Online Dynamic Power Management with Hard Real-Time Guarantees*

Jian-Jia Chen[1], Mong-Jen Kao[2], D. T. Lee[2,3], Ignaz Rutter[1], and Dorothea Wagner[1]

1 **Faculty for Informatics, Karlsruhe Institute of Technology (KIT), Germany**
j.chen@kit.edu, rutter@kit.edu, dorothea.wagner@kit.edu
2 **Institute for Information Science, Academia Sinica, Taipei, Taiwan**
mong@iis.sinica.edu.tw
3 **Department of Computer Science and Information Engineering, National Chung-Hsing University, Tai-Chung, Taiwan**
dtlee@ieee.org

Abstract

We consider the problem of online dynamic power management that provides hard real-time guarantees for multi-processor systems. In this problem, a set of jobs, each associated with an arrival time, a deadline, and an execution time, arrives to the system in an online fashion. The objective is to compute a non-migrative preemptive schedule of the jobs and a sequence of power on/off operations of the processors so as to minimize the total energy consumption while ensuring that all the deadlines of the jobs are met. We assume that we can use as many processors as necessary. In this paper we examine the complexity of this problem and provide online strategies that lead to practical energy-efficient solutions for real-time multi-processor systems.

First, we consider the case for which we know in advance that the set of jobs can be scheduled feasibly on a single processor. We show that, even in this case, the competitive factor of any online algorithm is at least 2.06. On the other hand, we give a 4-competitive online algorithm that uses at most two processors. For jobs with unit execution times, the competitive factor of this algorithm improves to 3.59.

Second, we relax our assumption by considering as input multiple streams of jobs, each of which can be scheduled feasibly on a single processor. We present a trade-off between the energy-efficiency of the schedule and the number of processors to be used. More specifically, for k given job streams and h processors with $h > k$, we give a scheduling strategy such that the energy usage is at most $4 \cdot \lceil \frac{k}{h-k} \rceil$ times that used by any schedule which schedules each of the k streams on a separate processor. Finally, we drop the assumptions on the input set of jobs. We show that the competitive factor of any online algorithm is at least 2.28, even for the case of unit job execution times for which we further derive an $O(1)$-competitive algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Energy-Efficient Scheduling, Online Dynamic Power Management

Digital Object Identifier 10.4230/LIPIcs.STACS.2014.226

1 Introduction

Reducing power consumption and improving energy efficiency have become important design requirements in computing systems. For mobile devices, effective power management

* This work was supported in part by National Science Council (NSC), Taiwan, under Grants NSC101-2221-E-005-026-MY2 and NSC101-2221-E-005-019-MY2.

31st Symposium on Theoretical Aspects of Computer Science (STACS'14).
Editors: Ernst W. Mayr and Natacha Portier; pp. 226–238

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

can considerably extend the standby period and prolong battery lifetime. For large-scale computing clusters, appropriately powering down the idling processing units can considerably reduce the electricity bill.

In order to increase the energy efficiency, two different mechanisms have been introduced to reduce the energy consumption. *(1) Power-down Mechanism:* When a processor is idling, it can be put into a low-power state, e.g., sleep or power-off. While the processor consumes less power in these states, a fixed amount of energy is required to switch the system back to work. In the literature, the problem of deciding the sequence of state transitions is referred to as *dynamic power management (DPM). (2) Dynamic Speed Scaling:* The concept of dynamic speed scaling refers to the flexibility provided by a processor to adjust its processing speed dynamically. The rate of energy consumption is typically described by a convex function of the processing speed. This feature is also referred to as *dynamic voltage frequency scaling (DVFS)*, following its practical implementation scheme.

The majority of previous work regarding energy-efficient scheduling focuses mainly on uni-processor systems. For systems that support power-down mechanism, Baptiste [5] considered hard real-time jobs, i.e., deadline misses of jobs are not allowed, with unit execution times and proposed the first polynomial-time algorithm that computes an optimal strategy for turning on and powering off a processor. In a follow-up paper, Baptiste et al. [6] further extended the result to jobs with arbitrary execution times and reduced the time complexity. When considering workload-conserving scheduling, i.e., the system is not allowed to enter low-power states when the ready queue is not empty, Augustine et al. [4] considered systems with multiple low-power states and provided online algorithms.

Dynamic speed scaling was introduced to allow computing systems to reach a balance between high performance and low power consumption dynamically. Hence, scheduling algorithms that assume dynamic speed scaling, e.g., Yao et al. [30], usually execute jobs as slowly as possible while ensuring that timing constraints are met. When the energy required to keep the processor active is not negligible, however, executing jobs too slowly may result in more energy consumption. For most realistic power-consumption functions, there exist a *critical speed*, which is the most energy-efficient for job execution [12, 22].

Irani et al. [22] initiated the study of combining both mechanisms. For offline energy-minimization, they presented a 2-approximation. For the online version, they introduced a *greedy procrastinating principle*, which enables online algorithms that have certain properties and that are designed for speed scaling without power-down mechanism to additionally support the power-down mechanism. The idea behind this principle is to postpone job execution as much as possible in order to bundle workload for batch execution. The usage of job procrastination with dynamic speed scaling for periodic tasks has later been explored extensively in a series of studies [11, 12, 26].

The combination of the power-down mechanism with dynamic speed scaling suggests the philosophy of *racing-to-idle*: Execute jobs at higher speeds and gain longer quality sleeping intervals. Albers and Antoniadis [1] showed that the problem of minimizing the energy consumption for speed scaling with a sleep state is NP-hard and provided a $\frac{4}{3}$-approximation.

