Energy Efficient Deadline Scheduling in Two Processor Systems

Tak-Wah Lam[1], Lap-Kei Lee[1], Isaac K. K. To[2,⋆], and Prudence W. H. Wong[2,⋆]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
{twlam,lklee}@cs.hku.hk
[2] Department of Computer Science, University of Liverpool, UK
{isaacto,p.wong}@liverpool.ac.uk

Abstract. The past few years have witnessed different scheduling algorithms for a processor that can manage its energy usage by scaling dynamically its speed. In this paper we attempt to extend such work to the two-processor setting. Specifically, we focus on deadline scheduling and study online algorithms for two processors with an objective of maximizing the throughput, while using the smallest possible energy. The motivation comes from the fact that dual-core processors are getting common nowadays. Our first result is a new analysis of the energy usage of the speed function OA [15,4,8] with respect to the optimal two-processor schedule. This immediately implies a trivial two-processor algorithm that is 16-competitive for throughput and $O(1)$-competitive for energy. A more interesting result is a new online strategy for selecting jobs for the two processors. Together with OA, it improves the competitive ratio for throughput from 16 to 3, while increasing that for energy by a factor of 2. Note that even if the energy usage is not a concern, no algorithm can be better than 2-competitive with respect to throughput.

1 Introduction

Energy usage is an important concern in the design of modern processors. To be more energy efficient, many modern processors adopt the technology of dynamic speed scaling, where the processor can adjust its speed dynamically in some range without any overhead. In general, running a processor at speed s would consume energy at the rate s^α, where α is a constant believed to be 2 or 3 (see, e.g., [7,9,13,14]). That is, a processor can save energy by running slower. This energy saving capability has triggered a lot of work to revisit processor scheduling; the concern is how to exploit this capability to reduce the energy usage, while achieving as much as possible the original scheduling objectives (such as throughput and flow time).

The pioneering work along this direction was due to Yao et al. [15]. They considered the problem of deadline scheduling on a processor that can vary

⋆ This research is partly supported by EPSRC Grant EP/E028276/1.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 476–487, 2007.

its speed between 0 and infinity. We call this the infinite speed model. In this model, it is always feasible to complete all jobs by their deadlines; the concern is how to adjust the processor speed and whether the total energy usage can be $O(1)$-competitive. Yao et al. answered in the affirmative by showing an online algorithm called AVR to be $2^{\alpha-1}\alpha^{\alpha}$-competitive, and they proposed another algorithm called OA (Optimal Available), which is later proved by Bansal et al. [4] to be α^{α}-competitive. Bansal et al. also gave a new algorithm that is $2(\alpha/(\alpha-1))^{\alpha}e^{\alpha}$ competitive. Note that all these algorithms only give a speed function for the processor; all jobs are scheduled in an EDF (Earliest Deadline First) manner. Recently several interesting results on flow time scheduling have also been revealed under the infinite speed model (see [1,5,10]).

The infinite speed model is a convenient model for studying different speed functions and their energy usage. Yet in reality, the speed of a processor is bounded. Chan et al. [8] recently initiated the study of the bounded speed model, where a processor can vary its speed between 0 and a fixed maximum speed T. In this model, deadline scheduling becomes more complicated and even the optimal offline algorithm may not be able to complete all jobs. A natural objective is to maximize the throughput (i.e., the total work of jobs that can be completed by their deadlines), while using the smallest possible energy. They showed that the energy demanded by the speed function OAT, which is simply OA capped at T, is at most $\alpha^{\alpha} + 4^{\alpha}\alpha^{2}$ times of the energy of any offline algorithm that produces the maximum throughput. They further showed that OAT can support a job selection strategy called FSA to be 14-competitive for throughput. More recently, Bansal et al. [3] showed that OAT is indeed "fast" enough to support a more aggressive job selection strategy called Slow-D, thus improving the competitive ratio from 14 to 4. Note that even if the energy issue is ignored, no algorithm can be better than 4-competitive with respect to throughput [6].

Many modern processors are indeed dual-core (or even quad-core) processors. It is natural to extend the study of energy efficient scheduling to the multi-processor setting. In this paper we take the first step to investigate the case of two processors, and we would like to derive an online algorithm with a throughput that can match or is close to the known lower bound, while being $O(1)$-competitive on energy. In the infinite speed model, it is relatively trivial to derive a two-processor algorithm that can complete all jobs (1-competitive for throughput) and is $O(1)$-competitive for energy (see explanation below). It is the bounded speed model that needs attention. In this model, if energy is not a concern, it has been known that the two-processor algorithm Safe-Risky is 2-competitive for throughput [11], and no online algorithm can be better than 2-competitive [12]. Thus, it is natural to ask for an online algorithm for two processors that is 2 or close to 2 competitive for throughput and $O(1)$-competitive for energy.

