. Irani, S., Karlin, A.R.: Online computation. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-Hard Problems, pp. 521–564. PWS Publishing Company (1997)
17. Irani, S., Shukla, S.K., Gupta, R.K.: Online strategies for dynamic power management in systems with multiple power-saving states. ACM Transaction in Embedded Computing Systems 2, 325–346 (2003)
18. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News 36, 63–76 (2005)
19. Irani, S., Shukla, S.K., Gupta, R.: Algorithms for power savings. ACM Transactions on Algorithms 3 (2007)
20. Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive randomized algorithms for nonuniform problems. Algorithmica 11, 542–571 (1994)
21. Lam, T.-W., Lee, L.-K., To, I.K.K., Wong, P.W.H.: Energy efficient deadline scheduling in two processor systems. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 476–487. Springer, Heidelberg (2007)
22. Lam, T.-W., Lee, L.-K., To, I.K.-K., Wong, P.W.H.: Competitive non-migratory scheduling for flow time and energy. In: Proc. 20th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 256–264 (2008)
23. Lam, T.-W., Lee, L.-K., To, I.K.K., Wong, P.W.H.: Speed scaling functions for flow time scheduling based on active job count. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 647–659. Springer, Heidelberg (2008)
24. Li, M., Liu, B.J., Yao, F.F.: Min-energy voltage allocation for tree-structured tasks. Journal on Combintorial Optimization 11, 305–319 (2006)
25. Li, M., Yao, A.C., Yao, F.F.: Discrete and continuous min-energy schedules for variable voltage processors. Proc. National Academy of Sciences USA 103, 3983–3987 (2006)
26. Li, M., Yao, F.F.: An efficient algorithm for computing optimal discrete voltage schedules. SIAM Journal on Computing 35, 658–671 (2005)
27. Pruhs, K., Uthaisombut, P., Woeginger, G.J.: Getting the best response for your erg. ACM Transactions on Algorithms 4 (2008)
28. Pruhs, K., van Stee, R., Uthaisombut, P.: Speed scaling of tasks with precedence constraints. Theory of Computing Systems 43, 67–80 (2008)
29. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communcations of the ACM 28, 202–208 (1985)
30. Yao, F.F., Demers, A.J., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. 36th IEEE Symposium on Foundations of Computer Science, pp. 374–382 (1995)