All of the aforementioned work mainly focuses on uni-processor systems. By contrast, for multi-processor systems, relatively fewer results are known. Demaine et al. [17] considered unit jobs and presented a polynomial-time algorithm based on dynamic programming for power-down mechanism. Approximations for several variations were also presented. In a follow-up paper, Demaine and Zadimoghaddam [18] presented logarithmic approximations for general formulations of scheduling problems with submodular objective functions, including energy consumption. Albers et al. [2] considered dynamic speed scaling with job migration

and presented polynomial-time algorithms based on maximum flow problems.

As scheduling to meet deadline constraints is a long-standing difficult problem [14–16, 20], additional augmentation on the hardware level, e.g., speed of the processors or number of the machines, has been considered to provide practical solutions. See, for example, [3, 8, 10, 28]. In practice, machine augmentation follows the trends in multi-core systems, while speed augmentation has been shown its limits as overclocking is difficult to achieve due to the dramatic increase of power consumption and thermal dissipation.

Our Focus and Contribution. In this paper, we examine the problem of online dynamic power management that provides hard real-time guarantees, i.e., each job must finish its execution before its deadline, for multi-processor systems. We assume a system equipped with multiple processors that are identical and that operate independently from each other, and we can use as many processors as necessary. Job executions can be preempted but can not be migrated, i.e., the execution of a job must be done on the same processor. The objective is to compute a schedule of the jobs and a sequence of switch on/off operations of the processors so as to minimize the total energy consumption. For this problem model we give an elaborate study that leads to practical energy-efficient solutions for real-time multi-processor systems.

First, we consider the case for which we know in advance that the set of jobs can be scheduled feasibly on one processor. We show that the competitive factor of any online algorithm is at least 2.06, even for this restricted case. Then we propose the idea of *energy-efficient anchors*, which are defined for each of the jobs, to indicate a proper moment for which the online scheduler should no longer postpone the execution of the jobs. We show that this idea leads to a 4-competitive online algorithm which uses at most two processors. For jobs with unit execution times, we show that the competitive factor improves to 3.59.

Second, we relax the conditions of our assumption by considering as input multiple streams of jobs, each of which can be scheduled feasibly on one processor. We present a simple strategy, as a byproduct of our first algorithm, to allow a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule. More specifically, for k given job streams and h processors with $h > k$, we give a scheduling strategy such that the energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used by any schedule which schedules each of the k streams on a separate processor.

The above algorithms lead to practical energy-efficient solutions in real-time systems for which partitioned scheduling with recurrent real-time task model is adopted [7, 9]. The recurrent task models, such as the *sporadic real-time task* model [27] or the *arrival curve* model in Real-Time Calculus (RTC) [29], describe an infinite sequence of job arrivals, generated by the tasks. Under such a model, the worst-case characteristics of job arrivals are specified. For example, a sporadic real-time task defines the minimum inter-arrival time of any two consecutive jobs. With the partitioned scheduling scheme, it is required that all the jobs generated by a recurrent task be executed on a single processor. In many cases, however, the real-time system needs to provide hard deadline guarantees and verifying the system could be very costly, and the goal is to make the schedule more efficient by using more processors without going through the costly verification steps. Therefore, even though the partitioned scheduling scheme is more restricted in the sense that the jobs are partitioned in advance, it has been widely adopted in practical real-time systems [7, 9] as it incurs no additional overhead for ensuring the feasibility of the resulting online schedule. Our algorithms provide an online energy-efficient solution with a reasonable trade-off for this situation.

Finally we drop the assumptions on the schedulability as well as the number of job

streams and consider general set of jobs with unit execution times.

We show that the competitive factor of any online algorithm is at least 2.28. For the positive side, we present a $O(1)$-competitive algorithm, which combines ideas from different results and is interesting in its own right.

2 Notations and Problem Definition

In this section, we formally define the scheduling problem we consider and introduce notations that will be used throughout this paper. Each job, say, j, is associated with three non-negative integral parameters, namely, the arrival time a_j, the execution time c_j, and the deadline d_j. The arrival time of a job is the time it arrives to the system. The execution time is the amount of CPU time it requires to finish its execution, and the deadline is the latest moment at which the job must be completed. We assume that c_j and d_j are known at the moment when j arrives to the system.

For notational brevity, we use a triple $j = (a_j, d_j, c_j)$ to denote the corresponding parameters for any job j. We say a job j is a unit job if $c_j = 1$ and we write $j = (a_j, d_j)$. In addition, a job j is said to be *urgent* if $c_j = d_j - a_j$.

We make the following assumptions on the processors. When a processor is *off*, it cannot execute any job and consumes no power. Switching on, or, alternatively, turning on, a processor requires E_w units of energy. When a processor is on, it can execute jobs at a fixed speed. We say that a processor is in the *busy* state if it is executing a job. If a processor is on but not executing any job, then it is said to be in the *standby* state. The energy consumed by a processor per unit of time, i.e., the power consumption, is ψ_b when it is busy and ψ_σ when it is in standby, respectively. We assume that $\psi_\sigma \leq \psi_b$. Initially all processors are off.

The *break-even time*, denoted by $\mathcal{B}$, is defined to be E_w/ψ_σ. Intuitively, this corresponds to the amount of time a processor has to stay in standby in order to have the same energy consumption for switching on a processor. The break-even time is an important concept that has been widely used for ski-rental-related problems [25] and dynamic power management algorithms in the literature, e.g., [21–23].

Below we define the concept of job scheduling. Let $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ be the set of processors and $\mathcal{J}$ be a set of jobs.

▶ **Definition 1** (A Schedule for a Set of Jobs). A schedule $\mathbf{S}$ for $\mathcal{J}$ on $\mathcal{M}$ is a set of pairs $(\mathcal{I}_1, \text{job}_1), (\mathcal{I}_2, \text{job}_2), \ldots, (\mathcal{I}_m, \text{job}_m)$, where for each $1 \leq i \leq m$,

- $\mathcal{I}_i$ denotes the set of time intervals during which processor M_i is on, and
- $\text{job}_i \colon \mathbb{R}^+ \longrightarrow \mathcal{J} \cup \{\emptyset\}$ is a function of time t that indicates the job to be executed on processor M_i at time t. If M_i is not executing any job or is off at time t, then $\text{job}_i(t) = \emptyset$.