To ease our discussion, we let Opt_1 to denote the optimal offline algorithm for a single processor that maximizes the throughput, while using the smallest energy, and similarly Opt_2 for the case of two processors. Furthermore, let $E_{\text{Opt}i}$ and $W_{\text{Opt}i}$ denote respectively the energy usage and throughput of Opt_i, where

$i = 1$ or 2. First of all, let us explain why scheduling two processors is relatively trivial in the infinite speed model. When the processor speed is unbounded, all jobs can be completed on time even using one single processor. Furthermore, we can use a single processor, which runs sufficiently fast, to simulate any two-processor schedule that completes all jobs; the energy usage increases by at most $2^{\alpha-1}$ times. Thus, $E_{\text{Opt1}} \leq 2^{\alpha-1}E_{\text{Opt2}}$. Given two processors, a trivial algorithm is to use OA for one processor and let the other idle. This would give an algorithm that is 1-competitive for throughput and $2^{\alpha-1}\alpha^\alpha$-competitive for energy.

The rest of this paper is devoted to the bounded speed model. First of all, let us look at the performance of the trivial algorithm that uses Slow-D and OAT on one processor, leaving the other idle. The simulation argument above is no longer valid in the bounded speed model, yet it is probably still true that $E_{\text{Opt1}} \leq 2^{\alpha-1}E_{\text{Opt2}}$, implying that the energy usage of OAT (and Slow-D) is at most $(2^{\alpha-1}\alpha^\alpha + 2^{3\alpha-1}\alpha^2)E_{\text{Opt2}}$. In this paper we give a better analysis of OAT, showing that the energy usage of OAT is at most

$$(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2)E_{\text{Opt2}}.$$

For the throughput, we can easily argue that $W_{\text{Opt1}} \geq W_{\text{Opt2}}/4$ (using the notion of minimally infeasible job set given in [8]). Thus, the trivial two-processor algorithm is 16-competitive for throughput and $O(1)$-competitive for energy.

The above competitive ratio for throughput is far from the lower bound of 2. This motivates us to develop a non-trivial strategy to schedule jobs with both processors. We develop a new job selection strategy called Slow-SR, with one processor following a schedule similar to Slow-D and the other working like the Risky processor of the Safe-Risky algorithm. The competitive ratio of throughput is reduced from 16 to 3. Both processors are running at a speed not exceeding that of OAT and the competitive ratio for energy becomes $2^\alpha\alpha^\alpha + 4^\alpha\alpha^2$.

In conclusion, we have devised a two-processor algorithm for the bounded speed model, which is 3-competitive for throughput and $2^\alpha\alpha^\alpha+4^\alpha\alpha^2$-competitive for energy. As to be explained, our algorithm requires job migration from the Risky processor to the Slow-D processor. It is interesting to find a non-migratory algorithm with similar performance (though in a dual-core processor, the two CPUs share the same memory, so the overhead of migration is not as expensive as the typical distributed model). We believe that for $m \geq 2$ processors, it is possible to have an algorithm that is $O(1)$-competitive for both throughput and energy usage. Yet it is hard to generalize the algorithm given in this paper. On the other hand, Albers et al. [2] have recently obtained some interesting results on multiprocessor deadline scheduling for the infinite speed model. They focused on the special case where jobs have agreeable deadlines (i.e., the deadlines of the jobs follow the order of their release times), and they showed a non-migratory multiprocessor algorithm that is $\alpha^\alpha 16^\alpha$-competitive for energy (recall that in the infinite speed model, the competitive ratio of throughput is always one).

Organization of the paper. In Section 2, we describe the algorithm Slow-SR. In Section 3, we establish a couple of key properties of the algorithm. Using

these properties, we analyze the throughput of Slow-SR in Section 4. Finally, we analyze the energy competitiveness of OAT in Section 5.

Definitions and Assumptions. Before we give the details of Slow-SR, let us define the scheduling problem formally. There are two processors, the speed of each can be independently scaled from 0 to a maximum speed T. A processor running at speed s can process s units of work per unit of time, consuming energy at the rate s^α. Jobs are released in an online fashion. Each job j is characterized by a release time r_j, a deadline d_j, and required work (or size) p_j. At any time, a job can be executed in only one of the two processors. Preemption and migration is allowed with no penalty. A job j is said to be completed if p_j units of work has been processed by its deadline. Note that there may be too many jobs to be completed. The objective of a scheduler is to maximize the throughput, while minimizing the energy. An online algorithm is said to be c-competitive for throughput and c'-competitive for energy if, for any job sequence, its throughput is at least $\frac{1}{c}W_{\text{Opt2}}$ and its energy usage is at most $c'E_{\text{Opt2}}$. To simplify our discussion, we assume that some suitable scaling has been done to the processor speed and job size so that $T = 1$.[1]

2 The Slow-SR Algorithm

The algorithm Slow-SR makes reference to the schedule of OA. Before describing the algorithm Slow-SR, we need a review of OA.

