Scheduling Unit Tasks to Minimize the Number of Idle Periods: A Polynomial Time Algorithm for Offline Dynamic Power Management

Philippe Baptiste*

October 7, 2005

Abstract

Power Management policies aim at reducing the amount of energy consumed by battery operated systems, while keeping the overall performance high. In this paper we focus on shut-down mechanisms that put a system into a sleep state when it is idle. A very small amount of energy is consumed in this state but, a fixed amount of energy is required when moving the system from the sleep state to the active state. The offline version of this problem consists in scheduling a set of unit execution tasks, with release dates and deadlines, on a single machine in order to minimize the number of idle time periods. We show that this problem can be solved in polynomial time by Dynamic Programming.

1 Introduction

Power Management policies aim at reducing the amount of energy consumed by battery operated systems, while keeping the overall performance high. Two mechanisms are used to save energy: speed scaling and shut-down mechanisms.

- *Speed scaling* mechanisms rely on the fact that the speed (or frequency) of processors can be changed online. As the energy required to perform the same task increases with the speed of the processor, speed scaling policies tend to slow down the processor as much as possible while performing all tasks within a reasonable delay.
- *Shut-down* mechanisms put a system into a sleep state when it is idle. A very small amount of energy is consumed in this sleep state but a fixed amount of energy is required when moving the system from the sleep state to the active state.

More complex systems play with several sleep states simultaneously together with speed scaling techniques. We reefer to [4] for an up-to-date survey on algorithmic problems in power management.

We assume the following model for the shut-down power management problems considered in this paper. n tasks $1, ..., n$ are to be processed on the processor and each task i is associated with a release date r_i, a deadline d_i and a processing time p_i. Tasks can be interrupted at any integer time point (i.e, preemption is allowed). The energy required by a transition from the sleep state to the active state is L times the energy required by a unit task. All data are integer and the objective is to minimize the total energy required to process all tasks. Informally, we look for a schedule with few and long idle time periods. Note that when L is 1, the problem reduces to computing a preemptive schedule of tasks, under time-windows constraints, with a minimum number of idle time intervals.

As tasks to be performed are often not known in advance, the online version of this problem is especially important. The optimal competitive ratio that can be achieved by any online algorithm is 2 [3]. There are few results on the offline case. Deciding whether there is a schedule with *no* idle times can be done in polynomial time (see [2] for the most recent results in this field). As reported in [5] as well as in a Dagstuhl 2002 Online algorithms

*Ecole Polytechnique, Laboratoire d'Informatique CNRS LIX, F-91128 Palaiseau, Philippe.Baptiste@polytechnique.fr

SODA '06, January 22-26, Miami, FL
 ISBN 0-89871-605-5/06/01

workshop, the complexity status of the general offline problem remains unknown. S. Irani and K. Pruhs [4] report that

> " [this problem] is the "most intellectually intriguing question related to speed scaling / power down strategies [...] Many seemingly more complicated problems in this area can be essentially reduced to this problem, so a polynomial time algorithm for this problem would have wide application."

In this paper we focus on the unit execution time tasks case (i.e., $\forall i, p_i = 1$) and we introduce a simple dominance property (Section 2) for optimal schedules. We introduce in Section 3 a decomposition scheme based on ideas introduced in [1] for scheduling equal length jobs on parallel machines. The decomposition leads to a simple polynomial time dynamic programming algorithm.

2 Dominance Properties

The following proposition shows that there are "few relevant time points" at which tasks start in some optimal schedule.

PROPOSITION 2.1. *There is an optimal schedule in which, for any task i, the distance between the starting time of i and one of the release dates or deadlines is at most n*

Proof. Among optimal schedules, consider the schedule $\mathcal{S}$ that lexicographically minimizes the vector $(t_1, ..., t_n)$ of starting times. Assume there is a task i such that $\forall j, |t_i - r_j| > n$ and $\forall j, |t_i - d_j| > n$. Let then s denote the largest time points before t_i such that $[s-1, s)$ is idle and let e denote the smallest time points after t_i such that $[e, e+1)$ is idle. The interval $[s, e)$ is full and there are $e - s \leq n$ tasks in this interval. As $\forall j, |t_i - r_j| > n$, tasks that start in $[s, e)$ do not start at their release dates. As $\forall j, |t_i - d_j| > n$, tasks that start in $[s, e)$ are not completed by their deadlines. So we can modify the schedule $\mathcal{S}$ by moving all tasks in $[s, e)$ from 1 unit either to the right (schedule $\mathcal{R}$) or to the left (schedule $\mathcal{L}$).