The schedule $\mathbf{S}$ is said to be *feasible* for $\mathcal{J}$ on $\mathcal{M}$ if for each job $j \in \mathcal{J}$, there exists a processor $M_i \in \mathcal{M}$ such that

$$\sum_{\mathrm{I} \in \mathcal{I}_i} \int_{\mathrm{I} \cap [a_j, d_j]} \delta(\text{job}_i(t), j) \cdot \mathrm{d}t = c_j,$$

where $\delta(x, y)$ is defined to be 1 if $x = y$ and 0 otherwise. In other words, the schedule $\mathbf{S}$ is feasible if for each job j, there exists a processor M_i such that the amount of time M_i is executing j during the time interval $[a_j, d_j]$ is c_j. We remark that, it is implicitly implied in the definition that a feasible schedule is also a preemptive schedule and the jobs can only be executed without migration.

The number of processors the schedule $\mathbf{S}$ uses is defined to be $|\{i\colon 1 \le i \le m, \mathcal{I}_i \neq \emptyset\}|$, i.e., the number of processors that have been switched on at least once in $\mathbf{S}$. The energy consumption of the schedule $\mathbf{S}$, denoted $\mathrm{E}(\mathbf{S})$, is defined as

$$\mathrm{E}(\mathbf{S}) = \sum_{1 \le i \le m} \left(E_w \cdot |\mathcal{I}_i| + \sum_{I \in \mathcal{I}_i} |I| \cdot \psi_\sigma \right) + \sum_{j \in \mathcal{J}} c_j \cdot (\psi_b - \psi_\sigma),$$

where $|\mathcal{I}_i|$ denotes cardinality of $\mathcal{I}_i$, i.e., the number of time intervals it contains, and $|I|$ denotes the length of the time interval I.

▶ **Definition 2** (DPM Job Scheduling)**.** Given a set $\mathcal{J}$ of jobs and a set $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ of processors, the *DPM Job Scheduling Problem* is to find a feasible schedule $\mathbf{S}$ for $\mathcal{J}$ on $\mathcal{M}$ such that $\mathrm{E}(\mathbf{S})$ is minimized.

In this paper, we consider the online version of the DPM job scheduling problem in which the jobs arrive to the system dynamically in an online manner, i.e., at any time t, the algorithm only sees the jobs whose arrival times are less than or equal to t, and the scheduling decisions have to be made without prior knowledge on future job arrivals. To be more precise, let $\mathcal{J}$ be the input job set and $\mathcal{J}(t) = \{j\colon j \in \mathcal{J}, a_j \le t\}$ be the subset of $\mathcal{J}$ that contains the jobs whose arrival times are no greater than t. At any time t, the algorithm sees the job set $\mathcal{J}(t)$ and is able to decide the assignment of the jobs to the processors as well as the state transitions of the processors, i.e., turning on or switching off, at that time. The objective is to compute a feasible schedule for $\mathcal{J}$ such that the energy consumed is minimized.

An online algorithm Π is said to be η-competitive for the online DPM job scheduling problem if for any job set $\mathcal{J}$, we have $\mathrm{E}(\Pi(\mathcal{J})) \le \eta \cdot \mathrm{E}(Opt(\mathcal{J})) + x$, where $\Pi(\mathcal{J})$ is the schedule computed by algorithm Π, $Opt(\mathcal{J})$ is an optimal schedule for $\mathcal{J}$, i.e., the one that results in the minimum energy consumption, and x is a constant.

3 Preliminary Results

In this section, we present preliminary results that are related to our assumptions for the problem models we addressed. We begin with the characterization of the job sets that are packable on one processor. For any job set $\mathcal{J}$ and any $0 \le \ell < r$, let $\Upsilon(\mathcal{J}, \ell, r) = \{j : j \in \mathcal{J},\ \ell \le a_j,\ d_j \le r\}$ be the set of jobs that arrive and have to be done within the time interval $[\ell, r]$. Let $\Upsilon^{\#}(\mathcal{J}, \ell, r) = \sum_{j \in \Upsilon(\mathcal{J}, \ell, r)} c_j$ denote the total amount of workload in $\Upsilon(\mathcal{J}, \ell, r)$.

Chetto et al. [13] studied the schedulability of any given set of jobs using the *earliest-deadline-first (EDF)* principle, which always selects the job with the earliest deadline for execution at any moment, and proved the following lemma.

▶ **Lemma 3** (Chetto et al. [13])**.** *For any set $\mathcal{J}$ of jobs, $\mathcal{J}$ can be scheduled on one processor using the EDF principle if and only if the following condition holds:*

$$\textit{For any time interval } [\ell, r], \quad \Upsilon^{\#}(\mathcal{J}, \ell, r) \le r - \ell. \tag{1}$$

It is well-known that, for any job set $\mathcal{J}$, if there exists a feasible schedule that uses only one processor for $\mathcal{J}$, then the EDF principle is also guaranteed to produce a feasible schedule [19]. Hence, Condition (1) gives a necessary and sufficient condition for any set of jobs to be able to be packable on a processor.

However, even when the set of jobs is known in advance to be packable on a processor, we still need multiple processors in order to achieve energy-efficiency in an online setting. This is illustrated by the following lemma.

▶ **Lemma 4.** *Let* Π *be an online algorithm that produces feasible schedules using only one processor for any job set that can be packed feasibly on one processor. For any* $\alpha > 0$*, there exists a job set* $\mathcal{J}_\alpha$ *that can be packed feasibly on one processor such that the competitive factor of* Π *on* $\mathcal{J}_\alpha$ *is at least* α*.*

Hence it is essential to use additional processors in order to give an energy-efficient scheduling scheme for the online DPM job scheduling problem. This dilemma is further extended in §4.1 to obtain a lower bound on the competitive factor of any online algorithm.