OA works for one processor only. We characterize OA by the schedule it plans to use at any time t, assuming no more jobs are released after t. Let $I(t)$ be the subset of jobs that has arrived up to time t. We use $\rho(t_1, t_2)$ to denote the remaining work of the jobs in $I(t)$ with deadlines in the interval $(t_1, t_2]$. The speed function that OA plans to use looks like a staircase, with speed reduced at certain "critical" times c_0, c_1, ... defined as follows. Let $c_0 = t$. For any i, the speed that OA plans to use immediately after c_i is $\rho_{i+1} = \max_{t' > c_i} \rho(c_i, t')/(t' - c_i)$; OA maintains this speed until c_{i+1}, defined as the earliest time after c_i such that $\rho(c_i, c_{i+1})/(c_{i+1} - c_i) = \rho_{i+1}$. The intervals $[c_i, c_{i+1}]$ are called the critical intervals. Within each critical interval, jobs with deadlines in this interval are executed in an EDF order. Intuitively, OA is very lazy; within each critical interval, OA just uses the minimum speed that would not cause any job to miss a deadline. Note that if no more jobs are released, OA never changes the speed planned as above.

Consider any time t. With respect to the speed function planned by OA at time t, we define $t_{\text{slow}} \geq t$ to be the first time when OA plans to use speed 1 or less. Furthermore, we say that t is a "slow time" if OA actually runs at speed 1

[1] Given a job set I to be scheduled on a processor with maximum speed $T > 1$, we define another job set I' by scaling the work of each job j in I to p_j/T. Then any schedule for I' with maximum speed 1 can be transformed to a schedule for I with maximum speed T (by increasing the speed by a factor of T), and vice versa. Therefore, the competitive ratios for throughput and energy both preserve.

Data Structures:
Q_{slow}, initially $\emptyset$
Q_{fast}, initially $\emptyset$
J_{uc}, initially *nil*
Scheduling framework:
At fast time, SP runs earliest deadline job in Q_{fast} at speed 1
At slow time, SP runs earliest deadline job in Q_{slow} at the same speed as OA
If $J_{\text{uc}} \neq nil$, RP runs J_{uc} at speed 1
Event: Release of job j
1: Update the simulated OA schedule by adding j
2: Move jobs in Q_{slow} with deadline at or before t_{slow} to Q_{fast}
3: **if** $d_j > t_{\text{slow}}$ **then**
4: Add j to Q_{slow}
5: **else if** $\{j\} \cup Q_{\text{fast}}$ is feasible **then**
6: Add j to Q_{fast}
Event: LST interrupt of job j
7: **if** $J_{\text{uc}} = nil$ or $p(j) > p(J_{\text{uc}})$ **then**
8: $J_{\text{uc}} \leftarrow j$
Event: Job completion in SP
9: Remove the completed job from Q_{slow} or Q_{fast}
10: **if** $J_{\text{uc}} \neq nil$ and $Q_{\text{fast}} = \emptyset$ **then**
11: $Q_{\text{fast}} \leftarrow \{J_{\text{uc}}\}$
12: $J_{\text{uc}} \leftarrow nil$

Algorithm 1. Slow-SR

or less at t, and a "fast time" otherwise. By definition, OA plans to change speed at t_{slow}, and t_{slow} is a critical time. That means, OA plans to execute only jobs with deadlines at or before t_{slow} until t_{slow}.

Slow-SR is defined in Algorithm 1.The two processors are labeled as SP (Slow-D Processor) and RP (Risky Processor), respectively. The algorithm keeps two queues (sets), both on jobs committed to run on SP. The queue Q_{slow} contains jobs that OA plans to run "slowly" (at speed 1 or less), and the queue Q_{fast} for jobs that OA plans to run "quickly" (at speed over 1). The algorithm also keeps a job J_{uc} committed to run on RP. For each job j not in Q_{fast} or Q_{slow}, an LST (latest-start-time) interrupt will occur at time $d_j - p_j$, which attempts to retain job j as J_{uc} for execution.

3 Relations of the Job Queues

We establish two main characteristics of Slow-SR in this section, namely that SP completes all jobs that have ever entered Q_{fast} and Q_{slow}, and that RP is busy only during fast time. For any time t, let $t_{\text{slow}}(t)$ denotes t_{slow} at t, before events at t (if any) are taken into account. A job j is said to be *active* at time t if $t < d_j$ and j has not yet been completed by t. Before we begin, it is essential for us to understand the following implications of slow time.

Lemma 1. *At any slow time, all active jobs are in* Q_{slow}.