As the initial schedule $\mathcal{S}$ is an optimal schedule that lexicographically minimizes the vector of starting times, the cost of the schedule $\mathcal{L}$ is strictly greater than the cost of $\mathcal{S}$ (because $\mathcal{L}$ is lexicographically better than $\mathcal{S}$). Hence, the idle period before $[s, e)$ on $\mathcal{S}$ is larger than L (it can be infinite) while the idle period after $[s, e)$ on $\mathcal{S}$ is smaller than L. So, when moving from $\mathcal{S}$ to $\mathcal{R}$, the cost associated to the right idle interval is decreased while the cost associated to the left idle interval does not change. Hence the schedule $\mathcal{R}$ strictly improves on $\mathcal{S}$. Contradiction. ∎

Thanks to Proposition 2, we know that there is an optimal schedule in which start and completion times belong to the set Θ.

$$\Theta = \bigcup_i \{r_i - n, ..., r_i + n\} \cup \{d_i - n, ..., d_i + n\}$$

Note that there is a quadratic number of time points in the set Θ.

3 Decomposition

To simplify the presentation, we define the *relative cost of a schedule over an interval* $[s, e)$ as the cost of a schedule built as follows: schedule a fake job in $[s-1, s)$, follow the initial schedule from a to b and finally schedule a fake job in $[e, e+1)$. Informally speaking, these two jobs are used to take into account idle intervals right after s and right before e.

In the following, we assume that jobs are sorted in non decreasing order of deadlines, i.e., $d_1 \leq d_2 ... \leq d_n$.

We are now ready to define a state of the dynamic program. For any integer $k \leq n$, let $F_k(s, e)$ be the minimal relative cost over the interval $[s, e)$ among all schedules of the jobs $\{i \leq k, s \leq r_i < e\}$ such that

1. the machine is idle before s and after e,
2. starting times and completion times are in Θ.

If no such schedule exists, $F_k(s, e) = \infty$.

Note that the optimum of our initial problem is exactly $F_n(\min \Theta - L, \max \Theta + L) - 2L$ (the "$-2L$" is there to take into account the fake jobs that always create two interruptions with cost L that must not be taken into account). We also have

$F_0(s,e) = \min(L, e-s)$. The following proposition shows how to recursively compute $F_k(s,e)$ (see Figure 1) .

PROPOSITION 3.1. *If job $k > 0$ is such that $r_k \notin [s,e)$ then $F_k(s,e) = F_{k-1}(s,e)$. If $r_k \in [s,e)$ then $F_k(s,e)$ equals F' where*