Below we introduce notions related to the number of processors required by a set of *unit jobs*. For any job set $\mathcal{J}$ and any $0 \le \ell < r$, let

$$\rho(\mathcal{J}, \ell, r) = \frac{\Upsilon^{\#}(\mathcal{J}, \ell, r)}{r - \ell}$$

denote the density of workload of $\mathcal{J}$ with respect to the time interval $[\ell, r]$. Let $\hat{\varrho}(\mathcal{J}) = \max_{0 \le \ell < r} \rho(\mathcal{J}, \ell, r)$ denote the maximum density of $\mathcal{J}$. Let $\mathrm{P}^{\#}(\mathcal{J})$ denote the minimum number of processors required by any feasible schedule for $\mathcal{J}$. The following lemma gives an alternative characterization of $\mathrm{P}^{\#}(\mathcal{J})$ when $\mathcal{J}$ is composed merely by unit jobs.

▶ **Lemma 5** ([24]). *For any set* $\mathcal{J}$ *of unit jobs, we have* $\mathrm{P}^{\#}(\mathcal{J}) = \lceil \hat{\varrho}(\mathcal{J}) \rceil$*.*

However, for jobs with arbitrary execution times, packing the jobs using a minimum number of processors is a long-standing difficult problem even for the offline case [14–16, 20], and for the online version only very special cases were studied [16, 24].

4 Online Scheduling for Job Sets Packable on One Processor

In this section, we consider the case for which we know in advance that the input set of jobs can be scheduled feasibly on one processor, i.e., Condition (1) from Lemma 3 holds for the input set of jobs. First we prove a lower bound of 2.06 on the competitive factor of any online algorithm by designing an online adversary $\mathcal{A}$ that observes the behavior of the scheduling algorithm and that determines the forthcoming job sequence. In §4.2 we present an online strategy that gives a 4-competitive schedule using at most two processors. In §4.3 we show that this strategy leads to a 3.59-competitive schedule when the jobs have unit execution times.

4.1 Lower Bound on the Competitive Factor

Let Π be an online scheduling algorithm for this problem. We set $\psi_b = \psi_\sigma = \psi = 1$ and $E_w = k$, where k is an integer chosen to be sufficiently large. Hence the break-even time $\mathcal{B}$ is also k. We define a *monitor* operation of the adversary $\mathcal{A}$ as follows.

▶ **Definition 6.** When $\mathcal{A}$ **monitors** Π during time interval $[t_0, t_1]$, it checks if Π keeps at least one processor on between time t_0 and t_1. If Π turns off all the processors at some point t between t_0 and t_1, then $\mathcal{A}$ releases an urgent unit job $(t + 1, t + 2)$, forcing Π to turn on at least one processor to process it. If Π keeps at least one processor on during the monitored period, then $\mathcal{A}$ does nothing.

Let x, η, and χ, where $0 \leq x \leq \frac{2}{5}$, be three non-negative parameters to be chosen. The online adversary works as follows. At time 0, $\mathcal{A}$ releases a unit job $(0, \mathcal{B})$ and observes the behavior of Π. Let t be the moment at which Π schedules this job to execute. Since Π produces a feasible schedule, we know that $0 \leq t \leq \mathcal{B} - 1$. We have the following two cases.

Case(1): If $0 \leq t \leq \left(\frac{1}{2} - x\right)\mathcal{B}$, then $\mathcal{A}$ monitors Π from time t to $\frac{3}{2}\mathcal{B}$.
Case(2): If j is not executed till $\left(\frac{1}{2} - x\right)\mathcal{B}$, the adversary $\mathcal{A}$ releases $\left(\frac{1}{2} + x\right)\mathcal{B} - 1$ unit jobs with deadline $\mathcal{B}$ at time $\left(\frac{1}{2} - x\right)\mathcal{B} + 1$. As a result, the online algorithm is forced to switch on at least two processors to execute these jobs. The adversary continues to monitor Π until time $\left(\frac{3}{2} + \eta\right)\mathcal{B}$. If no urgent unit job has been released till time $\left(\frac{3}{2} + \eta\right)\mathcal{B}$, the adversary $\mathcal{A}$ terminates. Otherwise, $\mathcal{A}$ monitors Π for another $\chi\mathcal{B}$ units of time until $\left(\frac{3}{2} + \eta + \chi\right)\mathcal{B}$.

Let $\mathcal{E}(\Pi)$ and $\mathcal{E}(\mathcal{O})$ denote the energy consumed by algorithm Π and an offline optimal schedule on the input sequence generated by $\mathcal{A}$, respectively. By deriving a lower bound on $\mathcal{E}(\Pi)$ and an upper bound on $\mathcal{E}(\mathcal{O})$, we obtain the following theorem.

▶ **Theorem 7.** *The competitive factor of any online algorithm for the online DPM job scheduling problem is at least* 2.06*, even for the case of unit jobs that are known in advance to be packable on one processor.*

4.2 4-Competitive Online Scheduling

In order to design an online algorithm that produces an energy-efficient schedule, we have to deal with two questions. (1) To what extent should we bundle the execution of the jobs in order to achieve energy-efficiency? (2) Provided that the job execution may be delayed, how do we guarantee the feasibility of the resulting schedule?

For the former question, we introduce the concept of *energy-efficient anchors* for the jobs to determine the appropriate timing to begin their execution. For the latter question, the feasibility is guaranteed by suitably partitioning the job set such that the jobs that are delayed still satisfy Condition (1) from Lemma 3. Below we describe our algorithm in more detail. Let $\mathcal{J}$ be the input set of jobs, and recall that $\mathcal{J}(t)$ is the subset of jobs whose arrival times are smaller than or equal to t.