Proof. Let t be a slow time. So $t_{\text{slow}}(t) = t$. For any active job j, $d_j > t = t_{\text{slow}}(t) \geq t_{\text{slow}}(t')$ for any $t' \in [r_j, t]$, where the last inequality follows from the nature of OA that $t_{\text{slow}}(t)$ increases monotonically with t. Thus j entered Q_{slow} at release, and is not moved to Q_{fast} at or before t. So all active jobs are in Q_{slow}. □

Lemma 2. *The schedules of OA and Slow-SR during slow time are exactly the same (assuming they break ties of deadlines in the same way).*

Proof. Since Slow-SR uses the speed of OA during slow time, we only need to consider the choices of jobs of the two schedulers. Note that during slow time, both schedulers always choose the earliest deadline active job (for Slow-SR this is because Lemma 1 guarantees that Q_{slow} contains all active jobs). So it suffices to show that they have the same set of active jobs. Let t be the first slow time that OA and Slow-SR have different set of active jobs, and thus can have different scheduling. Let j be any job active at t in either scheduler, we will show that it is active in both schedulers, meaning the active job sets are the same, i.e., contradiction. We just need to observe that neither scheduler runs j during fast time before t, for OA it is because $d_j > t = t_{\text{slow}}(t) \geq t_{\text{slow}}(t')$ for any $t' < t$, so it has too late deadline; and for Slow-SR it is because the above implies that j has never been in Q_{fast}. In other words, scheduling of j before t must be exactly the same in the two schedulers, so it must be active in both. □

We now prove the two main properties of Slow-SR.

Property 1. SP completes all jobs that have ever entered Q_{fast} or Q_{slow}.

Proof. For jobs that are added to Q_{slow} at release and are never moved to Q_{fast}, Lemma 2 guarantees that their scheduling is the same as OA and will be completed. We now show that Q_{fast} remains feasible (using a speed 1 processor) at any time, which implies that each job in Q_{fast} is feasible at their deadlines, i.e., already completed at their deadlines. We first check that Q_{fast} does not become infeasible without jobs being added to it: if Q_{fast} is non-empty but feasible at t, there must be active jobs in Q_{fast}, and Lemma 1 implies that it is fast time. SP thus runs the earliest deadline job in Q_{fast} using speed 1, so Q_{fast} is feasible immediately after t.

Now we turn to the cases when jobs enter Q_{fast}. For cases where there is a feasibility check, the feasibility after the job entering Q_{fast} is trivial. The only remaining case is when jobs are moved from Q_{slow} to Q_{fast}. If Q_{fast} is feasible before such a move, then $Q_{\text{fast}} \cup Q_{\text{slow}}$ must also be feasible before such a move since their time of execution do not conflict: jobs in Q_{fast} have deadlines at or before $t_{\text{slow}}(t)$ and thus can only be processed at or before $t_{\text{slow}}(t)$, while scheduling of Q_{slow} is always the same as OA (Lemma 1) which plans to use time after $t_{\text{slow}}(t)$ to work on Q_{slow}. It is then obvious that the move does not cause infeasibility. □

Property 2. If $J_{\text{uc}} \neq nil$ at t, then (1) $Q_{\text{fast}} \neq \emptyset$, and (2) t must be a fast time.

Proof. Suppose at time t, $J_{\text{uc}} = j$ and $Q_{\text{fast}} = \emptyset$. So j must have caused an LST interrupt at some time $t_l < t$, and remains as J_{uc} since then. If Q_{fast} is non-empty at t_l, there must be a time in (t_l, t) at which Q_{fast} changes from non-empty to empty, when j would be moved to Q_{fast}, contradicting that $J_{\text{uc}} = j$ at t. But if Q_{fast} is empty at t_l, it means j can be completed with all jobs completed in Q_{fast} by t_l, so j should enter Q_{fast} at r_j, contradicting that J caused an LST interrupt. Therefore, $Q_{\text{fast}} \neq \emptyset$. This also implies that some active job is in Q_{fast} (Property 1), so by Lemma 1, the current time is a fast time. □

4 Throughput Competitiveness

We analyze the throughput of Slow-SR in this section. This is done by partitioning the set of jobs into two categories: L_s includes those that enter Q_{slow} on release, and L_f includes all other jobs. The jobs in L_s are all completed by Slow-SR (Property 1), while Slow-SR can miss the deadlines of some jobs in L_f. Note that the spans (i.e., time interval between release time and deadline) of jobs in L_f are completely in fast time, putting a bound on the amount of work that the optimal algorithm can complete for L_f. Let f denote the total length of periods of fast time. The core of the proof is Lemma 5, allowing us to show that Slow-SR completes f units of work, resulting into the following theorem.