$$F' = \min_{t,t+1 \in \Theta, s \leq t < e} F_{k-1}(s,t) + F_{k-1}(t+1,e)$$

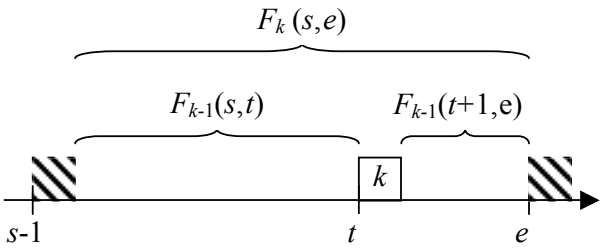

Figure 1: Decomposition Scheme

Proof. We first prove that $F_k(s,e) \leq F'$. If $F' = \infty$ this is obviously true. Now consider a schedule $\mathcal{S}$ that realizes $F_{k-1}(s,t)$ and a schedule $\mathcal{E}$ that realizes $F_{k-1}(t+1,e)$. We build a schedule as follows: from time s to t, follow $\mathcal{S}$, schedule k in $[t,t+1)$, from time $t+1$ to e, follow $\mathcal{E}$. The cost of this schedule is exactly F' Moreover, it contains all jobs in $\{i \leq k, s \leq r_i < e\}$, the machine is idle before s and after e, and starting times and completion times of jobs belong to Θ. So $F_k(s,e) \leq F'$.
We now prove that $F_k(s,e) \geq F'$. If $F_k(s,e) = \infty$ then our claim holds. Now, assume it is finite. As $k \in \{i \leq k, s \leq r_i < e\}$, job k is scheduled in all schedules that realize $F_k(s,e)$. Among such schedules, let $\mathcal{X}$ denote one in which the starting time of job k is maximal. We claim that all jobs in $\{i \leq k, s \leq r_i < e\}$ that are released before or at t are completed at t. If this were not the case, we could exchange such a job with k and thus we would have a feasible schedule with the same cost as before. This would contradict the fact that the starting time of k is maximal. So the restriction $\mathcal{S}$ of $\mathcal{X}$ to $[s,t)$ is a schedule that meets all constraints related to $F_{k-1}(s,t)$. Hence its cost is greater than $F_{k-1}(s,t)$. Similarly, the restriction $\mathcal{E}$ of $\mathcal{X}$ to $[t+1,e)$ is a schedule that meets all constraints related to $F_{k-1}(t+1,e)$. To conclude the proof, note that the cost of $\mathcal{X}$ is the sum of the costs of $\mathcal{S}$ and $\mathcal{E}$. ∎

The relevant values for s and e are exactly those in Θ. The values of $F_k(s,e)$ are stored in a multi-dimensional array of size $O(n^5)$ (n possible values for k, n^2 possible values both for s and e). Our algorithm works as follows:

- In the initialisation phase, $F_0(s,e)$ is set to $\min(L, e-s)$ for any values s, e in Θ ($s \leq e$).
- We then iterate from $k = 1$ to $k = n$. Each time, F_k is computed for all the possible values of the parameters thanks to Proposition 3.1, and to the values of F_{k-1} computed at the previous step.

The initialisation phase runs in $O(n^4)$ because the size of Θ is upper bounded by $O(n^2)$. Afterwards, for each value of k, $O(n^4)$ values of $F_k(s,e)$ have to be computed. For each of them, a maximum among $O(n^2)$ terms is computed (because there are $O(n^2)$ possible values for $t_k \in \Theta$). This leads to an overall time complexity of $O(n^7)$. A rough analysis of the space complexity leads to an $O(n^5)$ bound but since, at each step of the outer loop on k, one only needs the values of F computed at the previous step $(k-1)$, the algorithm can be implemented with 2 arrays of $O(n^4)$ size: one for the current values of F and one for the previous values of F. (To build the optimal schedule, all values of $F_k(s,e)$ have to be kept; hence the initial $O(n^5)$ bound applies.)

4 Conclusion and Open Problems

A natural generalization of our problem is to consider the situation in which task i has an arbitrary processing time $p_i \in \mathbb{N}$ and where preemption is allowed. Each task can be decomposed in p_i unit execution time tasks (with the same time windows as for the initial task) and thus, we have a straightforward pseudopolynomial time running in $O((\sum p_i)^7)$ for this problem. This also leads to a simple FPTAS for general processing times. We still do not know whether this problem can be

solved in strongly polynomial time. A straightforward adaptation of our algorithm seems unlikely as the set of time point Θ to consider is not polynomially bounded any more.

Acknowledgments

The author would like to thank Maxim Sviridenko who introduced this problem during the workshop on "Models and Algorithms for Planing and Scheduling" (may 2005, Siena).

References

[1] Philippe Baptiste. Scheduling equal-length jobs on identical parallel machines. *Discrete Applied Mathematics*, 103, 2000.

[2] Philippe Chretienne. On the no-wait single machine scheduling problem. In *Proceeding of the 7th Conference on Models and Algorithms for Planning and Scheduling*, pages 76–79, June 2005.

[3] Sandy Irani and Anna Karlin. *Approximation Algorithms for NP-complete Problems*, chapter Online Computation, pages 521–559. PWS Publishing Company, 1997.

[4] Sandy Irani and Kirk Pruhs. Algorithmic problems in power management. volume 36, pages 63–76, New York, NY, USA, 2005. ACM Press.

[5] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 37–46, 2003.