For any $t, t^\dagger$ with $0 \leq t \leq t^\dagger$, let $\mathbf{Q}(t)$ be the subset of $\mathcal{J}(t)$ that contains the jobs that have not yet finished their execution up to time t, and let $\mathbf{Q}(t, t^\dagger)$ be the subset of $\mathbf{Q}(t)$ containing those jobs whose deadlines are smaller than or equal to $t^\dagger$. Note that, by definition, we have $\mathbf{Q}(t, t^\dagger) \subseteq \mathbf{Q}(t) \subseteq \mathcal{J}(t) \subseteq \mathcal{J}$. For notational brevity, let $c'_j(t)$ denote the remaining execution time of job j at time t, and let $W(t) = \sum_{j \in \mathbf{Q}(t)} c'_j(t)$ and $W(t, t^\dagger) = \sum_{j \in \mathbf{Q}(t, t^\dagger)} c'_j(t)$ denote the total remaining execution time of the jobs in $\mathbf{Q}(t)$ and $\mathbf{Q}(t, t^\dagger)$, respectively. Furthermore, we divide $\mathbf{Q}(t)$ into two subsets according to the arrival times of the jobs. For any t, t^* with $0 \leq t^* \leq t$, let $\mathbf{Q}^{t^*}_{proc}(t)$ be the subset of $\mathbf{Q}(t)$ containing the jobs whose arrival times are less than t^*, and let $\mathbf{Q}^{t^*}_{forth}(t) = \mathbf{Q}(t) \backslash \mathbf{Q}^{t^*}_{proc}(t)$.

Let λ, $0 \leq \lambda \leq 1$, be a constant to be determined later. For each job $j \in \mathcal{J}$, we define a parameter h_j to be $\max\{a_j, d_j - \lambda\mathcal{B}\}$. The value h_j is referred to as the *energy-efficient anchor* for job j.

Let M_1 and M_2 denote the two processors which our algorithm $\mathcal{S}$ will manage. We say that the system is *busy*, if at least one processor is executing a job. The system is said to be *off* if both processors are turned off. Otherwise, the system is said to be in *standby*. During the process of job scheduling, our algorithm $\mathcal{S}$ maintains an *urgency flag*, which is initialized to be *false*. The description of the algorithm $\mathcal{S}$ is provided in Table 1.

Table 1 A description of the online scheduling algorithm $\mathcal{S}$.

At any time t, $\mathcal{S}$ proceeds as follows.

(A) Conditions for turning on the processors:	(B) Handling the job scheduling:	(C) Conditions for turning off the processors:
1. **If** the system is *off* and there exists some $j \in \mathbf{Q}(t)$ such that $h_j \leq t$, **then** turn on processor M_1. 2. **If** the *urgency flag* is *false* and there exists some $t^\dagger$ with $t^\dagger > t$ such that $W(t, t^\dagger) > t^\dagger - t$, **then** - turn on M_1 if it is *off*, - turn on M_2, set t^* to be t, and set the *urgency flag* to be *true*.	1. **If** the *urgency flag* is *true*, **then** use the EDF principle to schedule jobs from $\mathbf{Q}^{t^*}_{proc}(t)$ on M_1 and jobs from $\mathbf{Q}^{t^*}_{forth}(t)$ on M_2. 2. **If** the *urgency flag* is *false* and the system is not *off*, **then** use the EDF principle to schedule jobs from $\mathbf{Q}(t)$ on the processor that is *on*.	1. **If** the *urgency flag* is *true* and $\mathbf{Q}^{t^*}_{proc}(t)$ empty, **then** turn off M_1 and set the *urgency flag* to be *false*. 2. **If** the *urgency flag* is *false*, the system is *standby*, and $t - t_1 \geq \mathcal{B}$, where t_1 is the time processor M_1 was turned on, **then** turn off all processors.

Note that, M_1 and M_2 can both be on only when the *urgency flag* is *true*. To prove the claimed competitive factor, we analyze the relative positions between the time intervals during which an optimal offline schedule keeps the system off and our online algorithm is executing jobs. Then we charge the energy consumed by our schedule to that consumed by the optimal offline schedule to obtain the claimed bound.

▶ **Theorem 8.** *By setting $\lambda = 1$, the algorithm $\mathcal{S}$ computes a 4-competitive schedule that uses at most two processors for any set of jobs satisfying Condition (1) for the online DPM job scheduling problem.*

4.3 3.59-Competitive Scheduling for Unit Jobs

When the jobs have unit execution times, we show that we can benefit even more from a properly chosen parameter $\lambda = 4 - \sqrt{10}$. The major difference is that, when the system is in urgency while $\mathbf{Q}^{t^*}_{forth}(t)$ is empty, i.e, processor M_2 is in standby, we use a global earliest-deadline-first scheduling by executing two jobs on M_1 and M_2 instead of keeping one processor in standby, which in turn improves resource utilization. Let $\mathcal{S}^\dagger$ denote the modified algorithm. We have the following theorem.

▶ **Theorem 9.** *By setting $\lambda = 4 - \sqrt{10}$, the algorithm $\mathcal{S}^\dagger$ computes a 3.59-competitive schedule that uses at most two processors for any set of unit jobs satisfying Condition (1) for the online DPM job scheduling problem.*

5 Online Multi-Processor Scheduling

In this section we present results derived for online dynamic power management in multi-processor systems. In §5.1 we generalize the algorithm $\mathcal{S}$ presented in §4.2 for a given set of job streams, each of which delivers a set of jobs that can be scheduled feasibly on one processor. This allows a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule.

Then we consider online dynamic power management for general sets of unit jobs. In §5.2 we prove a lower bound of 2.28 on the competitive factor of any online algorithm. Finally in §5.3 we present an online algorithm that gives a $O(1)$-competitive schedule.