Theorem 1. *Slow-SR is 3-competitive on throughput.*

Proof. The optimal offline schedule can at most complete all jobs in L_s and $2f$ units of work in L_f, leading to a total throughput of $W_{\text{Opt}} \leq p(L_s) + 2f$. Slow-SR completes all jobs in L_s, so its throughput $W_{\text{Slow-SR}} \geq p(L_s)$. We will show in Lemma 3 that for jobs with deadlines in each maximal period of fast time F, Slow-SR completes an amount of work no less than the length of F. Summing over all such maximal periods, it means for jobs with deadlines in fast time, Slow-SR completes no less than f units of work, leading to $W_{\text{Slow-SR}} \geq f$. In conclusion, $W_{\text{Opt}} \leq W_{\text{Slow-SR}} + 2W_{\text{Slow-SR}} \leq 3W_{\text{Slow-SR}}$, completing the proof. □

It is tricky to show that Slow-SR completes enough work during each maximal period of fast time F. Although all jobs in SP meet deadlines (Property 1), and SP always works at speed 1 during fast time, Slow-SR can be idle in some fast time. So we consider each busy period within each maximal period of fast time. Let $t_0, t_1, \ldots, t_l$ be the list of times in F from earliest to latest when one of the following happens: (1) F begins, (2) F ends, or (3) SP switches from idle to busy. So $F = [t_0, t_l]$. Note that some job runs in SP at t_0, since some jobs with deadlines no later than t_{slow} must be released at the beginning of any fast time period, so if Q_{fast} is not occupied by some previously released jobs, it must accept one of those newly released jobs.

Lemma 3. *For jobs with deadlines in a maximal period of fast time F, Slow-SR completes an amount of work no less than the length of F.*

Proof. In Lemma 5, we will show that the amount of work completed before t_n for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$, plus the amount of work completed during $[t_{n-1}, t_n]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is at least $t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1})$. Now sum over all n from 1 to l. We can check that no work completed by Slow-SR is being counted more than once, and all of them are jobs with deadlines in F. So the work completed for jobs with deadlines in F is at least $\sum_{n=1}^{l} t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1}) = t_{\text{slow}}(t_l) - t_{\text{slow}}(t_0) = t_l - t_0$, i.e., the length of F. □

To establish Lemma 5, we depend on two facts. The first is that whenever a job j cannot enter either Q_{fast} or Q_{slow} because a job j' in $Q_{\text{fast}} \cup \{j\}$ would miss deadline, SP and RP of Slow-SR must be able to process an amount of useful work no less than $d_{j'} - r_j$. We say j' *repudiates* j in such cases. The second is a property of OA concerning $t_{\text{slow}}(t)$: the amount of work in jobs already released at t with deadlines between $t' \geq t$ and $t_{\text{slow}}(t)$ cannot be small.

Lemma 4. *Let t be a fast time. For any $t' \in [t, t_{\text{slow}}(t))$, the amount of work in jobs with deadlines in $(t', t_{\text{slow}}(t)]$ released before t is more than $t_{\text{slow}}(t) - t'$.*

Proof. Immediately before t, consider the jobs that OA may plan to run during $[t', t_{\text{slow}}(t)]$. It cannot run jobs with deadlines at or before t', because these jobs have deadlines passed. By the nature of OA, it does not plan to run jobs with deadlines after $t_{\text{slow}}(t)$ until $t_{\text{slow}}(t)$. So it can only plan to run jobs with deadlines $[t', t_{\text{slow}}(t)]$. Yet it plans to use faster than speed 1 during that whole period (so that the period is fast). The lemma arrives immediately. □

Lemma 5. *Consider any $n \in \{1, \ldots, l\}$. The amount of work completed before t_n for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$, plus the amount of work completed during $[t_{n-1}, t_n]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is at least $t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1})$.*

Proof. For $t \in [t_{n-1}, t_n]$, let $P(t)$ be the following proposition: If all jobs with release time at or after t are not released, the amount of work completed by Slow-SR before $t_{\text{slow}}(t)$ for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t)]$, plus the amount of work completed by Slow-SR during $[t_{n-1}, t_{\text{slow}}(t)]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is no less than $t_{\text{slow}}(t) - t_{\text{slow}}(t_{n-1})$.

When t is set to be the last busy time t_b in $[t_{n-1}, t_n]$, this is exactly Lemma 5: First, the work that would be done before $t_{\text{slow}}(t_b)$ if no more jobs are released at or after t_b is exactly the same as the amount of work actually completed before t_b (since Slow-SR idles afterwards). Second, $t_{\text{slow}}(t_b)$ is the same as $t_{\text{slow}}(t_n)$ since it cannot change between t_b and t_n, without causing a job to be accepted into Q_{fast} and extending the current busy period or starting a new one.