5.1 Trading the Energy-Efficiency with the Number of Processors

In §4.2 we have shown how a stream of jobs satisfying Condition (1) can be scheduled by the algorithm $\mathcal{S}$ to obtain a 4-competitive schedule which uses at most two processors. In this section, we show that, by suitably delaying and bundling the workload, we can generalize the algorithm $\mathcal{S}$ for a given set of job streams, each of which delivers a job set satisfying Condition (1), to allow a trade-off between the number of processors we have and the energy-efficiency of the resulting schedule.

Let $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_k$ be the given set of job streams, where $\mathcal{J}_i$ satisfies Condition (1) for all $1 \leq i \leq k$. Below we present a strategy that leads to a schedule that uses at most h processors for any $h > k$ and whose energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used by any schedule which schedules each of the k streams on a separate processor.

First, if $h \geq 2k$, then we apply the algorithm $\mathcal{S}$ on every pair of the streams, i.e., on $\mathcal{J}_{2i}$ and $\mathcal{J}_{2i+1}$, for all $1 \leq i \leq \frac{k}{2}$, and we get a schedule with a factor of 4. For the case $k < h < 2k$, we divide $\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_k$ into $h-k$ subsets such that each subset gets at most $\lceil k/(h-k) \rceil$ streams. The h processors are allocated in the following way. Each stream of jobs, i.e., $\mathcal{J}_i$ for all $1 \leq i \leq k$, gets one processor, and each of the remaining $h-k$ processors are allocated to each of the $h-k$ subsets such that no two processors are allocated to the same subset. Therefore, the problem is reduced to the remaining case, $h = k+1$.

Below we describe how the case $h = k+1$ is handled. Let $M_0, M_1, \ldots, M_k$ denote the $k+1$ processors to be managed, and let $\mathbf{Q}_i$, $1 \leq i \leq k$, be the corresponding ready queue for processor M_i. We use the parameter $\lambda = 1$ to set the energy-efficient anchor for each job that arrives. Recall that $c'_j(t)$ is the remaining execution time of job j at time t. For any $t \geq 0$ and any $t^\dagger$ with $t^\dagger \geq t$, we use $W_0(t, t^\dagger) = \sum_{j \in \mathbf{Q}_0, d_j \leq t^\dagger} c'_j(t)$ to denote the total remaining execution time of the jobs in $\mathbf{Q}_0$ whose deadlines are less than or equal to $t^\dagger$.

The algorithm works as follows. When a job $j \in \mathcal{J}_i$, $1 \leq i \leq k$, arrives to the system, we check whether or not processor M_i is on. If M_i is on, then we add j to $\mathbf{Q}_i$. Otherwise, we further check whether $W_0(t, t^\dagger) + c_j \leq t^\dagger - t$ holds for all $t^\dagger \geq d_j$. If it does, then j is added to $\mathbf{Q}_0$. Otherwise, we add j to $\mathbf{Q}_i$ and turn on the processors M_i and M_0 (if M_0 is off). At any time t, we have the following further conditions to consider.

(a) Conditions for turning on M_0: If the energy-efficient anchor of some job in $\mathbf{Q}_0$ is met or if $\exists\ t^\dagger \geq t$ such that $W_0(t, t^\dagger) \geq t^\dagger - t$, then we turn on M_0 if it is off.

(b) Job scheduling: For each $0 \leq i \leq k$ such that M_i is on, we use the EDF principle to schedule the jobs of $\mathbf{Q}_i$ on M_i.

(c) Conditions for turning off the processors: If $\mathbf{Q}_0$ becomes empty and M_0 has been turned on for at least $\mathcal{B}$ amount of time, then we turn off M_0 immediately. For $1 \leq i \leq k$, if M_i is on, $\mathbf{Q}_i$ becomes empty, and M_0 is off, then we turn off M_i.

Let $\mathcal{S}_{multi}$ denote the algorithm. We have the following theorem.

▶ **Theorem 10.** *Given a set of k job streams, each of which can be scheduled feasibly on one processor, algorithm $\mathcal{S}_{multi}$ computes a schedule that uses at most h processors, for any $h > k$, such that the energy usage is at most $4 \cdot \left\lceil \frac{k}{h-k} \right\rceil$ times that used in any schedule which schedules each of the k streams on a separate processor, for the online DPM job scheduling problem.*

5.2 Lower Bound on the Competitive Factor

In this section we prove a lower bound of 2.28 on the competitive factor of any online algorithm for the online DPM scheduling problem with unit jobs. Let Π be an online

scheduling algorithm for this problem. More specifically, we present an online adversary $\mathcal{A}$, which observes the behavior of Π and which decides the set of forthcoming jobs such that the competitive ratio of Π on the input sequence generated by $\mathcal{A}$ is at least 2.28.

We set $\psi_b = \psi_\sigma = \psi = 1$ and $E_w = k$, where $k \gg 1$ is an integer chosen to be sufficiently large. Hence the break-even time, $\mathcal{B}$, is also k. The adversary $\mathcal{A}$ works in two stages. In the first stage, $\mathcal{A}$ uses the gadget designed in [24] for the machine-minimizing job scheduling problem to force Π to use more processors than necessary. As a result, at the time when the first stage ends, the number of processors Π uses is at least 2.09 times that required by any optimal schedule for $\mathcal{J}^*$ in terms of number of processors. In the second stage, $\mathcal{A}$ monitors the number of active processors Π keeps and makes sure that Π does not turn off the processors too fast. Below we describe the second stage in more detail.

Stage II. Let η be a non-negative real number to be decided later. We define the real-valued function $f_\eta(t)\colon [0,1] \to \mathbb{R}$ to be $f_\eta(t) = \eta + e^t \cdot (3.09 - 2\eta)$, where e is the base of natural logarithm, i.e., the Euler's number.

The adversary $\mathcal{A}$ works as follows. At each time t with $q\alpha^2 \le t < q\alpha^2 + \mathcal{B}$, the adversary checks if the number of processors algorithm Π keeps in the state of on is at least $\hat{\varrho}(\mathcal{J}^*) \cdot f_\eta\left(\frac{t-q\alpha^2}{\mathcal{B}}\right)$. If it is, then $\mathcal{A}$ does nothing. Otherwise, $\mathcal{A}$ punishes the aggressive behavior of Π by releasing $\lceil \hat{\varrho}(\mathcal{J}^*) \rceil$ urgent jobs with deadline $t+1$ and terminates.

▶ **Lemma 11.** *If the algorithm Π gets punished by $\mathcal{A}$, then the competitive factor of the resulting schedule is at least η.*

Lemma 11 gives a bound when the algorithm Π turns off the processors in an aggressive way. On the other hand, if Π does not behave aggressively, then the resulting competitive factor will decrease as η increases. By setting η to be 2.28, we get the following theorem.

▶ **Theorem 12.** *The competitive factor of any online algorithm for the online DPM job scheduling problem is at least* 2.28*, even for unit jobs.*

5.3 O(1)-Competitive Online Scheduling for Unit Jobs

In this section we consider the case for which the jobs have unit execution times. The lower bound result provided in §5.2 gives a rough idea on the difficulty of this problem, which includes the following. (a) First, how many processors should be used when we have no prior knowledge on future job arrivals? (b) Second, how can we turn the standby processors off so that we do not suffer much when we have to turn them on again later?

We incorporate the results of [24] as a partial solution to question (a) mentioned above and give a $O(1)$-competitive online algorithm, denoted $\mathcal{S}^\dagger_{multi}$, for this problem. The algorithm works as follows. At each moment, $\mathcal{S}^\dagger_{multi}$ computes the density of the workload that has arrived to the system "recently" and makes its scheduling decisions accordingly. If the density is *low*, then $\mathcal{S}^\dagger_{multi}$ adopts the strategy presented in §4.2 and §4.3 to bundle the execution of the jobs on two processors. Otherwise, $\mathcal{S}^\dagger_{multi}$ uses the approach suggested in [24] to estimate the number of processors required by future job arrivals for multi-processor scheduling. For question (b), we let each processor stay on for an additional amount of time before it is turned off.

The approach we use combines ideas from different results. Although the idea is conceivable, bounding the energy efficiency is tricky and requires further non-trivial observations on the connections between online schedules and optimal schedules.

Below we describe the algorithm $\mathcal{S}^{\dagger}_{multi}$ in more detail. Let $\mathcal{J}$ denote the input set of jobs and $\mathbf{Q}$ denote the ready queue which contains the set of jobs that arrive and that are not yet scheduled. The algorithm $\mathcal{S}^{\dagger}_{multi}$ maintains a variable t^*, initialized to be -1, to denote the last time when $\mathbf{Q}$ becomes empty. Let $\mathcal{J}^* = \mathcal{J} \backslash \mathcal{J}(t^*)$ be the set of jobs that arrive after time t^*. In addition, the algorithm $\mathcal{S}^{\dagger}_{multi}$ maintains another variable t^*_h to denote the first time for which the workload density becomes greater than or equal to 1 since time t^*, i.e., t^*_h is the smallest integer such that $t^*_h > t^*$ and $\hat{\varrho}(\mathcal{J}^*(t^*_h)) \geq 1$. For notational brevity, t^*_h is set to be ∞ if there is no such moment.

At each time t, the algorithm $\mathcal{S}^{\dagger}_{multi}$ computes the workload density $\hat{\varrho}(\mathcal{J}^*(t))$ and updates the value of t^*_h if necessary. Depending on the value of t^*_h, the jobs that arrive are handled differently. If $t^*_h > t$, then the jobs that have just arrived are added to the *lightly-loaded ready queue* $\mathbf{Q}_\ell$. Otherwise, they are added to the *heavily-loaded ready queue* $\mathbf{Q}_h$.

For the jobs that are added to $\mathbf{Q}_\ell$, we use the algorithm $\mathcal{S}^{\dagger}$ proposed in §4.3 to schedule them. To help describe the algorithm, in the following we use M_1 and M_2 to denote the two specific processors that are used by $\mathcal{S}^{\dagger}$. In addition, we use $\#_i$, where $i \geq 1$, to denote the remaining processors that will be used to handle the jobs that are added to $\mathbf{Q}_h$.

Handling the *heavily-loaded ready queue* $\mathbf{Q}_h$. Let γ_2 be a constant chosen to be 5.2. If $t^*_h > t$, then $\mathbf{Q}_h(t)$ is empty and there is nothing to process. If $t^*_h \leq t$, then the algorithm $\mathcal{S}^{\dagger}_{multi}$ makes sure that at least $\lceil \gamma_2 \cdot \hat{\varrho}(\mathcal{J}^*(t)) \rceil$ processors, excluding M_1 and M_2, have been turned on for job execution. Let $\#(t)$ be the number of processors that are on, excluding M_1 and M_2, and let $\chi = \min\{\#(t), |\mathbf{Q}_h(t)|\}$. We remark that, as $\hat{\varrho}(\mathcal{J}^*(t))$ changes over time, it is possible that $\#(t) > \lceil \gamma_2 \cdot \hat{\varrho}(\mathcal{J}^*(t)) \rceil$.

The algorithm $\mathcal{S}^{\dagger}_{multi}$ fetches χ jobs with earliest deadlines from $\mathbf{Q}_h(t)$ and assigns them for execution on $\#_1, \#_2, \ldots, \#_\chi$. If $|\mathbf{Q}_h(t)| < \#(t)$, then $\mathcal{S}^{\dagger}_{multi}$ continues to fetch jobs from $\mathbf{Q}_\ell(t)$, if there exists any, using the first-fit principle, i.e., the processor with smaller index has higher priority for job execution, such that either all of the $\#(t)$ processors are occupied or $\mathbf{Q}_\ell(t)$ becomes empty.