The base case $t = t_{n-1}$ is trivial. Suppose $P(t)$ is true for all $t \in [t_{n-1}, u)$. We now establish $P(u)$. It only uses release times less than t, so the transfinite induction works (i.e., it always terminates after a finite number of steps).

Let S be the set of jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(u)]$ already released before u. By Lemma 4, the amount of work in S must be more than $t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$. If all jobs in S enter Q_{fast} or Q_{slow} on release, $P(u)$ is satisfied

by just these jobs. We thus assume that some jobs in S are not accepted into Q_{fast} or Q_{slow} on release, i.e., some repudiation occurred during $[t_{n-1}, u)$. Let j_r be the latest deadline repudiating job during these repudiations, which occurred at $t_r < t$. The set of jobs Γ with deadlines in $(d_{j_r}, t_{\text{slow}}(u)]$ released before u must all be accepted into Q_{fast} or Q_{slow} on release, since no job can repudiate them. This also shows that d_{j_r} must be larger than $t_{\text{slow}}(t_{n-1})$, otherwise all jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$ must be admitted into Q_{fast} or Q_{slow}, contradicting our assumption.

If no job in Γ is run at any fast time before r_{j_r}, we can check that the total completed work counted in $P(u)$ is at least $t_{\text{slow}}(u) - t_{n-1} > t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$, so $P(u)$ is satisfied. The amount of work $t_{\text{slow}}(u) - t_{n-1}$ comes from three disjoint parts of useful work processed by SP and RP: (1) Those processed before $t_{\text{slow}}(u)$ with deadlines in $(d_{j_r}, t_{\text{slow}}(u)]$—at least $t_{\text{slow}}(u) - d_{j_r}$ by Lemma 4; (2) Those processed by SP during $[t_{n-1}, t_r]$—exactly $t_r - t_{n-1}$; and (3) Those with deadlines at or before d_{j_r} that are planned to be processed by SP at the repudiation time t_r, plus the job that is being repudiated or the replacement job that eventually get completed by RP (with or without the help of SP)—the repudiation implies this to be at least $d_{j_r} - t_r$.

Finally, we consider the case if some job $j_e \in \Gamma$ runs at some fast time before r_{j_r}. Let t_e be a time immediately after such execution. Note that $t_e \geq t_{n-1}$: otherwise j_e must be in Q_{fast} and partially executed immediately before t_{n-1}, contradicting that SP is slow or idle by then. By $P(t_e)$ (induction hypothesis), the amount of work counted by $P(t_e)$ is no less than $t_{\text{slow}}(t_e) - t_{\text{slow}}(t_{n-1})$. To see $P(u)$ is satisfied, we note that $P(u)$ includes the work completed for jobs with deadlines in $(t_{\text{slow}}(t_e), t_{\text{slow}}(u)]$, which is not counted in $P(t_e)$. This is more than $t_{\text{slow}}(u) - t_{\text{slow}}(t_e)$ by Lemma 4, so the amount of work counted by $P(u)$ is more than $t_{\text{slow}}(t_e) - t_{\text{slow}}(t_{n-1}) + t_{\text{slow}}(u) - t_{\text{slow}}(t_e) = t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$. □

5 Energy Competitiveness

We analyze the energy consumption of the single processor schedule OA capped at a maximum speed of 1 (OAT) in this section, showing that it is $(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2)$-competitive in energy against the minimum energy 2-processor offline schedule that achieves the maximum throughput. By Property 2, the speeds of both processors of Slow-SR never exceed that of OAT, so this implies that Slow-SR is $(2^\alpha\alpha^\alpha + 2^{2\alpha}\alpha^2)$-competitive in energy against this offline schedule. The competitive ratio of OAT also implies that Slow-D [3] is $(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2)$-competitive in energy in the same setting.

The analysis of OAT against optimal 2-processor schedule is a modification of the proof presented in [8]. Let's review the notations used. At any time t, we use $E_{\text{OAT}}(t)$ and $E_{\text{Opt}}(t)$ to denote the amount of energy already spent by OAT and OPT respectively. We overload the speed function OAT with an actual schedule that always executes the same job as OA, using the OAT speed function (i.e., capped at speed 1). This way, OAT always processes a job whenever its speed is non-zero, but may not process enough work to complete some jobs. In contrast,

OPT never processes a job without eventually completing it. We call work that is done for a job completed by OPT to be type-1 work, and other work to be type-0 work.

Consider OA at any time t. We use two functions $\phi(t)$ and $\beta(t)$ (as in [8]). The function $\phi(t)$ is 0 at the beginning and the end of the schedule, and changes continuously except when jobs are released. The function $\beta(t)$ is α^2 times the amount of type-0 work that would be completed by OAT if no more jobs are released after t. Our main theorem and the outline of its proof is as follows.