Turning off the processors. After the scheduling decisions on the ready queues, i.e., $\mathbf{Q}_\ell$ and $\mathbf{Q}_h$, are made, the algorithm $\mathcal{S}^{\dagger}_{multi}$ checks the following conditions. For all i with $1 \leq i \leq \#(t)$, if processor $\#_i$ has stayed in standby for $\mathcal{B}$ amount of time since turned on, then $\mathcal{S}^{\dagger}_{multi}$ switches processor $\#_i$ off immediately.

At any time t, if the ready queue $\mathbf{Q}$ becomes empty after the scheduling decisions on $\mathbf{Q}_\ell$ and $\mathbf{Q}_h$ are made, then t^* is set to t and t^*_h is set to be ∞. We conclude the result with the following theorem.

▶ **Theorem 13.** *The algorithm $\mathcal{S}^{\dagger}_{multi}$ computes a $(\gamma_1 + 52 \cdot \gamma_2 + 1)$-competitive schedule for the online DPM job scheduling problem with unit jobs, where $\gamma_1 = 3.59$ is the competitive factor of $\mathcal{S}^{\dagger}$ and $\gamma_2 = 5.2$ is a constant.*

References

1 Susanne Albers and Antonios Antoniadis. Race to idle: new algorithms for speed scaling with a sleep state. In *SODA*, pages 1266–1285, 2012.

2 Susanne Albers, Antonios Antoniadis, and Gero Greiner. On multi-processor speed scaling with migration: extended abstract. In *SPAA*, pages 279–288, 2011.

3 S. Anand, Naveen Garg, and Nicole Megow. Meeting deadlines: How much speed suffices? In *ICALP (1)*, volume 6755 of *LNCS*, pages 232–243. Springer, 2011.

4 John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. In *FOCS*, pages 530–539, 2004.

5 P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: A polynomial time algorithm for offline dynamic power management. In *SODA*, pages 364–367, 2006.

6 Philippe Baptiste, Marek Chrobak, and Christoph Dürr. Polynomial time algorithms for minimum energy scheduling. In *ESA*, pages 136–150, 2007.

7 Sanjoy K. Baruah and Nathan Fisher. The partitioned multiprocessor scheduling of sporadic task systems. In *RTSS*, pages 321–329, 2005.

8 V. Bonifaci, A. Marchetti-Spaccamela, and S. Stiller. A constant-approximate feasibility test for multiprocessor real-time scheduling. In *ESA*, pages 210–221, 2008.

9 Jian-Jia Chen and S. Chakraborty. Resource augmentation bounds for approximate demand bound functions. In *RTSS*, pages 272–281, 2011.

10 Jian-Jia Chen and S. Chakraborty. Partitioned packing and scheduling for sporadic real-time tasks in identical multiprocessor systems. In *ECRTS*, pages 24–33, 2012.

11 Jian-Jia Chen and Tei-Wei Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. In *LCTES*, 2006.

12 Jian-Jia Chen and Tei-Wei Kuo. Procrastination determination for periodic real-time tasks in leakage-aware dynamic voltage scaling systems. In *ICCAD*, 2007.

13 Houssine Chetto and Maryline Silly-Chetto. Scheduling periodic and sporadic tasks in a real-time system. *Inf. Process. Lett.*, 30(4):177–184, 1989.

14 Julia Chuzhoy and Paolo Codenotti. Resource minimization job scheduling. In *APPROX'09/RANDOM'09*, pages 70–83, Berlin, Heidelberg, 2009. Springer-Verlag.

15 Julia Chuzhoy, Sudipto Guha, Sanjeev Khanna, and Joseph Naor. Machine minimization for scheduling jobs with interval constraints. In *FOCS'04*, 2004.

16 M. Cieliebak, T. Erlebach, F. Hennecke, B. Weber, and P. Widmayer. Scheduling with release times and deadlines on a minimum number of machines. In *IFIP*. Springer, 2004.

17 E. Demaine, M. Ghodsi, M. Hajiaghayi, A. Sayedi-Roshkhar, and M. Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *SPAA*, pages 46–54, 2007.

18 Erik D. Demaine and Morteza Zadimoghaddam. Scheduling to minimize power consumption using submodular functions. In *SPAA*, pages 21–29, 2010.

19 Michael L. Dertouzos. Control robotics: The procedural control of physical processes. In *IFIP Congress*, pages 807–813, 1974.

20 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, 1979.

21 Kai Huang, Luca Santinelli, Jian-Jia Chen, Lothar Thiele, and Giorgio C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Real-Time Systems*, 47(2):163–193, 2011.

22 Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *SODA*, pages 37–46, 2003.

23 Sandy Irani, Sandeep K. Shukla, and Rajesh K. Gupta. Online strategies for dynamic power management in systems with multiple power-saving states. *ACM Trans. Embedded Comput. Syst.*, 2(3):325–346, 2003.

24 M.-J. Kao, J.-J. Chen, I. Rutter, and D. Wagner. Competitive design and analysis for machine-minimizing job scheduling problem. In *ISAAC*, pages 75–84, 2012.

25 A. Karlin, M. Manasse, L. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

26 Yann-Hang Lee, Krishna P. Reddy, and C. M. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *ECRTS*, pages 105–112, 2003.

27 A. K. Mok. Fundamental design problems of distributed systems for the hard-real-time environment. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

28 Cynthia A. Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation (extended abstract). In *STOC*, pages 140–149, 1997.

29 L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. *ISCAS*, 4:101–104, 2000.

30 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, pages 374–382, 1995.