Theorem 2. $E_{\mathrm{OAT}} \leq (2^{\alpha-1}\alpha^{\alpha} + 2^{2\alpha-1}\alpha^2)E_{\mathrm{Opt}}$.

Proof. We will show that $E_{\mathrm{OAT}}(t)+\phi(t)-\beta(t) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)$. Before showing how to prove the inequality, let's consider its consequence. Consider a time t_e after all job deadlines. At that time, $E_{\mathrm{OAT}}(t_e) = E_{\mathrm{OAT}}$, $E_{\mathrm{Opt}}(t_e) = E_{\mathrm{Opt}}$, and $\phi(t_e) = 0$. So $E_{\mathrm{OAT}} - \beta(t_e) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}$. Lemma 6 will show that $\beta(t_e) \leq 2^{2\alpha-1}\alpha^2 E_{\mathrm{Opt}}$. The theorem arrives immediately.

We now prove that $E_{\mathrm{OAT}}(t) + \phi(t) - \beta(t) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)$. Before any job is released, the inequality holds trivially, since all the terms are 0. Lemma 7 will show that when no job is being released, the rate of changes of the terms in the inequality satisfies $E_{\mathrm{OAT}}(t)'+\phi(t)'-\beta(t)' \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)'$. Lemma 8 will show that when a job is released, the change of the terms in the inequality satisfies $\Delta\phi(t) - \Delta\beta(t) \leq 0$. $E_{\mathrm{OAT}}(t)$ and $E_{\mathrm{Opt}}(t)$ obviously remain unchanged. Thus no event invalidates the inequality since the time before any job is released. □

Now we bound the amount of type-0 work that OAT eventually completes.

Lemma 6. *If t_e is a time after the deadlines of all jobs, $\beta(t_e) \leq 2^{2\alpha-1}\alpha^2 E_{\mathrm{Opt}}$.*

Proof. We first bound E_{Opt}. Let S be a subset of I, the input job set. We say S is minimally infeasible if S is infeasible, but any proper subset of S is feasible, using a speed-1 processor. The union of spans of jobs in a minimally infeasible job set, $span(S)$, forms a continuous time interval (otherwise one of the sub-intervals alone must be infeasible). Let $\mathcal{M}$ be the set of all minimally infeasible subsets of I. Let $span(\mathcal{M})$ be the union of all spans of those subsets within $\mathcal{M}$. A job j is "overloaded" if its span is in $span(\mathcal{M})$ completely, and "underloaded" if otherwise. So all jobs in $\mathcal{M}$ are overloaded. In [8], it is shown that (1) all type-0 jobs are overloaded, and (2) underloaded jobs do not affect the feasibility of job sets. Because of the latter, OPT must maximize the amount of work completed in overloaded jobs.

Let $\mathcal{M}_0$ be a minimal subset of $\mathcal{M}$ with $span(\mathcal{M}_0) = span(\mathcal{M})$. The spans of minimally infeasible subsets in $\mathcal{M}_0$ must all have different start times in their spans, otherwise one of them can be removed from $\mathcal{M}_0$, so $\mathcal{M}_0$ is not minimal. So let $\mathcal{M}_0 = \{S_1, S_2, \ldots\}$, where the start time of span of S_1 is earlier than that of S_2, etc. The span of S_i cannot overlap with the span of S_{i+2} or any later subsets, since otherwise S_{i+1} can be removed from $\mathcal{M}_0$. It is thus feasible to complete using one processor all jobs in S_1, S_3, ... except the smallest job in each subset; and to complete using the other processor all jobs in S_2, S_4,

... except the smallest job in each subset, leading to an amount of work no less than half of the total length of $span(\mathcal{M})$ (which we will denote by $|span(\mathcal{M})|$). The minimum energy schedule for OPT to complete this amount of work within a time period of $|span(\mathcal{M})|$ is to spread it over all the time in two processors. This results in speed 1/4 throughout the span, so $E_{\text{Opt}} \geq 2|span(\mathcal{M})|/4^\alpha$. Recall that $\beta(t_e)$ is α^2 times the amount of type-0 work completed by OAT. Since all type-0 work are overloaded, they must be executed in $span(\mathcal{M})$, resulting in $\beta(t_e) \leq \alpha^2|span(\mathcal{M})| \leq 2^{2\alpha-1}\alpha^2 E_{\text{Opt}}$ as claimed. □

To continue with the proof we need the definition of $\phi(t)$. Recall that the plan of OA defines critical times $c_0, c_1, \ldots$; and during each critical interval $[c_i, c_{i+1}]$, OA plans to use constant speed ρ_{i+1}. Let $w_{OAT}(i)$ be the amount of unfinished work under OAT in jobs with deadlines in $(c_{i-1}, c_i]$, according to the schedule if no more jobs are to be released after t. Let $w_{OPT}(i)$ be the amount of unfinished type-1 work under OPT in jobs with deadlines in $(c_{i-1}, c_i]$. Then

$$\phi(t) = \sum_{i \geq 1} (\min\{\rho_i, 1\})^{\alpha-1} (\alpha\ w_{OAT}(i) - \alpha^2 w_{OPT}(i)) \ .$$

We analyze the rate of change in terms of the inequality $E_{\text{OAT}}(t) + \phi(t) - \beta(t) \leq 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)$, when no job is released.

Lemma 7. *When no job arrives,* $E_{\text{OAT}}(t)' + \phi(t)' - \beta(t)' - 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)' \leq 0$.

Proof. Without new jobs, the expected schedule used by OAT is not changed, so $\beta(t)' = 0$. OAT runs at speed $s = \min\{\rho_1, 1\}$, so $E_{\text{OAT}}(t)' = s^\alpha$. For OPT, assume its two processors run at speed s_1 and s_2, so $E_{\text{Opt}}(t)' = s_1^\alpha + s_2^\alpha$. We need to bound $\phi(t)'$ from above. The function $\phi(t)$ consists of two sets of components, one for $w_{OAT}(i)$ and the other for $w_{OPT}(i)$. For $w_{OAT}(i)$, only $w_{OAT}(1)$ is changing, at a rate of $-s$. For $w_{OPT}(i)$, we do not know which i corresponds to the running jobs in the two processors, but $i = 1$ has the largest scaling factor, which results in the largest change of $\phi(t)$. So we have $\phi(t)' \leq s^{\alpha-1}(-\alpha s + \alpha^2(s_1 + s_2)) = -\alpha s^\alpha + \alpha^2 s^{\alpha-1}(s_1 + s_2)$. Therefore,

$$\begin{aligned}
&E_{\text{OAT}}(t)' + \phi(t)' - \beta(t)' - 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)' \\
&\leq s^\alpha - \alpha s^\alpha + \alpha^2 s^{\alpha-1}(s_1 + s_2) - 2^{\alpha-1}\alpha^\alpha(s_1^\alpha + s_2^\alpha) \\
&= \frac{(1-\alpha)s^\alpha + 2\alpha^2 s^{\alpha-1}s_1 - 2^\alpha\alpha^\alpha s_1^\alpha}{2} + \frac{(1-\alpha)s^\alpha + 2\alpha^2 s^{\alpha-1}s_2 - 2^\alpha\alpha^\alpha s_2^\alpha}{2} \\
&= (s_1^\alpha f(s/s_1) + s_2^\alpha f(s/s_2))/2 \ ,
\end{aligned}$$

where $f(z) = (1-\alpha)z^\alpha + 2\alpha^2 z^{\alpha-1} - 2^\alpha\alpha^\alpha$. We note that $f(0) = -2^\alpha\alpha^\alpha < 0$, and when $z \to \infty$, $f(z) \to (1-\alpha)z^\alpha < 0$. For maximum value, we set $f'(z) = (1-\alpha)\alpha z^{\alpha-1} + 2\alpha^2(\alpha-1)z^{\alpha-2} = 0$, which implies that $z = 2\alpha$, and $f(z) = (1-\alpha)(2\alpha)^\alpha + 2\alpha^2(2\alpha)^{\alpha-1} - (2\alpha)^\alpha = 0$. So $f(z)$ is non-positive for any $z \geq 0$, concluding our proof. □

Finally, we note that the proof in [8] can be used directly to show the following lemma, which concerns with how the release of jobs (rather than the running of jobs) affects $\phi(t)$ and $\beta(t)$.

Lemma 8. *At the time when a job is released,* $\Delta\phi(t) - \Delta\beta(t) \leq 0$.

References

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 621–633. Springer, Heidelberg (2006)
2. Albers, S., Muller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proc. SPAA (2007)
3. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. Manuscript
4. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. In: Proc. FOCS, pp. 520–529 (2004)
5. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. SODA, pp. 805–813 (2007)
6. Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D.: On-line scheduling in the presence of overload. In: Proc. FOCS, pp. 100–110 (1991)
7. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
8. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.: Energy efficient online deadline scheduling. In: Proc. SODA (2007)
9. Grunwald, D., Levis, P., Farkas, K.I., Morrey, C.B.: Policies for dynamic clock scheduling. In: Proc. OSDI, pp. 73–86 (2000)
10. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
11. Koren, G.: Competitive On-Line Scheduling for Overloaded Real-Time Systems. PhD thesis, New York University (1993)
12. Koren, G., Shasha, D.: MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. Theor. Comput. Sci. 128(1&2), 75–97 (1994)
13. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proc. SOSP, pp. 89–102 (2001)
14. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proc. OSDI, pp. 13–23 (1994)
15. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. FOCS, pp. 374–382 (1995)